Delft University of Technology
Master of Science Thesis in Embedded Systems

Fault-tolerance in decentralized motorway traffic management

Yann Rosema

Embedded Networked Systems

TU Delft Delft University of Technology

Fault-tolerance in decentralized motorway traffic management

Master of Science Thesis in Embedded Systems

Embedded and Networked Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Yann Rosema
Y.Rosema@student.tudelft.nl
yann.rosema@hotmail.com
TU Delft Student Number: 4218248

17-01-2020

Author
Yann Rosema (Y.Rosema@student.tudelft.nl)
(yann.rosema@hotmail.com)
(**TU Delft Student Number:** 4218248)

Title
Fault-tolerance in decentralized motorway traffic management

MSc Presentation Date
31-01-2020

Graduation Committee

dr. ir. Fernando Kuipers (chairman)	Delft University of Technology
dr. ir. Neil Yorke-Smith	Delft University of Technology
dr. ir. Gerard Avontuur	Rijkswaterstaat
ir. Jorik Oostenbrink	Delft University of Technology

Abstract

In the Dutch motorway traffic management (MTM) system used by Rijkswaterstaat, a central system and multiple outstations alongside the road communicate with each other to keep highways safe. This makes the MTM system a form of distributed flow control system. However, Rijkswaterstaat faces 2 problems with regards to traffic control: the control system is hierarchical leading to a single point of failure and communication failures between the central system and the outstations lead to unsafe situations.

A solution to both problems is the use of a multi-agent system (MAS). However, no fault-tolerant MAS solution exists for the type of flow network that the highway network is. We therefore design our own fault-tolerant MAS solution for flow networks where the failure of a controller does not change the characteristics of the flow network.

Our design associates an agent with every outstation in the network giving it the ability to make its own decisions. The communication protocol used by these agents to communicate with each other is capable of detecting communication failures with outstations and rearrange them so that the control action remains unchanged. This results in a system which keeps the highway safe when one or more outstations or their communication abilities fail.

Preface

This thesis concludes my master Embedded Systems at Delft University of Technology. It is the result of 1.5 years of work to find a solution for a problem faced by Rijkswaterstaat where I had to opportunity to undertake an internship. Finding the right topic was hard but thanks to my supervisors at both the TU Delft and Rijkswaterstaat I managed to find one that was interesting and challenging. The research question I settled for fulfilled both my desires to work on a high-level and practical design.

I would like to thank my supervisors at Rijkswaterstaat, Gerard Avontuur and Christiaan van Deuzen, for their help and guidance during my internship. I also wish to thank my supervisors at Delft University of technology, Fernando Kuipers and Jorik Oostenbrink for the very helpful feedback they gave during this project.

I would like to thank my colleagues at Rijkswaterstaat for their time when I had questions and for making me feel very welcome during my internship. Lastly, I would like to thank my friends and family for their support and especially my mother for always finding the time to give me writing feedback when I needed it.

Yann Rosema

Delft, The Netherlands
17th January 2020

Contents

Chapter 1

Introduction

The level of intelligence of road going vehicles has come a long way in the last few years. The first fully autonomous cars are now in their prototyping phase and autonomous driving on the highway is already available in some production cars. Although the car industry has made huge steps forward, the infrastructure (i.e. the road system) used by these cars has mostly stayed the same. In order to get the most out of autonomous cars, these smart vehicles would need to communicate with the road system in some way. This can be useful for both vehicles and infrastructure: the cars can get traffic updates allowing them to, for example, reroute their trajectory if needed and the road system would be able to collect data from the autonomous cars in order to improve traffic flow.

In the Netherlands, the highways are under the supervision of Rijkswaterstaat. Since the early nineties, Rijkswaterstaat has used a *motorway traffic management* (MTM) system to control traffic flow on the highway. This system consists of a central system, adaptable signage displays, traffic sensors and outstations. These elements are organised in a hierarchical structure: the central system communicates with multiple outstations, who, in turn, communicate with multiple displays and sensors. Thus, the outstations function as a translator between the central system and the sensors/displays. With this MTM system, Rijkswaterstaat can show traffic symbols based on the state of the traffic or based on anomalies (e.g. road work, an accident) in order to warn the drivers.

The elements of the MTM system have not changed significantly since their first implementation and become outdated. Recently, Rijkswaterstaat has decided to give some parts of the MTM system an update. They launched the iWKS project to completely redesign the outstation hardware in order to make it more modular, manageable and maintainable. While the functionality of the outstation has stayed the same, the new hardware presents an opportunity for the development of new traffic solutions.

Although the outstations have been revamped, the whole MTM system is still not up to speed with the car industry. If Rijkswaterstaat gives the task of communicating with autonomous cars to the current MTM system, a few problems arise. First, the current system has a hierarchical structure with a single point of failure, the central system. If the central system, or the communication lines with the central system fail, it becomes impossible to communicate with the autonomous cars. Even in the current situation, Rijkswaterstaat can lose

the ability to update the adaptable signage displays, which can lead to unsafe situations. Second, there is the possibility that the amount of data that needs to be exchanged between cars and central system exceeds the central system's or the network's capacity. This would mean that the central system and its connections with the rest of the network would need to be updated, which is an expensive undertaking.

Besides the problems arising as a result of the current system's structure, Rijkswaterstaat copes with another problem. Currently, the communication between central system and outstation frequently fails. This can be due to a fault in the communication line between both elements or it can be due to a failure of the outstation itself. In both cases, Rijkswaterstaat loses the ability to update the displays, which can lead to unsafe situations.

Thus, the MTM system copes with two problems: first, the hierarchical structure of the MTM system has a single point of failure and might not be able to deal with large amounts of data. Second, the MTM system experiences frequent failures between central system and outstations.

1.1 Problem statement

Since the traffic on the highway can be viewed as a flow of cars, the highway network can be seen as a flow network. The outstations, which control the traffic flow, can be seen as flow controllers. This means that the MTM system is, in essence, a distributed flow controller.

One way to implement a network of flow controllers, is using a *multi-agent system* (MAS). A MAS, is a system of nodes (agents) in a network capable of making their own decisions and capable of communicating with each other. All the agents in the network work together towards a set of goals determined by the designer of the system or by the agents themselves. This approach would solve the problems related to the hierarchical structure of the current MTM system, since no agent is the sole decision-maker of the network.

However, unlike most flow networks controlled by distributed controllers, like the power grid, the highway network has the property that the traffic can still flow even when an outstation fails. To the best of our knowledge, no fault-tolerant MAS solution exists for these types of flow networks and their associated controller network. This leaves the second problem faced by the current MTM system unsolved.

We therefore design our own motorway traffic control MAS that is tolerant to outstation/controller failures. We assume that, in case of such a failure, the outstation is unable to perform its control tasks and unable to communicate with other outstations. The system should then be able to re-evaluate its current actions in order to maintain its control goals.

1.2 Contribution

We design our own MAS system capable of replacing the current MTM system while being tolerant to outstation failures. Our design associates an agent with every outstation in the MTM network. Each outstation has the capability to make its own decisions and the capability to communicate with other outsta-

tions. Using both these capabilities, outstations are able to generate the symbols for the displays associated with them. Furthermore, using this communication, outstations are also able to detect faulty outstations amongst themselves. When a failure is detected, a fault-recovery protocol kicks in that reconfigures the relations between the outstations. This reconfiguration makes sure that the displays of the outstations that are still operational, maintain a safe sequence of symbols. The result is a MAS that is fault-tolerant to any number of failures and does not have a hierarchical structure.

1.3 Organisation

Chapter 2 gives an overview of the current MTM system and defines some concepts used throughout this thesis. We then start our search for a new MTM system by looking at the failures currently sustained by the outstations. Chapter 3 presents a failure analysis of the faults sustained over a whole year. We study where and when these faults occur in order to determine their causes. We present related work in chapter 4 and, based on that, design our own fault-tolerant multi-agent system in chapter 5. In order to validate the design, we create an implementation of the design and test it in chapter 6. Finally, we draw conclusions and make recommendations on were to focus future research in chapter 7.

Chapter 2

The MTM system

This chapter describes the components of the current Dutch *motorway traffic management* (MTM) system and their function. We also give a few definitions used throughout this thesis.

2.1 System components

Figure 2.1 gives an overview of the hierarchical structure of the whole system.

The technical aspect of the information chain consists of the following elements:

- The central system: in the context of our work, it is the hard- and software combination that makes the decisions about what symbols to display and where do to so. Some of these decisions are taken automatically and the other are taken manually by the traffic manager. The system receives data about traffic density from outstations and sends them instructions in return. The central system consists of two main components:

  The *transaction oriented processor* (TOP): this piece of software interprets the decisions given by the traffic manager and selects the appropriate symbols according to a set of rules (e.g. when a red cross needs to be placed above the road at location n, place an arrow to divert the traffic to another lane at location $n - 1$).

  The *front end processor* (FEP): when the TOP has decided where to show what symbols, it sends that information down to the FEP. Multiple FEP's can be connected to a TOP. The FEP then interprets the symbols and transforms them into messages for the outstations according to a proprietary protocol.

- The outstation: as mentioned in the previous item, the FEP communicates with one or more outstations. These devices are located alongside the Dutch highways, spaced approximately 500 to 700 meters apart. Their job is to send data about traffic speed and density to the central system and to update the adaptable signage displays according to the central system's decisions. In order to do these jobs the outstation has two interfaces: a sensor module for the traffic data and an actuator module to connect to the adaptable signage displays.

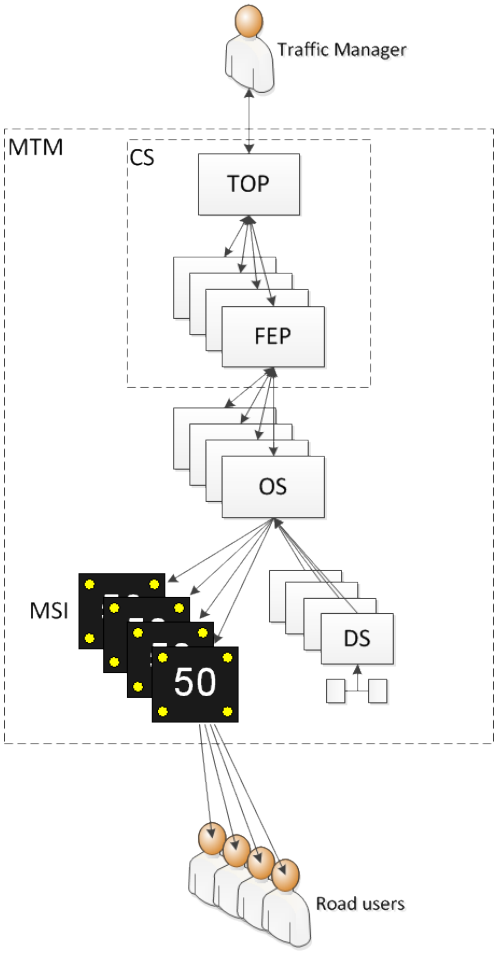


Figure 2.1: **Communication chain involving the central system and the outstations. All elements combined, constitute the Motorway Traffic Management (MTN) system. The Central System (CS), which includes the Transaction Oriented Processor (TOP) and the Front End Processor (FEP) translates a command from the traffic manager for the Outstations (OS). The outstation communicates with the Adaptable Signage (MSI) to show symbols on the road and with the Detector Station (DS) to receive traffic state information.**

- The *adaptable signage displays* (Matrixsignaalgeverinstallatie or MSI): these devices can be found above the road displaying various symbols to inform the driver about the traffic conditions ahead.
- The detector stations: these devices are located inside or close to an outstation and are in charge of reading out the sensors in the highway. They then interpret the result and generate data in the form of traffic speed and intensity data points that are transmitted to the outstation.

2.2 System behaviour

The main task of the MTM system is to display traffic symbols on highways in order to warn the drivers for traffic conditions ahead. Two triggers can lead to an update of the MSIs.

The first trigger is a congestion trigger which results from the data point generated by the detector stations. When the outstation receives such a data point, the values are compared to a threshold. If the threshold is crossed, the outstation notifies the central system of the congestion.

The second trigger is the result of an intervention by the traffic manager who can add custom traffic restrictions when he sees fit. This is mostly the case when an accident occurs or when road works have been scheduled.

Both triggers result in the same action: recalculating the symbols to display on the MSIs. The central system performs this action using a set of predefined rules that determine the relations between triggers and MSIs, and between the MSIs themselves. After the calculation, the central system sends an updated version of the MSIs to the outstations. The outstations are then responsible for actuating the MSIs with the new symbols.

In case of a faulty outstation or network link, the central system is notified as a periodic polling system senses the availability of the outstations. However, no action is undertaken by the central system to make sure the sequence of symbols displayed remains coherent with the rules. This poses a problem because the absence of a symbol in a sequence can lead to unsafe situations on the highway.

2.3 Measure and action

We set a few definitions used in the rest of this thesis.

A traffic symbol always has a reason for being displayed, this reason is called the measure. A measure can span multiple outstations with different symbols. There are two types of measures: AID measures and central measures. The first type is the result of a local congestion trigger, the second is the result of a central trigger.

A measure consists of multiple parts, of which we will retain one, the basic measure. The basic measure is the part of the measure that is specified by the trigger. All the other parts of the measure are results of the rules set by Rijkswaterstaat to safely continue or lead-in the basic measure. An example of a measure is given in figure 2.2. The measures, are combined into a single set of symbols by the outstation which displays them on the MSIs. This set of symbols is called the outstation's action.

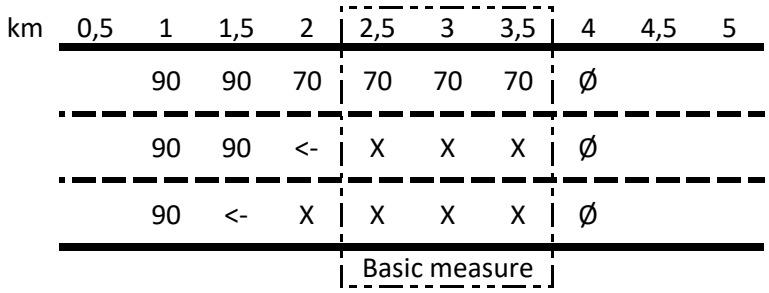


Figure 2.2: **Example of a measure with the basic measure framed by a dotted line. Traffic travels from left to right and is diverted from the two lowest lanes, which have been crossed off by the basic measure.**

Chapter 3

Data analysis

The goal of the MTM system is to improve traffic safety and flow by displaying measures based on traffic sensor data. The current communication failures have two main consequences for Rijkswaterstaat: the inability of displaying traffic control measures and the loss of sensor data.

The first consequence is a measurable problem. In 2017, 1 613 measures could not be applied to the MSIs. Although compared to the 5.1 million total measures placed in 2017, 1 613 only represents 0.032% of all measures, it still presents a dangerous situation.

The second consequence is a result of the current architecture of the MTM system: during the original design, consistency and continuity of sensor data were not of importance. However, nowadays data are the fuel of the economy [7] and thus loss of data is a real problem for Rijkswaterstaat.

Before designing a new system tolerant to outstation communication failures, we need to know the origin of the communication failures. Three possibilities are direct causes:

- outstation power failures
- hard and software failures
- connection failures

Furthermore, we test three indirect causes, which are origins of faults that lead to direct causes:

- road work
- system wear
- peripheral interference

3.1 Data logging

The data for this analysis are extracted from the 2017 logs of the TOP's of the 5 traffic control centres (i.e. central systems). Therefore the data contain information about the FEP's, outstations, MSIs and detector stations. Of the

128 291 737 messages present in the logs, 33 817 indicate a communication failure between an outstation and a central system.

These messages specify the date and time of the event, an identification number of the incident and the location of the outstation that lost communications. The messages can be grouped into 15 228 events: an instance in time where one or more outstations lost communications. This means that, on average, an event consists of 2.22 outstations affected by a communication failure.

Of the 6 133 outstations operating in 2017, 4 422 suffered from communication failure. This is quite alarming as it shows that 72% of the outstations in operation have experienced communication failure at least once in 2017.

Another alarming fact is that, given the 33 817 cases of communication failure and the only 6 133 outstations, an outstation lost communications, on average, 5.5 times over the year. However, this average is very unrepresentative of what happens since some outstations appear more than 500 times in the messages. This means that 202 outstations (or 3.3% of all outstations) are responsible for 50% of the communication failures.

Another interesting metric that emphasizes the size of communication failures, is the amount of messages per unit of time. As mentioned earlier, there where 33 817 communication loss messages in total over 2017, which is equivalent to 92.65 messages per day or 3.86 per hour. This amounts to 1.74 events per hour in 2017, which is significant.

3.2 Power failures, hard and software failures and connection failures

To find out if power failures or hard and software failures are the causes of the communication failures between outstation and central system, we can check for a few correlations in the logs. We begin by noting that these failures only affect single outstations or outstations in the same cabinet. A cabinets is also called a *Wegkantstation* (WKS) at Rijkswaterstaat and, on average, a cabinet holds 1.5 outstations. Usually both the left and right side of the road have an outstation controlling traffic at the same point, so it makes more sense to put both outstations in the same cabinet. All outstations inside a WKS have a common power connection.

We can now divide the failures in events that concern single outstations, whole WKS's and multiple WKS's as shown in figure 3.1. It is clear, from this figure, that not all failures are caused by single outstations or WKS's failing. A large part of the failures concern multiple WKS's and are most likely caused by entire connection line failures. Although single outstation or single WKS failure do not represent all failures, it is still useful to see how and why these subsystems fail.

In order to limit the amount a power failures in the first place, WKS's are equipped with an *uninterruptible power supply* (UPS) which is able to supply power to the outstations and their peripherals for at least 2 hours after the main power supply has failed.

Usually, when a power failure occurs, the main power disconnects first which is notified to the central system with a "no power" flag. If the power failure lasts for more than 2 hours, eventually the UPS will also fail and the outstation

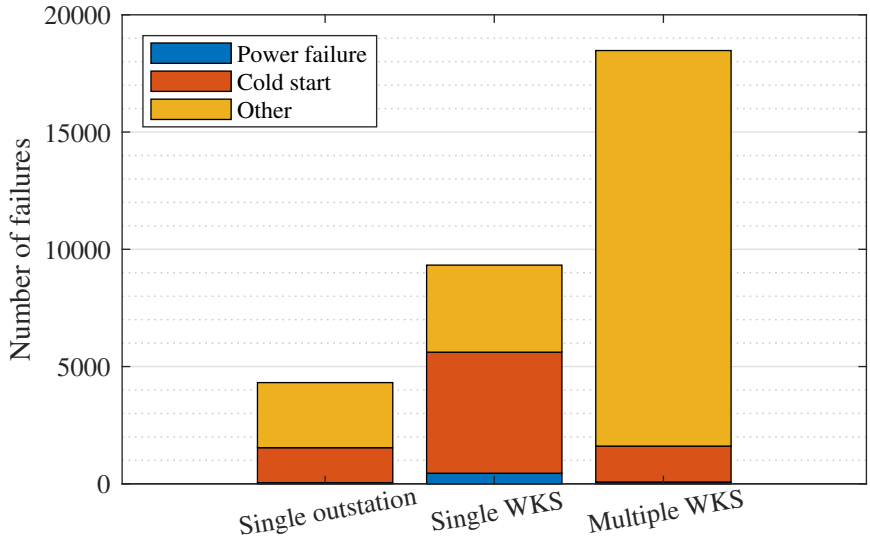


Figure 3.1: **Number of outstation communication failures per category and per cause.**

shuts down. As soon as the central system notices that the outstation is not responding, it create a notification of the failure. Using the "no power" flags, we where able to identify that 3.5% of the single outstation and single WKS failures are due to power failures.

Hardware issues need a technician to be resolved and, usually, requires a full power cycle. This results in a "cold start" message being sent to the central system upon reboot. If, for any other reason, a technician needs to perform a hard reset of the outstation, the resulting messages are exactly the same. Using the "cold start" messages we where able to identify that 47% of the outstation failure are due to a hardware failure or technician intervention.

When a software failure occurs, no special message is sent to the central system and no "cold start" message is send when the issue is resolved. These issues are therefore categorised as "other" failures together with other unknown direct causes.

Connection failures result in multiple WKS's disconnecting during the same event. This represents 68% of the outstation communication failures. There can be multiple causes that lead to a connection failure such as broken cables or disconnected plugs; both can be intentional or accidental. Unfortunately, the logs are not detailed enough to extract more data about this direct cause.

3.3 Road works

Rijkswaterstaat is responsible for the maintenance of the highways. Road works, called "changes" at Rijkswaterstaat, are logged to a database which is populated manually by Rijkwaterstaat's change managers. As these message are only meant to be read by people performing or preparing the changes, no "computer-friendly" template is used. This makes it difficult to identify the exact outstations that are, or might, be concerned by changes.

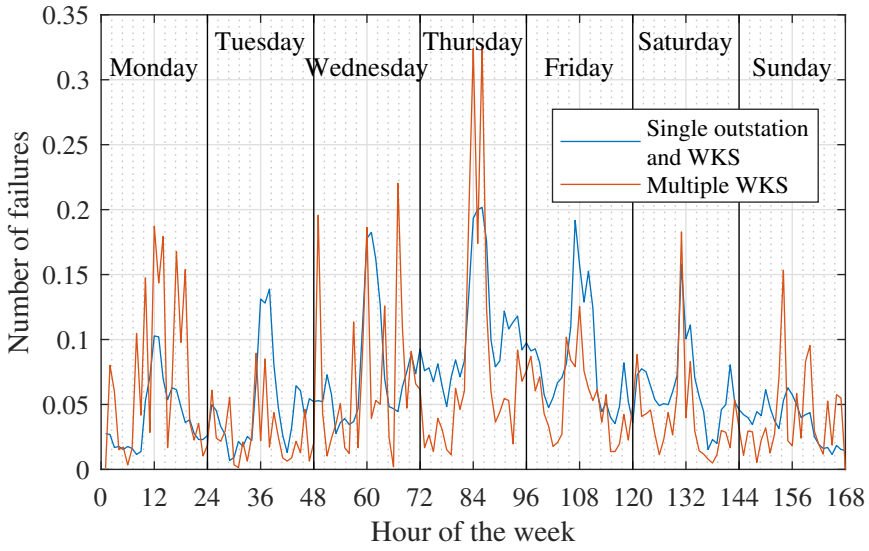


Figure 3.2: **Number of outstation communication failures per hour for single outstation and single WKS failures and for multiple WKS failures.**

Making things worse, in the current MTM system, outstations have a "local" mode which allows them to keep performing a limited set of actions when no communication with the central system can be established. Therefore, change managers often do not mention, in their change entry, which outstations are concerned with the change, as the outstations are expected to keep working in "local" mode.

Nonetheless, we were able to identify that at least 4.869 cases of communication failures are linked to road works. This number is quite low, as it amounts to only 14.4% of the total cases, and, since the database is not a reliable source of information, we need to look further.

Rijkswaterstaat has preferred timeslots between which they perform changes. The default hours to work alongside the road (where the outstations are located), are between 10:00 and 15:00 and for bigger projects, these works are moved to the night between 23:00 and 05:00. If we plot the distribution of the communication failures over a week, as shown in figure 3.2, we see that the correlation between communication failures and change hours is high.

The correlation coefficients between the moment the events take place and the time Rijkswaterstaat is working on or alongside the road are given in table 3.1. The correlation with the midday hours is high for single outstation and single WKS failure events and moderate for multiple WKS failure events. The midnight hours have a very low and negative correlation with both types of events. Evening works usually involve changes to the road surface itself instead of the infrastructure around it. Given that outstations are located on the side of the road, working during low traffic hours at midday suffices to perform outstation changes. This suggests that the midday road works are a likely cause of outstation failures, especially for single outstation and single WKS failures.

	single outstation and single WKS failure events	multiple WKS failure events
10:00 - 15:00	0.6000	0.3991
23:00 - 05:00	−0.1928	−0.1632

Table 3.1: **Correlation coefficients between the different types of events and the working hours of Rijkswaterstaat.**

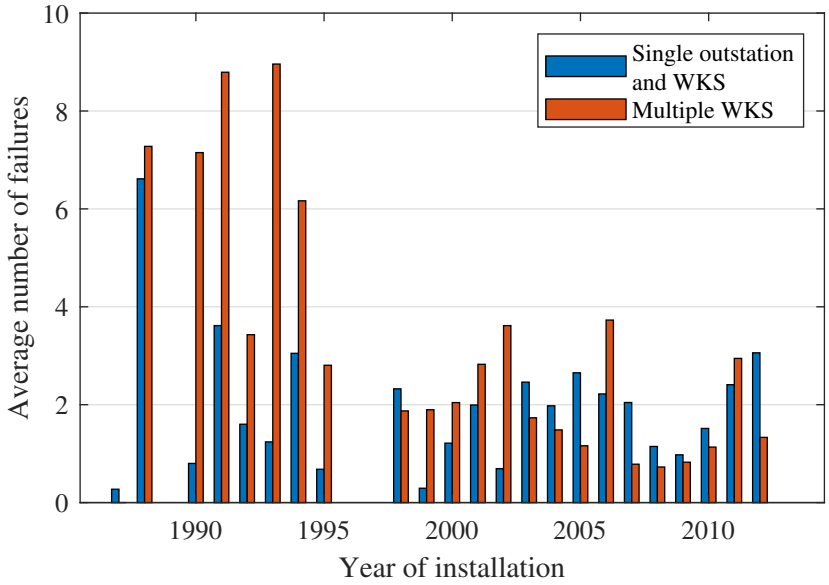


Figure 3.3: **Average amount of outstation communication failures per year of installation.**

3.4 System wear

The oldest outstations in the MTM system were placed in 1992 and some of those systems are still around. This old age can lead to failures due to wear of the outstation. This can be weather related wear, damage from an accident or hardware components that degrade over time, to name a few examples. We split this section in two parts: wear of the outstations and wear of the connection line.

Figure 3.3 shows the average number of times an outstation fails for single outstation and single WKS faults, per year of installation. The bar graph does not show a significant difference between young and old outstations except for outstation installed in 1993. These outstations failed on average over $6,5$ times in 2017. Still, this does not mean that outstations installed in 1993 are worse than others, since it might be that there were a high number of road works near outstations from 1993, for example.

Furthermore, their is no apparent decline in failures as the years progress in the bar graph. We therefore conclude that outstation or WKS wear is not a factor with a large impact on the amount of communication failures.

We can check for connection wear by looking at the installation year of the

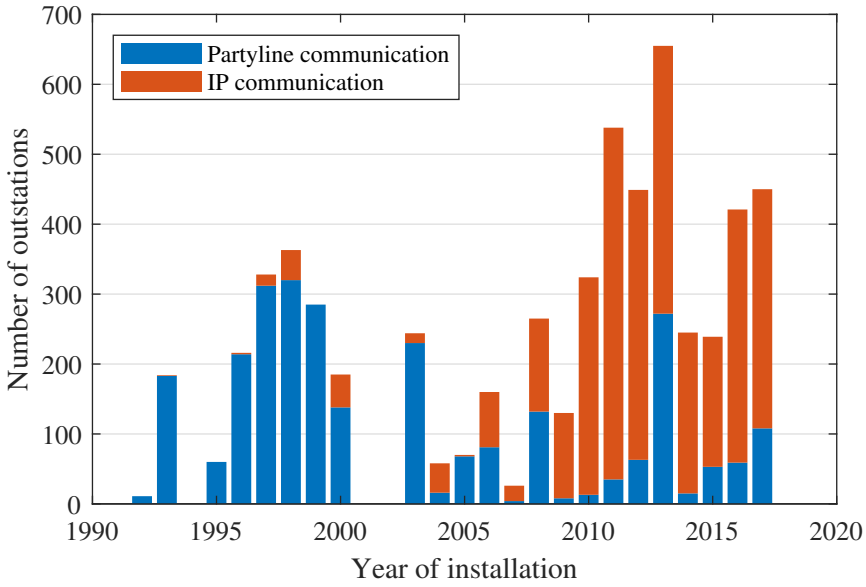


Figure 3.4: **Number of OS's per year of installation separated in the ones with fibre connection and copper connection.**

outstations failing due to a connection failure. This metric is shown in figure 3.3. In this case, there is a clear difference between outstations installed before 2002 and the ones installed after. Outstations installed before 2002 fail between 2 and 8 time more often than outstations installed after 2002. This could still be related to road works but given the range of years concerned in this huge disparity, another cause is more likely.

Before outstations were connected to a fibre-optic IP network, they used to be connected with the FEP's using serial copper partylines. Unfortunately, not all outstations have been updated to the fibre-optic network yet and, thus, there are still outstation using the copper connection lines. Figure 3.4 shows the number of outstations installed by year of installation and the type of connection used.

The correlation between the average number of failures of an outstation, due to a connection problem and the proportion of outstations installed on copper partylines is $\rho = 0.5472$, which indicates a moderate correlation. This result suggests that the use of the copper serial partyline connections leads to more failures than the use of fibre-optic IP connections.

3.5 Peripherals interference

As mentioned in the previous chapter, outstations communicate with MSIs and detector stations. Both theses systems operate separately from the outstation and only receive commands from the outstation. However, their is a possibility that a peripheral generates a failure at the outstation, for example, due to power issues, since MSIs and detector stations have the same power source as the WKS's. MSIs are not known for causing outstation problems, but, instead, shutdown gracefully when a failure is detected at an MSI. On the other

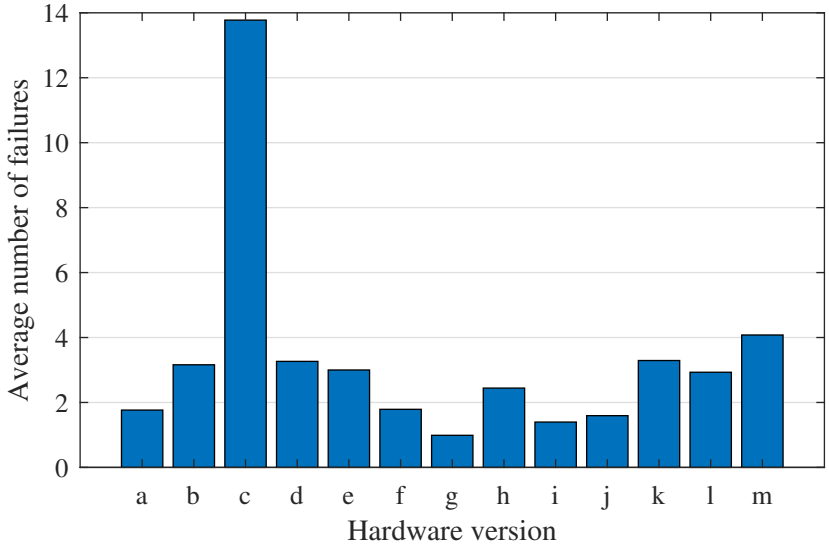


Figure 3.5: **Average number of failures per type of detector station.**

hand, there are cases where detector stations caused outstations to fail due to compatibility errors.

Figure 3.5 shows the average amount of failures per type of detector station. Detector station version c clearly stands out from the rest of the outstations. Upon inspection, we found that this was due to an isolated case where one WKS failed multiple times over 3 days with very small time intervals between failures. This is not representative of a structural fault related to this particular detector station version.

For the other detector stations, the average number of failures varies between 1 and 4. Although this is a big gap, the averages are evenly distributed over de detector station versions. We therefore cannot draw definitive conclusions about the involvement of detector station version differences in outstation failures.

3.6 Conclusion

We set out to test 3 possible direct and 3 indirect causes for the communication failures between central system and outstations.

Concerning direct causes, we managed to identify that power failures are the origin of a small part of the communication failures. Thanks to the UPS, power failures at an outstation are relatively rare (3.5%). Failures that recovered with a “cold start” message, signifying the system recovered from a hard reset, represent close to 50% of the single outstation and single WKS failures. Since the information contained in the logs is limited, other direct causes for the failure of single outstations and single WKS’s are unknown. We suspect that software failures and connection failures inside WKS’s make up the rest of the failures.

We were able to establish that 68% of all communication failures have a connection failure as the direct cause. Again, given the limited amount of information present in the logs, more specific causes are unknown.

Concerning indirect causes, we were able to identify 2 aspects that can be the origin of a large number of outstation faults. First, the correlation between the time of outstation failures and the moment road works are planned is apparent (0.6000 and 0.3991 during the midday hours). This suggests that road works play a large role in causing communication failures. Second, the correlation between the outstations involved in multiple WKS failures and older outstations, installed predominantly on copper connection, is significant. We therefore conclude that copper connections lead to an increase in communication failures.

The causes we identified will partly be fixed with the iWKS project, which serves as a platform for our new MAS design. The iWKS project plans to roll out the new outstations on fibre-optics only. We also expect that the impact of hard and software failures will change with the new outstation hardware. It is, at the moment of writing, not known if the reliability of the iWKS outstations is better or worse than the older design and we will only be able to know this after a more meaningful sample size of iWKS outstation data is collected.

On the other hand, some causes will not be resolved with the installation of iWKS outstations. For example, power failures and road works will still occur even with new outstation hardware. Furthermore, a large number of failures is still unidentified and their cause might still play a role despite the update brought by the iWKS project. Therefore, a design that can handle outstation communication failures is still needed for.

Chapter 4

Related work

This chapter presents work related to traffic management MAS and fault-tolerance in MAS. Starting with decentralised motorway traffic management in section 4.1, we zoom out to decentralised traffic management in section 4.2 and to fault-tolerance in general MAS solutions in section 4.3. We discuss the reasons why available solutions are not suitable for our problem and try to find elements of existing solutions that might prove useful in our design.

4.1 Decentralized motorway traffic management

Multi-agent systems are not new to motorway traffic management [3, 1, 11], but there are few approaches that resemble the highway control infrastructure in place in the Netherlands.

In [14], Hernández et al. propose a system called TRYSA_2 that divides the traffic management of the motorways around Barcelona into a number of sections that autonomously communicate with each other in order to obtain a consensus on how to mitigate congestion.

In [25], the solution proposed to reduce congestion involves the communication and coordination between regulatory systems alongside the road. The difference between [14] and [25] is that TRYSA_2 involves zones containing multiple regulatory systems whereas the system of Van Katwijk et al. involves the regulatory systems themselves.

Both MAS implementations only consider ramp metering (a technique to limit the amount of vehicles allowed to enter the motorway) and variable message signs which are used only for traffic routing on a network-wide scale. The work done by outstations, on the other hand, involves more localised and critical processes. This could be the reason why fault-tolerance was not discussed in both articles.

4.1.1 Decentralised motorway traffic control

Although the MTM system in use at Rijkswaterstaat contains only a set of static rules and no control algorithms, both traffic management and control are similar in terms of design. In [22], Popov et al. propose a system that uses *variable speed limits* (VSLs) to counteract an ongoing traffic shock wave. These VSLs are very

similar to the MSIs used by Rijkswaterstaat and described in chapter 2. The way they achieve this shock wave reduction is by using an evolutionary algorithm at every VSL to deal with the non linearity of the problem. VSLs can take input not only from speed detectors at its location but also from up to 6 neighbouring sensors. This way, the system has a form of "look ahead" functionality to take incoming shock waves into account.

In [12], Ghods et al. use game theory and model predictive control to decentralise the control of traffic. In simple terms, game theory works on the principle that a system is composed of players all playing the same game and thus working towards a similar goal. In this case the players would be ramp metering devices and VSLs. Each player processes the following algorithm:

Initialisation In this step the system initialises with a random control value and stores it in a history.

Sampling A control value is drawn from the history of each controller arbitrarily.

Optimisation The model predictive controller now runs its optimization function that minimizes a cost function, in this case travel time, assuming the other controllers used the same control value drawn in step 2.

Store The control value calculated in step 3 is stored in the history.

Stop-condition Check whether the convergence of the cost function for each controller has occurred then stop and repeat this algorithm for the next iteration of the model predictive controller, else go to step 2.

With this algorithm players will know each other's strategies in an iterative fashion, this is known as the sample fictitious play approach.

Both systems presented in this section are control solutions, as opposed to management solutions. This means that concepts like game theory which are very useful in control algorithms are not needed in the case of this thesis. Also, no form of fault-tolerance is present in the solutions of Popov et al. and Ghods et al.

4.2 Multi-agent traffic management in other domains

Besides motorway traffic, MASs have been used in other types of traffic management. MAS implementations in air traffic management have been well researched over the past decades[24, 19, 15].

Bicchi et al. [5] have researched an MAS implementation that aims to replace the current centralized air traffic control operation by a decentralized system. Their approach involves associating an agent with every airborne plane. Each agent/plane has an (identical) alert-radius that defines the limit of safe flight. If two planes enter each other's alert-radii, the agents associated with each plane calculate a new trajectory for both aircraft. Since the calculation is identical for both agents, the consensus reached is the same for both planes.

Fault-tolerance was assessed in this article by simulating a 'crisis' where the agent of a plane would communicate false readings. In the simulations, depending on the alert-radius, the number of failures (occasions where a crisis results in a critical situation for flight safety) could be halved compared to a centralised solution.

Another approach to decentralized air traffic management was proposed by Breil et al. [6]. Their approach is similar to the previous one but instead of changing the heading of the aircraft, the agents involved change the speed of the plane in order to avoid a conflict. Also, the way the authors test fault-tolerance differs between the two systems. In this article, instead of generating a crisis on every aircraft in the simulation, 10% of the planes have their agent disabled meaning that other agents now have to resolve a conflict without a predictable counterpart. The solution proposed is designed in such a way that this only results in an approximately 5% increase in failures compared to a simulation where all agents are enabled.

In these last two articles, the fault-tolerance aspect of the decentralized system resides in the fact that the traffic is managed by the entities (in this case, air planes) involved in the traffic. This way, when one of two (or more) agents involved in a conflict is down, the other (or others) can still get out of the conflict since their agents are operational and can correct by recalculating their trajectory or speed accordingly. An issue with this method is the one-sidedness of the computation when an agent is down, which results in a sub optimal decision by this agent.

Roussos et al. [23] propose a solution for this problem by introducing a priority system. The idea is to assign agents to mobile objects (like aircraft) and also assign them a priority, with 1 being the highest. Immovable objects or faulty agents get priority 0, which means that they will not cooperate in the calculation for a new trajectory. This way an agent in conflict with another agent with higher priority will handle the new trajectory on its own. At equal priority (except for priority 0), two agents both participate in the calculation for a new trajectory.

The main difference between the solutions presented in this section and MASs for motorway traffic management is the location of control with respect to the traffic to be controlled. All MAS implementations in this section place the agents in the vehicles taking part in the traffic. This allows for a very granular way of influencing the traffic, however this requires each vehicle to have the capabilities of running an agent implementation (e.g. communication, computation). However, since we are not interested in associating an agent of our MAS with every individual element of the flow we want to control, this solution does not suit our needs.

An alternative would be to implement a platform of agents on Rijkwaterstaat's WKSs that would perform the calculations for individual vehicles. This would require each WKS to have the ability to track vehicles in the motorway network and would require the vehicles to have the ability to communicate with the WKS and follow instructions from the latter.

Since Rijkswaterstaat cannot influence what vehicles take part in traffic (force the implementation of vehicle-to-infrastructure communication) they are limited to communicating with the driver through devices external to the individual vehicles. This method does not allow individual vehicle management but only a general flow management.

4.3 Fault-tolerance in general MAS

As described by Guessoum et al. [13] fault-tolerance in general MAS can be classified into two categories: corrective en preventive. In the corrective approach, the process consists in making an accurate diagnosis of the fault and repairing the damage. In the preventive approach, the idea is to continue to deliver the desired services when one or more faults occur. This last category is the most interesting for this thesis. This category can be further divided in two main branches: the concepts based on replication/redundancy and the ones based on reorganisation.

Replication-based concepts assure that an agent has a backup available in case a failure occurs. This can be done for every agent which results in a very resource demanding system or it can be done by selectively assigning replication to important agents in the system [13, 2, 18]. This isn't a good solution for our design, since replication means that each outstation would need a backup outstation on location since that is where the faults we want to prevent occur. Such a solution will need lots of resources to the point of being unfeasible. Reorganisation for fault-tolerance takes advantage of the distributed nature of MAS and looks for a service on a different agent when the original one fails. This solution is closer to the approach sought for in this thesis.

Horling et al. [16] propose a reorganisation concept for MAS based on self-diagnosis using a modelling language called TÆMS to model the tasks and goals of an agent. TÆMS uses two views of the agent, one is the subjective view which lists all the possible roles and responsibilities (similar to interrelations with other agents) and the second is the conditioned view which represents the set of assigned roles and responsibilities. The goal of the whole system is thus to find a mapping between the subjective and conditioned view that is the most optimal for the whole system. In order to achieve this, the agents are equipped with the ability to self-diagnose. Based on the environment and models of the environment, the agents can detect certain symptoms which are combined into a diagnosis. This diagnosis leads to a reaction (an action on the environment) that mitigates the symptoms.

In [17], Kota et al. show a different way to reorganise agents in a MAS. Their system works under the assumption that a MAS performs certain services in a certain order. Each service costs processing power per unit of time, which is a limited resource for agents (also called a budget, defined over a time slot). Services are (randomly) distributed among agents which need to communicate with each other to perform the sequence of services since one agent probably cannot execute all services. The communication happens on three levels: acquaintance (every agent knows of the other agents, this is the default level), peer (infrequent communication) and subordinate (frequent communication). Each level is also associated with a cost. When a root task is called on an agent, it tries to execute it and tries to execute the subtasks. If the task is in the set of that agent but exceeds the computation budget of that agent, it can be pushed to the next time slot. If the task is not in the set of that agent, it will ask its subordinates first and then its peers if they have the task and the budget to do it. The system can adapt itself at any time if no subordinate or peer has the task needed or if the queue starts piling up. By calculating the cost of changing its relationships and estimating the resulting general cost of executing the root service, the system can calculate an overall cost. If the reward (faster execution)

outweighs the cost of reorganisation, the agents change their relationships.

The fault-tolerance aspect of the concepts of the previous two papers might not be immediately apparent. In [16], the self-diagnosis aspect can be used to detect failed agents since it bases its diagnosis on the environment of which other agents are a part of. This detection would result in a reorganisation since the service at the faulty agent would not be available any more. In [17], a failed agent would result in an unavailable subordinate or peer which means that the load on the other agents will increase and thus another organisation may emerge that involves other agents with the unavailable service.

However, both concepts pose a few problems given their very general nature. First of all, although the reorganisation is local, the agents involved require a global view of the MAS. Second, certain steps in the process are application specific and thus not elaborated. For example in [16], the reactions associated with a certain diagnosis are not specified. Also, in [17], the selection of an agent to execute a certain service is random while in the case of traffic management, the physical location of a certain task is essential and thus requires a different computation. The result is that a lot of the decision making of an agent still needs to be designed and implemented.

4.3.1 Adaptive Multi-Agent Systems

One branch of multi-agent systems, called *adaptive multi-agent systems* (AMASs), focusses on a specific type of MAS. An AMAS [8] is a multi-agent system designed without a general goal. Although the general goal of the whole system does exist (otherwise the desire for developing such a system in the first place would not exist), it is not retrievable from the final implementation. Instead, the interactions between the individual agents define the goal(s) and functionalities of the whole system. As described by Capera et al. [8], each agent computes a partial function fp_i, and the combination of all the partial functions produces the global emergent function fs. The advantage of this approach is that, for complex systems, the designer only has to focus on small individual elements of a bigger systems instead of the whole system itself. The authors do not provide a way of implementing this solution and instead leave it up to the designer of the system.

As described by Picard et al. [21], AMASs can be used for fault-tolerance due to their capability of reorganizing themselves. Agents in an AMAS have the ability to detect conflicts, where a conflict can, for example, be an unresponsive agent or a conflict of interests between agents. These conflicts lead to non-cooperative states, which is a state of the MAS where the goal cannot be attained. The deployment of the MAS specifies how to return to a cooperative state. Multiple methodologies exist to help do this [4, 10] in a structured manner. This solution generates a MAS that is capable of handling foreseeable conflicts and failures in a fully distributed manner.

To the best of our knowledge, no traffic management solution is based on this AMAS theory which can form a solid base for a fault-tolerant solution.

Chapter 5

System Design

Having established what might cause communication failures between outstation and central system, and why currently available solutions are not suitable for our problem, we design our own solution. After discussing the requirements the design has to fulfil, we break the design of this system up in 2 parts: first, we discuss the design of the outstation and the central system, and second, we discuss the communication protocols that make this design fault-tolerant.

5.1 Requirements

We start this design by setting the requirements the new systems needs to fulfil.

First, we want the new system to work on any communication network topology as long as the outstations can communicate with each other. With this requirement we allow our design to be applicable for a larger amount of systems.

Second, since Rijkswaterstaat plans on deploying different types of sensors in the near future, we want the new system to be unrestricted in the type and amount of sensors connected to the outstations.

In terms of fault-tolerance, we want the outstation network to recover from a communication failure in less than 60 seconds. We chose this value because, in the current system, the fastest changing input, which can generate a change in actuation, is the traffic sensor input. This input generate a new data point every 60 seconds and we want to miss, at most, 1 data point. After a failure has occurred, the new system should recover in such a way that the sequence of symbols shown to the drivers is, at least, as restrictive as it was before the failure.

Lastly, we want the system to be able to recover from any number of outstation failures, be it consecutive (single outstations failing one after the other) or simultaneous failures (multiple outstations failing at the same time).

5.2 Design

Our goal is to create a multi-agent motorway traffic management system that is fault-tolerant to outstation communication failures. We want a design where the outstations detect the failure of another outstation, after which, they change their relation with the outstations around them, in order to maintain a sequence

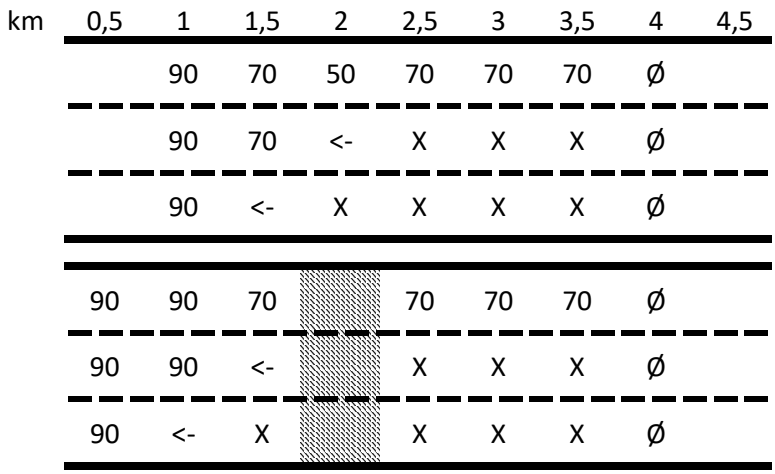


Figure 5.1: **Two identical symbols sequences under the new design. The upper sequence has all outstations operational. In the lower sequence, the outstation at kilometre 2,5 is faulty. The lower sequence shifts one outstation upstream from the failed one compared to the upper sequence.**

of symbols that is safe for the drivers, as shown in figure 5.1. Therefore, we associate an agent with every outstation in the network. We also give every outstation the capability of locating itself with regards to other outstations within the highway network. This means that the resulting system works for any type of communication network topology linking the outstations, since their relations are based on their location in the highway network.

We based our methodology for designing a single agent/outstation, on the methodology used for TRYSA$_2$ [20], a MAS traffic control system developed for the Barcelonian highway network. This methodology allows for the design of modular and versatile agents, as will be explained in the next section.

First, we change the way triggers are handled under normal operation. Since, in our MAS, each outstation can 'think' on its own, the congestion trigger is handled at the outstation instead of at the central system.

Second, we move the calculation of the new symbols to the outstations. The rules are implemented locally instead of at the central system and the outstations set the new symbols on their associated MSIs. The central system remains the receiver of triggers from the traffic manager. However, since the central system does not process these triggers any more, it forwards them to the outstations.

Third, when an outstation fails, a protocol kicks in that reconfigures the relations between outstations so the sequence of symbols stays in accordance with the rules set by Rijkswaterstaat.

5.2.1 Single outstation

To develop our outstation agent, we used the methodology of the developers of TRYSA$_2$. It is based on *knowledge units* (KUs) acting on different layers of the decision making process. A knowledge unit is a concept proposed by Cuena

Layer	KU Name	Function
Normative layer	Central KU	Stores central triggers
Social layer	Agent dependence KU	Stores information about the outstations the agent is currently communicating with
	Action interrelation KU	Stores information about the actions or measures of the outstations whose information is stored in the Agent dependence KU
Local layer	Configuration KU	Stores information about this agent's configuration in the road network
	Congestion KU	Stores information about the local traffic state
	Rules KU	Contains the local rules that determine how to update the MSIs

Table 5.1: **Knowledge units (KUs) in an outstation and their function.**

and Molina [9] to describe a subsystem of an intelligent agent responsible for a small part of the information processed by that agent. As the authors state: "The knowledge unit is a building block which models a knowledge area. The knowledge unit does not follow the classical computational division of process and data. On the contrary, it models a knowledge body to be identified in the modelling process of the intelligence of an agent." Internally, a KU is defined by what it knows (the data) and what to do with it (the data processing) for a specific type of information.

The methodology used in TRYSA_2 separates the KUs in 3 layers and in such a way that each layer can be implemented and tested individually. These layers are: the local layer, responsible for the knowledge and actions taken locally, the social layer, responsible for the knowledge and actions taken based on other outstations and the normative layer responsible for the knowledge and actions taken based on a network-wide action. The interactions between these layers form a complete and intelligent agent.

At the core, the new outstations also use these knowledge units and KU layers to decide what to display on the MSIs. All KUs are summarised in table 5.1.

At the local layer, an outstation has 3 KUs: the configuration KU stores information about the agent's location in the road network. The congestion KU tracks the local traffic state (congested/not congested). The last local KU contains the rules used to determine what symbols to display.

At the social layer, an outstation has an action interrelation KU and agent dependence KU. The agent dependence KU stores information about the other outstations it is currently communicating with. The action interrelation KU stores information from the outstations in the agent dependence KU that can affect local MSIs.

In the normative layer, an outstation has a single KU, the central KU, that stores information from the triggers from the central system.

What makes an agent intelligent are the relations between its knowledge units. These interactions are shown in figure 5.2 in the form of a sequence diagram.

When an agent at an outstation starts up, a configuration file is provided. This file contains information about the outstation's local configuration (e.g. number of lanes, number of MSIs, lanes relations) and its location in the road network (e.g. road number, road side, hectometre position). This information is stored in the configuration KU.

The sensor(s) connected to the outstation update(s) the congestion KU. This is a periodic process with a frequency dictated by the sensor in use, in our case, the period is 60 seconds. When the congestion KU changes from congested to not congested, or vice versa, this is a local trigger, and the outstation calls upon the rules KU to check if the MSIs need an update. If so, the new symbols are calculated and the MSIs updated.

When a traffic manager sets a new measure, this is sent to the outstations where the central KU is updated. This also leads to a trigger (a central trigger) which calls upon the rules KU to update the MSIs. With these relations, an outstation is capable of displaying symbols based on local triggers and a central system.

As outstations now calculate their MSI updates locally and with the KU relations described so far, a full sequence of symbols cannot be displayed. Indeed, outstations do not know what to display in relation to each other. In the current system, this is implemented in the central system, but, since we have intelligent agents, this functionality is transferred to the outstations.

The agent dependence and action interrelation KUs are responsible for this task. These KUs are closely linked to the communication between outstations. We will have a closer look at the communication protocols later in this chapter. The action interrelation KU is updated when a message from another outstation, whose information is stored in the agent dependence KU, is received. The information in this message and, thus, in the action interrelation KU, consists of the measures and/or actions of the other outstations. With this information, the outstation might need to update its MSIs and therefore calls upon the rules KU to do so.

Depending on the implementation, the number of outstations, of which information is stored in the agent dependence KU, can vary. When the contents of this KU changes, it means that the outstation's relations with other outstations has changed. As this often goes in pair with a change in the action interrelation KU, a change in the agent dependence KU often updates the MSIs as well.

Lastly, the outstation sends a message to all outstations, for which information is contained in the agent dependence KU, containing the outstation's measures and/or actions. And it regularly sends updates of its MSIs to the central system so the traffic manager can see what is happening.

5.2.2 Central system

An outstation, with an agent associated to it, can operate on its own in a network with other outstations. However, in some cases, like the one for Rijkswaterstaat, it is still important that a central authority (a traffic manager) can interact with the system. We therefore append this design with a central system.

The central system is also considered an agent in the whole system but it is very different from the outstations. In the new design, the central system has two main tasks: transmitting the triggers from the traffic manager to the

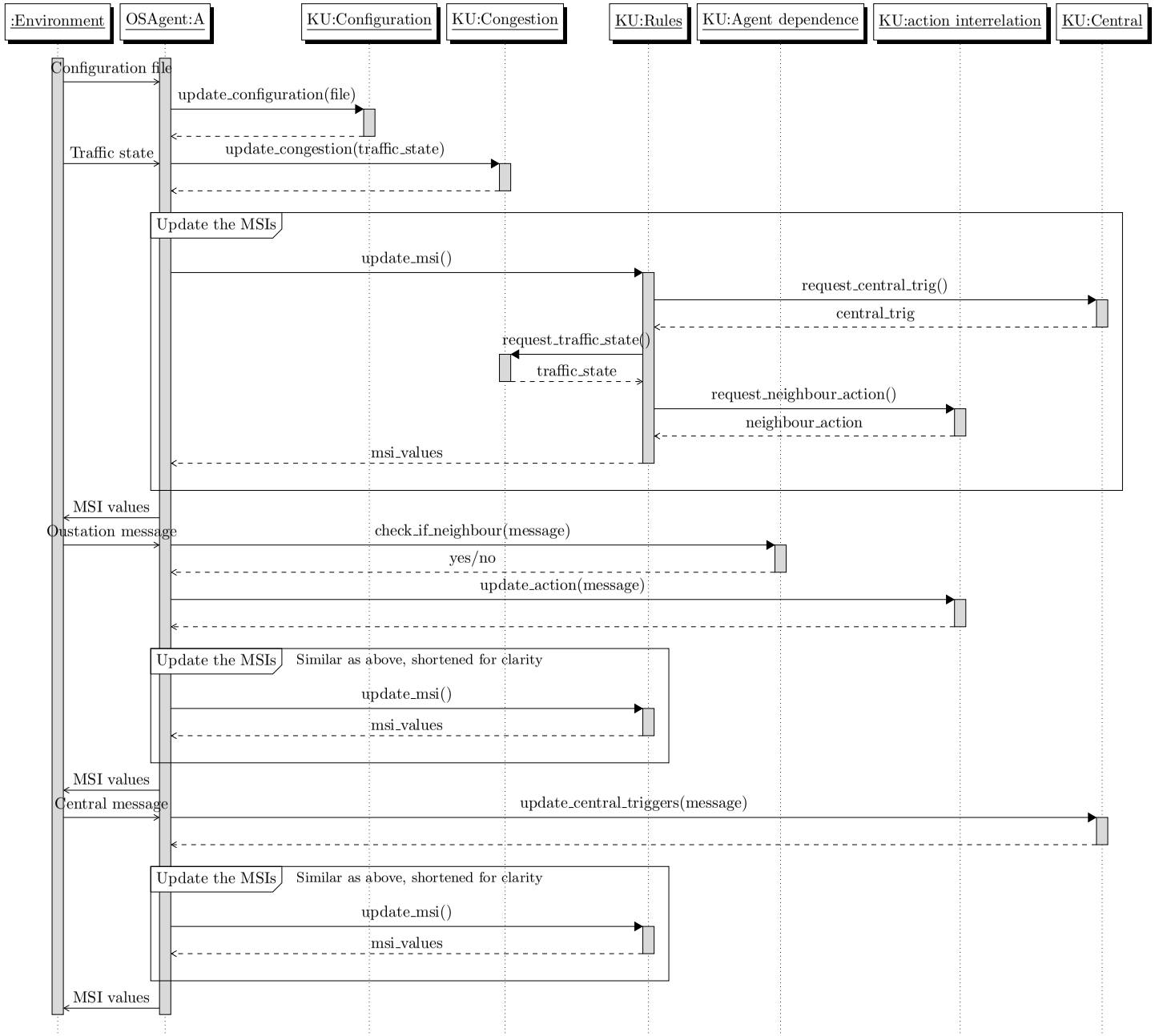


Figure 5.2: **Sequence diagram of the interactions between KUs at on outstation. All blocks named "Update the MSIs" are similar; the last two are shortened for the clarity of the diagram.**

outstations and showing the symbols currently displayed by all the outstations to the traffic manager.

When a traffic manager wants to set a (set of) symbol(s), he communicates with the central system and defines a basic measure, for example, a cross on the first lane of a road between two outstations. This basic measure is then stored by the central system together with an ID and sent to the outstations on the road(s) of the measure. When a measure needs removing, the traffic manager communicates with the central system and deletes the appropriate basic measure. This removes the measure from the storage and a message to remove the basic measure is sent to the outstations.

5.3 Communication

In order to make the outstations and the central system communicate with each other, we design a new set of protocols. These protocols are key to the fault-tolerance of the system since the only way to know if an outstation is not working, is by checking if it responds to messages. The goal of the new MAS design is to decentralise big parts of the functionality, including fault detection. Outstations will now have to check for faults amongst each other. The following sections will focus on the different interactions between sub-systems.

5.3.1 Central to outstation

Central to outstation communication is only relevant for central measures. When the traffic manager sets a basic measure, the central system sends a message with the basic measure to the outstations. This message has the following content: a symbol set, a road name/number, a start hectometre, an end hectometre and ID number.

The symbols set defines what is to be displayed as the basic measure. The road name/number defines the road where the basic measure applies. The start and end hectometre are used by the outstations to determine if they are in range of the basic measure and thus if it applies to them. The ID number is used to identify each basic measure since multiple measures can apply at the same location with the same symbols. If a measure needs removing the central system sends a message with only the ID of the measure that needs removing.

The central system sends these measures to all outstation in the network. The reason why we chose to do so is explained in section 5.3.4.

5.3.2 Outstation to central

The only message sent from outstation to central is the feedback from the action of the outstation. Its content is limited to the symbol set of the MSIs of the outstation. This is the only feedback needed for the traffic manager.

5.3.3 Outstation to outstation

The most complex protocol is the one between outstations. It is responsible for sharing MSI information and detecting faulty outstations. Under normal operation, when all outstations function properly, the protocol shares information with the outstations whose information is stored in the agent dependence

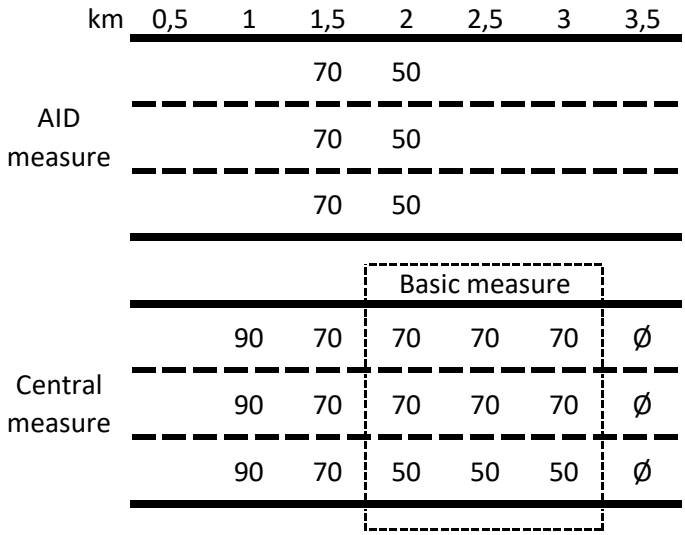


Figure 5.3: **An example of two measures, which, when combined, lead to incomplete information at kilometre** 1**.**

KU. Multiple options are available to do this, we explore three of them that differ in information content and recipients: action-share, measure-share and type-multicast.

Action-share This protocol variation periodically shares the outstation's action (see section 2.3) with its direct neighbours. Sharing only the action yields a small packet of information which keeps the bandwidth low. By sharing information only with direct neighbours (i.e. the first outstations up and downstream), we further reduce the bandwidth.

However, this method of communicating has a downside: in some cases it does not respect the rules set for the whole sequence of symbols, even if it respects the rules in the rules KU. For different types of measures (AID and central), the continuation up and downstream is different. For example, in figure 5.3, two measures are shown that both have effect on the outstations at kilometre 1,5 and 2. The MSIs at kilometre 1,5 show a row of 70 km/h restriction across the road in both cases. For the AID measure, the sequence does not travel further upstream, for the central measure, the sequence is extended with a 90 km/h restriction. The outstation located at kilometre 1 is only aware of the 70 km/h restriction, not the type of measure, and thus, does not know if it should display a 90 km/h restriction or not. For safety reasons, it will display the 90 km/h restriction but, in some cases, this will result in an over-restricted traffic flow.

Measure-share A solution to the over-restriction problem is present in the measure-share protocol variation. This variation shares measures instead of actions with its direct neighbours. By sharing the list of measures (this can be seen as the input of the rules KU before performing the calculation), neighbours are aware of the type of measure associated with a symbol and can therefore make

the right choice in terms of up and downstream continuation. The downside is that the amount of data to transmit increases, as, at one outstation, multiple measures, for which the symbol set and type are transmitted, can be active.

Type-multicast The last variation is a compromise between the two previous protocol variations. It periodically shares the type of basic measure with a set of neighbours up and downstream. This protocol is based on the principle that a sequence of symbols along the road always has a finite length. This length is directly related to the number of lanes controlled by the outstation.

If we take the central measure depicted in figure 5.3 as an example, the basic measure is active at kilometre 2 and 3, the rest is a reaction to that basic measure. In this case the reaction is limited to 2 outstations upstream of the basic measure and 1 downstream. This depends on the type of measure.

For a three lane highway, the maximum length of an upstream sequence is 5 and downstream is 1. Based on the number of lanes associated with the particular outstation, which is given in it's configuration, we can set the size of the set of neighbours to communicate with.

During normal operation, only the outstation where the basic measure or congestion trigger applies shares the symbols of the measure with the outstations around it. Using the rules stored in the rules KU, the outstations around it will know how do react.

This is different from the measure-share protocol in that, in the measure-share protocol, all outstations share their measures, regardless if these measures are a basic measure or a reactions to a basic measure.

In terms of bandwidth, the number of recipients has grown with regards to the other two protocols, but the number of senders is smaller since reactions are not shared. Also the size of the message is reduced compared to the measure share protocol since less information needs to be sent.

5.3.4 Fault-tolerance

To assure a design that is tolerant to communication failures with the outstations, all protocols send their messages periodically instead of, for example, on change. Taking the action-share protocol as an example, under normal operation, an outstation sends a message to their upstream neighbour. This neighbour then replies, as shown in the upper box of figure 5.4. This happens periodically with a frequency defined by the implementation. To detect faults, both outstations in the conversation keep a timer for each other's messages. When the timer reaches a timeout, the outstation is considered failed and the recovery protocol kicks in.

Instead of sending a message with relevant content each period, it would be possible to simply send a keep-alive packet. We chose not to do so for two reasons: first, when sending content periodically, the receiver of the data knows when its information is out of date and, if so, how old it is. Second, the content of the message can be kept really low, if implemented efficiently, which reduces the bandwidth advantage of keep-alive messages.

When a communication failure occurs at an outstation and the timeout has been reached, the outstations that remain operational choose a new outstation to communicate with. In our design, outstations find a new neighbour by broadcasting that they are looking for a new neighbour. Since, when one outstation

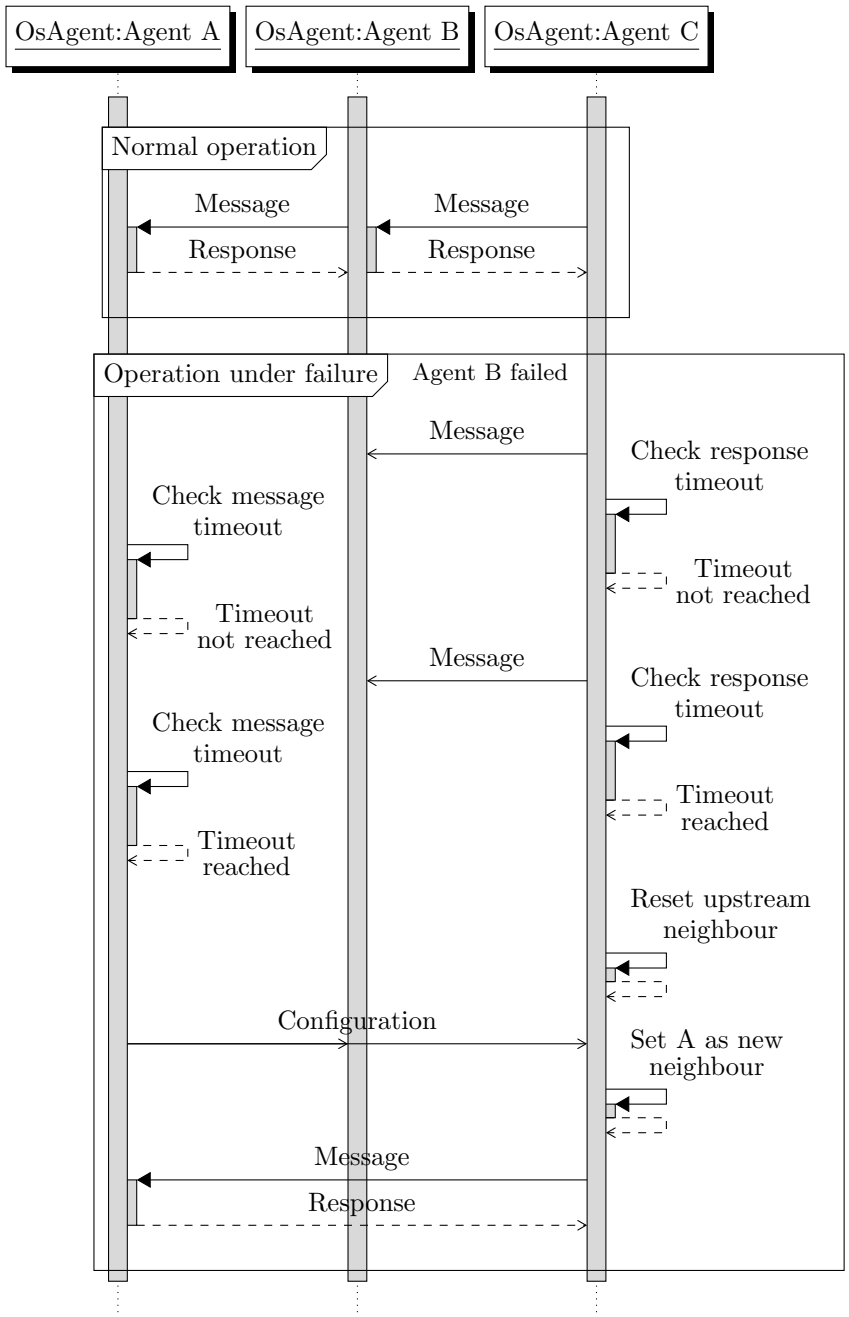


Figure 5.4: **Sequence of messages and actions under normal operation and when a failure occurs at agent B.**

fails, 2 others loose a neighbour, this would result in 2 outstations broadcasting a message when 1 outstation fails. We can reduce the amount of broadcasts sent by letting only the outstation downstream of the failed outstation broadcast and by letting the outstation upstream of the failed outstation listen to new broadcasts. When the listening outstation has chosen the broadcaster as its new neighbour, it sends a message back to the broadcaster. Upon reception of the message, the broadcaster recognises the listener as its new neighbour.

In order for the listening outstation to know if the broadcasting outstation is a good neighbour candidate, the broadcast needs to contain information about the outstation's position in the road network. With that information, the listener can deduce if the broadcaster is close enough to be its new neighbour. We encode this information in the outstation's configuration, which is stored in the configuration KU, as mentioned in section 5.2.1. The configuration is sent as content of the broadcast.

Figure 5.4 shows the recovery process, when 1 outstation fails. We consider 3 outstations, A, B and C ordered from upstream to downstream on the same stretch of road.

In the lower box of the figure, agent B failed and is thus unresponsive. Agent C sends a message to agent B, as would happen under normal operation. No response is received by agent C, so eventually the timeout for its upstream neighbour is reached. Agent C then resets the information stored in the agent dependence KU for its upstream neighbour. Since agent B is not sending any messages anymore, agent A will eventually reach the timeout for its downstream neighbour. Agent A will then broadcast its configuration (i.e. the information stored in the configuration KU). Afterwards, it resets the timeout to prevent itself from continuously broadcasting its configuration. Agent A's configuration is picked up by agent C and used to update its upstream neighbour information, stored in the agent dependence KU. Next, agent C sends a message to agent A which lets agent A know that agent C is now its downstream neighbour. The system can now continue functioning as normal while skipping the failed outstation in the symbol sequence.

Now, we consider the sequence of events when an outstation (re)joins the network. Using the same outstations as mentioned in the previous example, we consider the return of agent B to the network.

When the failed outstation, agent B, is restored and rejoins the network, it will have no upstream or downstream neighbour. For that reason, it will broadcast its configuration. Agent C and agent A are listening to any broadcasted configuration and compare agent B's configuration with the one stored in their agent dependence KU. If the newly received configuration is closer than the upstream neighbour currently stored in the agent dependence KU, the outstation switches neighbours. This is the case for agent C, who will therefore send a message to agent B, letting agent B know that agent C is now his new downstream neighbour.

This action leaves agent A without a downstream neighbour. So that agent A does not have to wait for a timeout, we let agent C send a "dump" message to agent A. This notifies agent A that it will need to look for a new downstream neighbour. Agent A therefore starts broadcasting it's configuration. This is picked up by agent B who sends a message to agent A letting it know that agent B is now his downstream neighbour. All three outstations are now able to resume normal operation as shown in figure 5.5.

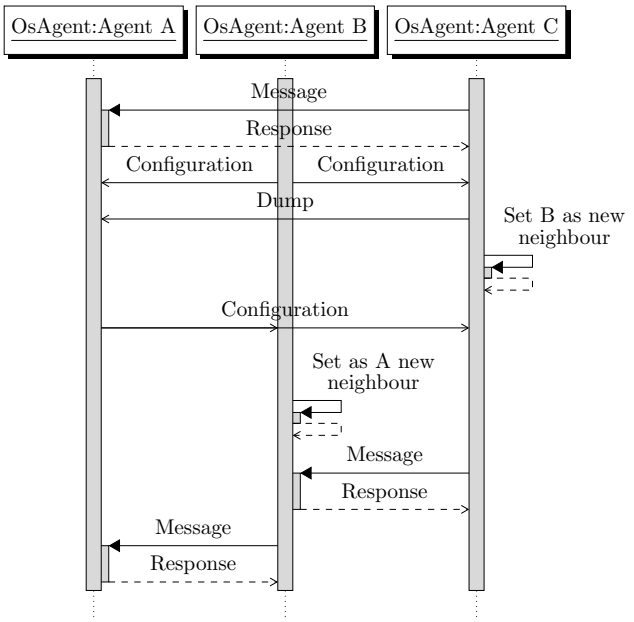


Figure 5.5: **Sequence of messages and actions when agent B joins an existing network.**

An alternative for the protocol described above, is to use the configurations to build a backup list of neighbours at each outstation. Upon failure, an outstation missing a neighbour can then use this list to find the next best communication partner. We did not choose for this method of reconfiguration for a two reasons.

First, after the list has been created at each outstation, by an operator or the outstation itself, it needs to be updated regularly. When structural changes are made to the highway network, like the addition of an outstation, the MTM operator or the other outstations need to engage in a backup list update process. Depending on the size of the list maintained by each outstation, a structural change of only one outstation, impacts multiple other outstations. Even if this process is automated, it will take multiple sequential message exchanges before the structural change is registered by all concerned outstations. The operator making a structural change will need to include, at least, the closest neighbours to the change before the automated update process can start. Therefore, the operator should have knowledge of the complete network structure. Our design allows for a quick and easy way of updating the relations between outstations. When, for example, a new outstation is added, the operator only needs to give the new outstation its configuration, after which, a very localised number of outstation engages in an automatic reconfiguration.

Second, using a backup list, when multiple adjacent failures occur at the same time, the outstations will need to search their list of backup neighbours (iteratively) which takes time. This time increases with the number of outstations that fail simultaneously. In our design, an outstation starts searching for a new neighbour at the moment of failure. The time needed to do so is constant, regardless of the number of adjacent failed outstations. However, when two non-adjacent outstations fail simultaneously, our design might take more

time than the backup list solution. These failures are rare, since simultaneous failures usually involve a connection failure and, given that adjacent outstations use the same connection, these failures usually involve adjacent outstations. Nonetheless, in these rare cases, our design might first choose a sub-optimal configuration as a neighbour before switching to the best new neighbour. This takes multiple rounds of broadcasts and thus more time.

When an outstation (re)joins, we assume that it has lost all stored data, including applicable central measures. Therefore, upon reconnection with a new neighbour, the downstream outstation sends a copy of the currently applicable central measures to the reconnected outstation. Central measures are then checked against any already applicable central measures in order to avoid double measures. The ones that are not already applied are then added to the list of central measures.

This, however, poses a problem regarding the requirements set earlier. Indeed, if the central measures are shared from outstation to outstation on reconnection, we cannot have all outstations disconnect since the central measures would no longer be stored anywhere. However, for Rijkswaterstaat, it is highly unlikely that all outstations disconnect.

Another option for restoring central measures on (re)joining outstations is too have the central system send them to these outstations. The downside of this solution is that it creates a single point of failure, which is what we want to avoid. Data storage virtualisation, like RAID, can also be used to restore central measures. In that case, all outstations or groups of outstation could be combined into a single distributed storage space. When an outstation fails, the storage can recreate the missing data from the remaining outstations and central measure can stay in place. When an outstation (re)joins the network, it can be integrated in the virtualised storage. The problem with this solution is that this virtualisation usually places a limit on the number of physical storage units that can fail before the data is unrecoverable. This clashes with our requirement to allow any number of outstations to fail. Furthermore, for Rijkswaterstaat the amount of central measures active at any point in time is vastly smaller than the storage available to a single outstation. In 2017, the maximum amount of central measures active at any point in time was 413. At a few tens of bytes per central measure stored, the total size of these measures is in the order of tens of kilobytes. We therefore choose to store all the central measures on all the outstations. This permits all outstations, except one, to fail before any central measure is lost.

5.3.5 Timing

Timing of the protocol is important, since the requirements set a bound on the recovery time.

The time needed to reconfigure the system after a failure is dependant on the timeout value, the travel time of the message and the different compute times needed at the agents to perform all actions. Assuming both the timeouts for the receiver and the sender of a request have the same value, the worst case occurs when a configuration is broadcasted but the expected receiver for the configuration has not yet reset its upstream configuration. This means that the configuration will need to be broadcasted again in order to recover from the failure as is the case in figure 5.6.

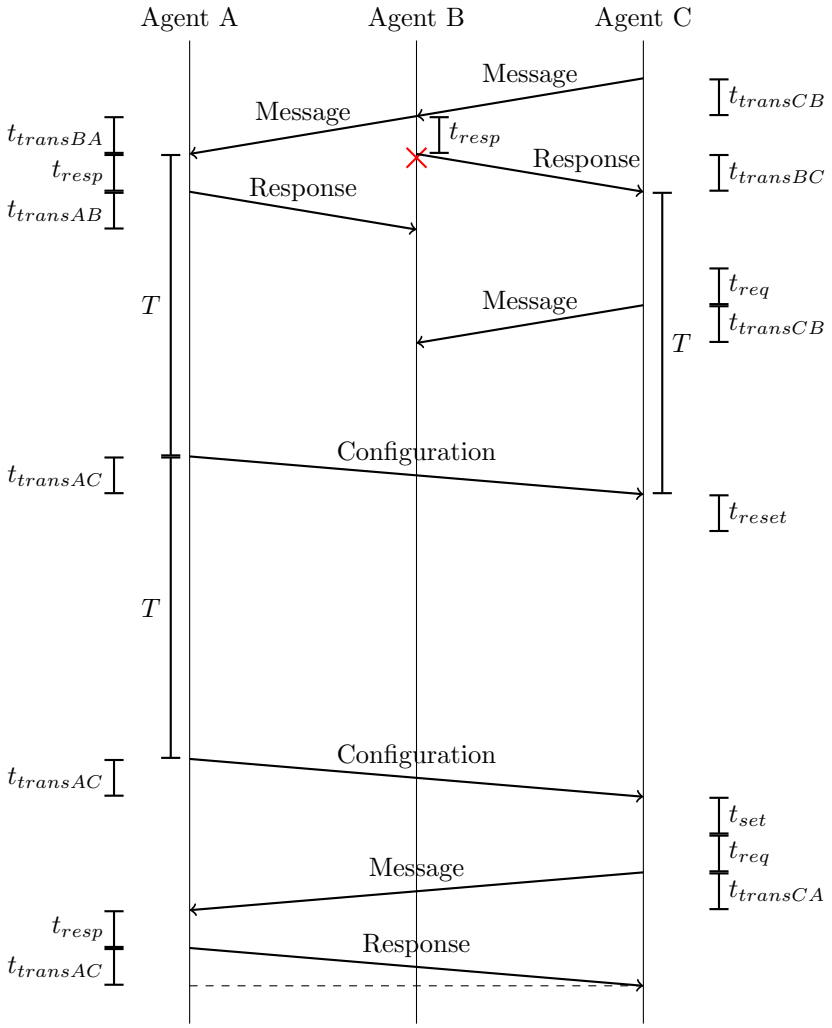


Figure 5.6: **Sequence of messages between 3 agents during failure and recovery including timing in a worst case scenario. The red cross indicates the moment agent B fails. The dashed line indicates the moment the system has recovered. T represents the timeout. $t_{transXY}$ is the time taken by the message to travel from X to Y. t_{resp} is the time needed to calculate a response to a message (this is considered to be constant regardless of the message content). t_{reset} is the time needed for the agent to reset the information about a neighbour. t_{set} is the time needed to assign a new configuration to a neighbour. t_{req} is the time needed to create a request before sending it.**

In order to find the theoretical worst case time needed to recover from a failure, the following assumptions are made: the agents are assumed to be single threaded and we assume that the time needed to compute a response message is constant regardless of the content of the message. Lastly, as mentioned earlier, we assume that the timeouts for the sender and receiver of a message are of the same length.

In figure 5.6, for the first configuration broadcast from agent A to arrive before agent C has reset its upstream neighbour, we need the following condition.

$$\text{Time to receive configuration} < \text{Time to reset neighbour}$$
$$-t_{resp} + t_{transBA} + T + t_{transAC} < t_{transBC} + T + t_{reset} \tag{5.1}$$

Where T represents the timeout, $t_{transXY}$ is the time taken by the message to travel from X to Y, t_{resp} is the time needed to calculate a response to a message (this is considered to be constant regardless of the message content) and t_{reset} is the time needed for the agent to reset the information about a neighbour. If this condition is true, agent A would need to resend the configuration in which case the total time needed to recover from the failure is equal to the following sum.

$$\begin{aligned} t_{recover} &= -t_{resp} + t_{transBA} + 2T + t_{transAC} + t_{set} + t_{req} + t_{transCA} + t_{resp} + t_{transAC} \\ &= t_{transBA} + 2T + 2t_{transAC} + t_{set} + t_{req} + t_{transCA} \end{aligned} \tag{5.2}$$

Where t_{set} is the time needed to assign a new configuration to a neighbour and t_{req} is the time needed to create a request before sending it.

This equation represents the worst-case time when the last action of the failing agent is responding to a downstream agent. It is also possible that the agent's last action is sending a request upstream. In this case, the condition for sending a configuration twice changes.

$$t_{transBA} + T + t_{transAC} < -t_{req} + t_{transBC} + T + t_{reset} \tag{5.3}$$

In this case the total time before reaching a recovered state is equal to the following sum.

$$\begin{aligned} t_{recover} &= t_{transBA} + 2T + t_{transAC} + t_{set} + t_{req} + t_{transCA} + t_{resp} + t_{transAC} \\ &= t_{transBA} + 2T + 2t_{transAC} + t_{set} + t_{req} + t_{transCA} + t_{resp} \end{aligned} \tag{5.4}$$

The transmission times and the computation times are in the order of a few tens of milliseconds, since the communication medium is high bandwidth fibreglass and the computers used to run the agents are industrial PCs with a few gigahertz of processing speed. We can therefore assume that $t_{recover} \approx 2T$. Given the requirements set at the beginning of this chapter, we need $t_{recover} < 60$s or $2T < 60$s.

Using this timeout period we can also give a limit to the frequency for sending messages from one outstation to another. We estimate that the chance of losing a packet is low. Even if the connection distances can be long, the amount of hops is small (typically 2) from outstation to outstation with reduces the chances of loosing a packet in transit. Also, the amount of messages send to an outstation is relatively low as $T < 30$s. This means that nor the outstation, nor the network is expected to be overloaded with packets.

We therefore choose to send a packet only once before the timeout is reached. This results in the following condition for the message sending period τ.

$$\tau < T \tag{5.5}$$

We now allow the system to loose a packet, once, without interrupting normal operations.

When an outstation (re)joins the network, the timing is different due to the addition of the "dump" messages. When looking at figure 5.5, no timeouts are visible. The whole process therefore depends only on the processing time needed at the outstation to complete the sequence of actions, and on the communication delays. Both these times depend on the implementation but are assumed to be very small, as argued earlier. This results in an almost immediate recovery.

Lastly, we explore what happens when multiple outstations fail simultaneously. In that scenario multiple configurations would be broadcasted simultaneously and multiple outstations will be able to accept these new configurations. This means that it is possible that the first configuration accepted by an outstation with a missing neighbour is not the optimal one.

As mentioned earlier, agents will always switch to a neighbour that is closer than the current one. This means that if an outstation accepted a sub-optimal configuration as a neighbour, it will switch as soon as a better one comes in. This will lead to a new round of configuration broadcasts since it means that another outstation lost its neighbour. This is however a random process since each outstation schedules its own actions and it is thus very hard to put a number on the time or number of rounds of broadcast that will occur per amount of outstations failing.

The worst case for the amount of configurations broadcasted is given in equation 5.6 and explained in appendix A.

$$B(n) = \frac{n(n+3)}{2} \tag{5.6}$$

Where $B(n)$ is the worst case amount of broadcasts for n number of (re)joining outstations. Although this number will grow quadratically with increasing n, the likelihood of this worst case happening also decreases since the worst case involves selecting a suboptimal neighbour multiple times in a row, the chances of which reduce with every broadcast.

Chapter 6

Simulation and Results

Before drawing any conclusions about our design, we need to know if it meets the requirements set and how well it performs compared to the original MTM system. We start by discussing the way we implemented the design in such a way that it can reliably be tested. We then present the different tests we performed followed by the results of these tests.

6.1 Implementation

A heavily-used and well-documented framework for developing multi-agent systems is the *Java Agent DEvelopment* (JADE) framework. This framework simplifies the development of MAS by providing classes en methods to design new agents and launch them. It also provides tools to debug a running simulation.

In JADE, the programmer defines his agents as an extension of JADE's Agent class. The only mandatory method to program here is the setup method. This function is ran once at the start of the agent and defines all other behaviours of that agent.

When launched, an agent will first run the setup method after which, if any behaviours are defined, a scheduler will schedule the behaviours of that agent. A behaviour is a JADE class that represents actions of an agent. Behaviours can be periodical, execute once or execute on reception of a message.

6.1.1 Outstation

Outstations were modelled by a custom agent class, the osAgent class. During the setup method, the agent will create a configuration object containing its local configuration including location, road, lanes configuration etc. The setup method also initialises a number of behaviours: 3 message reception behaviours and the traffic sensing behaviour. The message reception behaviours are used to handle the central triggers, the configurations that are broadcasted and the messages sent by neighbours.

Lastly, the setup method defines a number of neighbour objects, divided in upstream and downstream neighbours that store the information related to the neighbours the outstation is communicating with.

Each upstream neighbour object has a waker behaviour (a behaviour that executes with a delay) and a message sender behaviour associated with it that respectively keep track of the timeout delay and sent messages to the neighbour. A downstream neighbour only has a waker behaviour for its timeout, since message reception is controlled by a single behaviour for all messages.

Lastly, the rule KU has been implemented as a single behaviour. Also, since the stretch of road simulated does not make use of every rule, only the relevant rules from the set of rules have been implemented. All other functions of the agent can be taken over directly from the design.

6.1.2 Simulation environment

With only agents in the simulation, nothing is going to happen. Therefore an environment is added for the agents to evolve in. The objects that make up this environment are the following: the main class of the simulation, the simulation files, the outstation objects and the timer object.

The main class of the simulation launches the rest of the events. It first starts up a JADE environment with an agent manager and the central system agent. After that it creates all the configurations based on the names of the simulation files. With these configurations, the main class creates the outstation objects for each outstation. This object contains an osAgent object (see 6.1.1) and all the methods to interact with it (start, stop, receive sensor data, etc.).

The simulation files consist of one file per outstation named with the configuration and containing the minute information of the traffic state at that location. This file is read one line a minute (simulation minute) by an outstation object and transferred to the agent.

Also part of the simulation files is the central measure file and the historic MSI data if needed. These two files are used by the central agent to set central measures and to display what Rijkswaterstaat's system does at any point in time in order to compare the old with the new system.

Lastly, the timer object is started by the main class. This object executes periodically and goes through each outstation object in order to send the minute-data and send updates to the central agent. One execution of the 'run' method of the timer object represents one minute of simulation.

6.2 Simulations

We implemented our design and the three protocol variations from section 5.3.3 in order to simulate their behaviour in a number of test scenarios. We group the tests in two categories: rule validity and fault-tolerance. We then compare the results from the original implementation with our design and compare the three protocol variations with each other.

The rule validity test tests the system for stability and coherence over a long period of time. One month of historic data has been gathered and replayed on the different implementations. This test shows if the system still follows the correct rules even after a long period of time.

The data chosen are from highway 13 going from the Hague to Rotterdam or, more specifically, every outstation from kilometre 5,481 to 17,8 included. The reason this road was chosen is because of the high amount of traffic leading to

frequent traffic jams (almost every morning and evening on weekdays) and the abundance of data for that stretch of highway.

This simulation also gives an interesting comparison for bandwidth utilisation of the network.

In order to check fault-tolerance, specific failure scenarios are tested. The time it takes to recover as well as the recovered states are evaluated in order to see if the symbols are coherent and the recovery time meets the requirements. We start by testing single outstation failures at the same location in the symbol sequence. This allows us to test the network recovery time but also the sequence recovery time since this last one depends on the specific failure location in the sequence. Next, we test the recovery time of simultaneous outstation failures and finish with a stress test revealing how much failures the new design can handle before showing incorrect messages.

6.3 Results

6.3.1 Rule validity

Before we can check how well the new design performs over a long period of time, we need a way to quantify its behaviour. We do this by testing if the implementation correctly follows the rules set by Rijkswaterstaat. We describe a few of the formalizations used in this test in the following sections.

This test is also a great opportunity to evaluate each protocol variation's bandwidth demands. The last section covers this metric.

Automatic incident detection

As mentioned in previous sections, AID measures are the result of a congestion trigger. When congestion is detected by any sensor, an AID is generated. This should result in a speed restriction of at least 50 km/h preceded by a speed restriction of at least 70 km/h. We can see how well AID measures are generated by looking at their percentage of coverage of congested traffic.

$$\text{AID coverage} = \frac{\text{covered congestion}}{\text{total congestion}} \tag{6.1}$$

In this context, "covered congestion" is represented by an MSI symbol at least as restrictive as a 50 km/h speed restriction. This means that a cross or an arrow, which are more restrictive are also counted as "covered congestion".

To formalize this metric, we define a number of variables. Given a point $P(x, l, t)$ defined by a location x, a lane l and a time t, a function $C(P)$ that returns 1 if there is congestion at P and 0 if there isn't based on historic sensor data, and a function $S(P)$ that returns the priority level of the symbol at P according to the mapping in appendix B; equation 6.1 is redefined as follows.

$$AIDc(S) = \frac{\sum_{P:S(P)\leq 15} C(P)}{\sum_{P} C(P)} \tag{6.2}$$

System	Original	Action-share	Measure-share	Type-multicast
AID coverage	0.8728	1.0000	1.0000	1.0000
AID false positive	0.1031	0.0000	0.0000	0.0000
AID lead-in	0.9998	1.0000	1.0000	1.0000
AID 90 km/h restriction	0.0011	0.1607	0.0000	0.0000

Table 6.1: **Summary table of the AID metrics for the rule validity test.**

For a system that follows this rule perfectly, we expect $AIDc(S) = 1$.

This metric does not tell the complete story, since full coverage can also be obtained by permanently setting the speed limit at 50 km/h but this is not desirable since traffic flow would be seriously compromised. It is therefore important to check for false positives, that is, points P where there is no congestion but there is a speed restriction of 50 km/h. We can do so, since no 50 km/h restrictions are present in the simulate central measures.

$$AIDfp(S) = \frac{\sum_{P:13 \leq S(P) \leq 15} 1 - C(P)}{|\{P : 13 \leq S(P) \leq 15\}|} \tag{6.3}$$

For a system that follows this rule perfectly, we expect $AIDfp(S) = 0$.

Notice that we do not consider the complete spectrum of symbols smaller or equal to 15 for this formalization. We remove the symbols under 13 (crosses and arrows) from this metric, since they have a different origin than the AIDs. They would therefore interfere with the results of the AID false positives.

Yet another important metric regarding AIDs is their lead-in speed, that is, the speed displayed upstream of the location where congestion is detected. This rule states that every 50 km/h restriction needs to be preceded (in space) by a symbol at least as restrictive as a speed limit of 70 km/h. This can be checked by overlaying the points P with each other but shifting x by one as shown in the following equation.

$$AIDli(S) = \frac{|\{P : S(P(x, l, t)) \leq 15 \wedge S(P(x-1, l, t)) \leq 21\}|}{|\{P : S(P) \leq 15\}|} \tag{6.4}$$

For a system that follows this rule perfectly, we expect $AIDli(S) = 1$.

The results of the three metrics (AID coverage, AID false positives and AID lead-in) are shown in table 6.1. From these metrics we see that the new designs perform as commanded by the rules and much better than the original system. This is due to the fact that the original system has an intentional or unintentional lag in the computation of MSI upon reception of a congestion trigger. This can be due to the limited amount of computation power relative to the amount of outstations the central systems needs to control. This results in MSI updates that are not as frequent as the sensor updates. Symbols are therefore displayed later compared to the new designs and removed later as well. A similar issue is happening for central measures.

The last metric in the table is not shown in the formalizations but is mentioned in the design of the three protocols (see section 5.3.3). In that section we mention

that the action-share protocol can be too restrictive, since for the same set of symbols, different upstream continuations are possible depending on the type of measure (AID or central). In order to show this, we added a metric that demonstrates the over-restrictiveness of the action-share protocol.

The metric used is a formalization of the example described in section 5.3.3. Thus, if the rules were followed correctly, this metric should be equal to 0. In the case of the action-share protocol, we clearly see the over-restrictiveness of the protocol given that over 16% of the AID triggers are too restrictive.

Central measures

Central measures differ from AID measure in that their symbols are not always the same. For an AID measure, if there is congestion, the symbols will at least be a speed restriction of 50 km/h. This is not the case for central measures, since traffic managers have a wide variety of symbols to choose from (see appendix B).

Similarly to the AID measures, we evaluate central measures by looking at their coverage, false positives and lead-ins. In addition, the evaluation for the continuation across the road is added to the validity check.

The way we check for coverage, false positives and lead-ins is similar to the AID measures, but function $C(P)$ gets replaced by a function $M(P)$ that returns the symbol value (as defined in appendix B) set by a central measure at a point P. If no measure is set, $M(P)$ returns 44, the value for a blank symbol. This also leads to different formalizations than those used to check AID measures.

Checking coverage is formalised in the following equation.

$$CMc = \frac{|\{P : S(P) \leq M(P) \wedge M(P) < 44\}|}{|\{P : M(P) < 44\}|} \tag{6.5}$$

When $M(P) < 44$, a central measure applies at P, since any other symbol than a blank MSI is applied at that point. By checking if $S(P)$ is at least as restrictive as $M(P)$, we can evaluate the central measure coverage.

Checking for false positives is quite a challenge since some central measures have the same symbols as AID measures. It is therefore only possible to check for restrictions that do not have a speed limit of 50 or 70 km/h. On the other hand, more restrictive symbols, e.g. crosses and arrows, have the most impact on traffic and it is therefore important to check if they are not shown when not needed. We therefore only check false positives by checking for crosses and arrows that are misplaced.

Checking lead-in speeds is a slight change compared to equation 6.4. Only the constants used for the symbols change since the methodology remains the same.

In the case of central measures, one metric is added compared to AIDs: continuation across the road. For example, when an arrow is set it should be bordered by a speed 20 km/h faster than the speed associated with the cross it precedes.

Table 6.2 gives the results of the durability tests for central measures.

These metrics, again, show the improvement of our design over the current design. Between the three protocol variations, little difference is visible. The only remarkable difference comes through in the “Central measure across” metric, where the action-share variation performs worse than the other two vari-

System	Original	Action-share	Measure-share	Type-multicast
Central measure coverage	0.9938	1.0000	1.0000	1.0000
Central measure false positive	0.0301	0.0000	0.0000	0.0000
Central measure lead-in	0.9980	1.0000	1.0000	1.0000
Central measure across	0.9922	0.9981	1.0000	1.0000

Table 6.2: **Summary table of the central measure metrics for the rule validity test.**

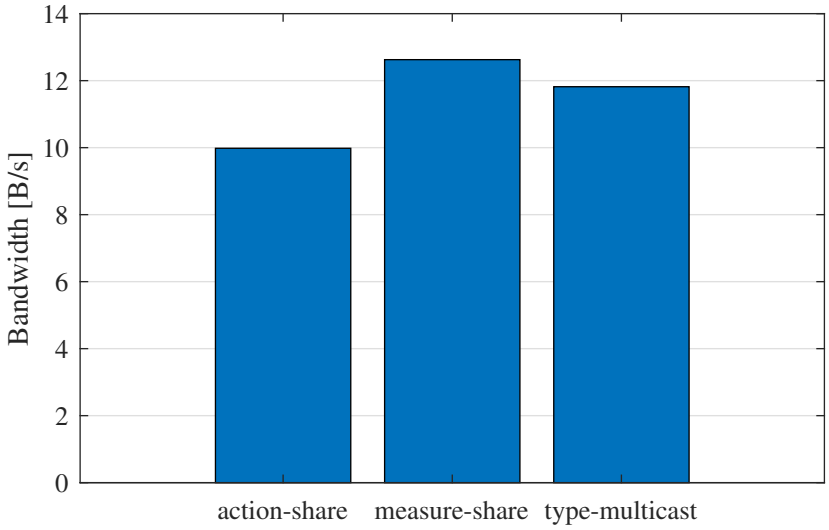


Figure 6.1: **Average bandwidth usage for an agent per protocol variation used.**

ations. The reason for this difference boils down to the over restrictiveness of that variation: AID measures, showing an extra 90 km/h compared to the other variations, interfere with the central measures.

Bandwidth

Figure 6.1 shows the average bandwidth usage at an agent per protocol variation. The bandwidth, measured in bytes per seconds, is very dependent on the implementation of the design and the amount of information sent per packet. In our implementation, the information density is not the highest since the content of the information is shared in plain text instead of being encoded in a binary array for example. Although the messages could have been smaller, their relative sizes allow for a meaningful comparison.

As expected, the action-share variation has the lowest average bandwidth requirements since outstation actions can be shared in small packets. The

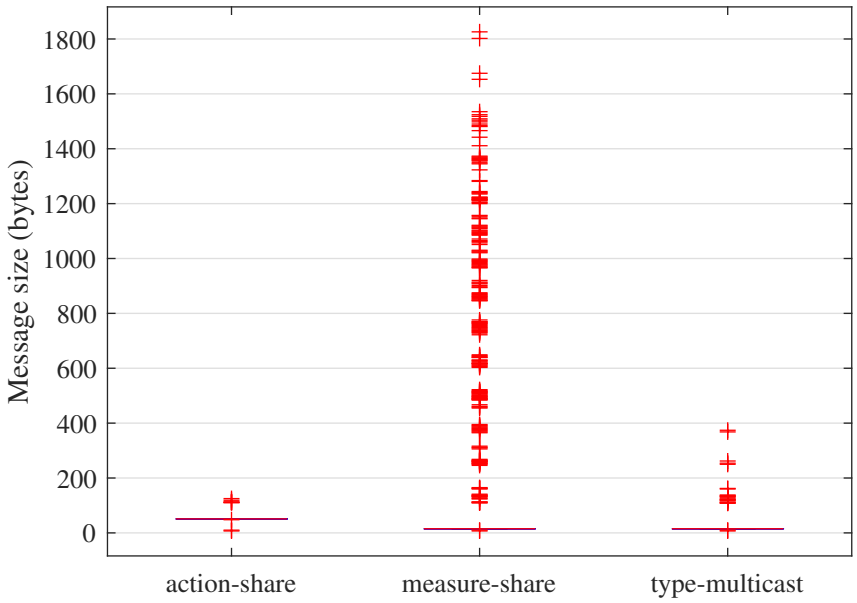


Figure 6.2: **Box plot of the message sizes for the three protocol variations during the bandwidth test.**

measure-share variation has an average bandwidth usage 26% higher than the action-share protocol variation. We expected this to be higher, but, outside of rush hour, very little measures apply on the highway, and so, the measure-share packets are empty. This is clearly visible in figure 6.2, where the median of the measure-share message size is lower than the median of the action-share message size. However, as visible from the same figure, some messages in the measure-share protocol are more than 10 times the size of the biggest message in the action-share variation. This explains the higher average bandwidth of the measure-share variation.

The type-multicast variation has an average bandwidth usage only 18% higher than the action-share variation. Not sharing all the measures applicable at a specific location results in smaller messages, as confirmed by figure 6.2. The median of the type-multicast message size is the same as that of the measure-share protocol but the maximum message size is close to 4.5 times smaller.

6.3.2 Fault-tolerance

For both an AID and a central measure, specific fault-tolerance tests have been performed in order to evaluate how the system reacts and within what time frame. A smaller system was modelled with 10 outstations for which both an AID and a central measure with a double cross apply, as shown in figure 6.3. This is a common scenario since, when the number of usable lanes is reduced, it often results in slow moving traffic. We then deliberately cause an outstation failure at different locations along the road to see how the system reacts. Both the resulting symbols are logged as well as the message exchanges between

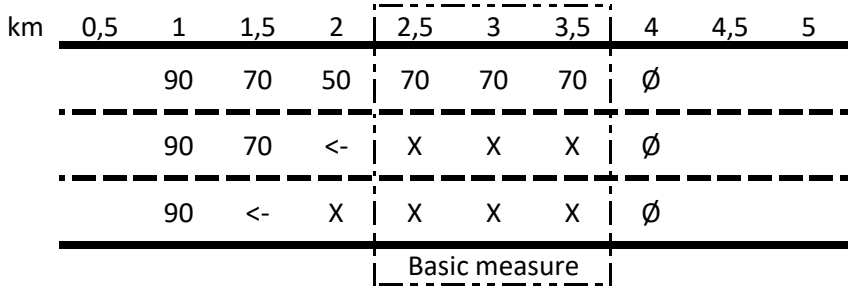


Figure 6.3: **The set of symbols shown during fault-tolerance tests under normal operation.**

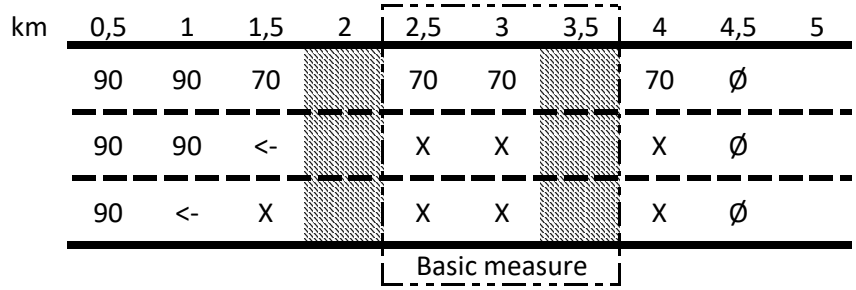


Figure 6.4: **Recovered state after successive failures of the outstations at kilometre** 2 **and** 3.5

outstations.

The most important metrics for fault tolerance are the accuracy of the recovered state compared to the original, non-failed, state of the network and the time needed to reach the new, recovered, state after the failure occurs. We will also have a look at the amount of simultaneous failures the system can handle before showing significant degradation in the symbol sequence.

Single fault

For this test, we disabled the outstations at kilometre 2 and 3.5 but not at the same time. We first disabled the one at kilometre 2, then the one at 3.5. We then let them rejoin in the same order. This way we can test 2 different single failures in the same simulation. It is also interesting to see the failures at different locations in the symbol sequence. For each of the three protocol variations, we ran the simulation 10 times.

All runs, regardless of the protocol variation used, resulted in a satisfying recovered state as shown in figure 6.4. The sequence of symbols created by the system after both failures have occurred and resolved, in all simulations, still limits the drivers from driving on the crossed off lanes at the basic measure, and traffic is slowed down and diverted on approach of the crossed off lanes.

The timing results are shown in tables 6.3 and 6.4.

Starting with table 6.3, we see that the delay for the network to recover from a failure, that is, the time between the failure of the outstation and the moment all outstations are back into a new stable configuration, is very different between

	Delay in seconds		
	action-share	measure-share	type-multicast
Network recovery after failure	$\mu = 39.31$ $\sigma = 8.93$	$\mu = 34.30$ $\sigma = 7.10$	$\mu = 43.64$ $\sigma = 5.28$
Symbol sequence recovery after failure	$\mu = 55.84$ $\sigma = 8.89$	$\mu = 50.09,$ $\sigma = 8.21$	$\mu = 36.28$ $\sigma = 8.54$
Network recovery after rejoin	$\mu = 10.97$ $\sigma = 0.36$	$\mu = 11.51$ $\sigma = 0.43$	$\mu = 11.39$ $\sigma = 0.41$
Symbol sequence recovery after rejoin	$\mu = 61.37$ $\sigma = 0.41$	$\mu = 63.82$ $\sigma = 1.75$	$\mu = 26.06$ $\sigma = 2.66$

Table 6.3: **Average and standard deviation for the recovery times for the outstation at kilometre 2 based on 10 measurements.**

	Delay in seconds		
	action-share	measure-share	type-multicast
Network recovery after failure	$\mu = 35.48$ $\sigma = 9.22$	$\mu = 31.81$ $\sigma = 9.84$	$\mu = 43.34$ $\sigma = 1.99$
Symbol sequence recovery after failure	$\mu = 36.55$ $\sigma = 8.29$	$\mu = 33.97$ $\sigma = 8.41$	$\mu = 22.63$ $\sigma = 5.28$
Network recovery after rejoin	$\mu = 10.93$ $\sigma = 0.23$	$\mu = 11.30$ $\sigma = 0.46$	$\mu = 11.80$ $\sigma = 0.73$
Symbol sequence recovery after rejoin	$\mu = 10.55$ $\sigma = 0.10$	$\mu = 10.63$ $\sigma = 0.16$	$\mu = 10.78$ $\sigma = 0.39$

Table 6.4: **Average and standard deviation for the recovery times for the outstation at kilometre 3.5 based on 10 measurements.**

the three protocols. The measure-share protocol variation is the fastest whereas the type-multicast variation is the slowest. However, the standard deviation has the same order of magnitude for all three protocol variations.

Recall from the previous chapter that we needed the timeout value to meet the requirement that $2T < 60$s. We therefore chose $T = 20$s. This condition is related to the fact that it is possible that after reaching one timeout, both parties that need to connect with each other, are not ready. This would lead to another try after reaching another timeout.

This is visible in the measure-share variation, where the mean time for the network to recover after failure ($\mu = 34.30$) is close to being halfway between T and $2T$ and the standard deviation is quite large since the delays are close to either T or $2T$. We expect the difference between the action-share variation and the measure-share variation to reduce with an increased sample size.

For the type-multicast protocol variation, the higher average delay time ($\mu = 43.64$) is due to multiple outstations re-establishing connection with each other, which can take multiple broadcasting occasions separated by multiple timeouts.

To check the requirement stating that the system should recover from a failure in less than 60 seconds, the maximum values of the network recovery time are more meaningful than the mean values. The maximum values for the network recovery time are 48, 47.64 and 49.62 seconds for respectively the action-share, measure-share and type-multicast variations. Based on these values, all protocol variations meet the requirement.

Moving on to the recovery of the symbol sequence after failure, the type-multicast variation performs the best. It is even faster than network recovery delay. This is because the outstations can reorder their neighbours, when using this variation, as soon as they detect the failure. Therefore, the outstations are able to adapt their displays earlier than the other 2 variations who will need multiple message exchanges to arrive at a recovered sequence of symbols.

When the failed outstation rejoins the network, both the network and the sequence have a much more defined recovery time, since this part of the protocol, for all variations, has less uncertainty (low standard deviation). Thanks to the "dump" messages and the immediate broadcast of the configuration upon rejoining the network, the system recovers in about 1 period of the protocol. This period was implemented as $\tau = T/2$ (where T is the timeout value set at 20s), thus, $\tau = 10$s. This corresponds to the values in the table.

After the outstation has rejoined it takes a while for the sequence of symbols to reset itself. Here, the type multicast system is again the fastest ($\mu = 26.06$) due to its ability to connect to multiple outstations at the same time. This allows the first outstation to reconnect with the recovered one to immediately notify multiple outstations about the new symbols to display.

In table 6.4, the network recovery delays, both after failure and rejoin, are very similar to the ones in table 6.3. The symbol sequence recovery is, in both cases, much faster. For all three variations, the amount of time needed to attain a new correct symbol sequence is reduced due to the specific location in the sequence where the fault occurs. This is because only a central measure is applied followed by an end-of-restriction symbol which means that it involves less outstations than the previous failure. Specifically, for the type-multicast variation, the benefit of communicating with more than 2 neighbours becomes obvious: after the failure, the outstations have recovered the sequence of symbols ($\mu = 22.63$) before the network does ($\mu = 43.34$).

Measurement point	Delay in seconds		
	action-share	measure-share	type-multicast
Network recovery after failure	$\mu = 36.61$ $\sigma = 8.77$	$\mu = 42.86$ $\sigma = 5.76$	$\mu = 45.98$ $\sigma = 2.48$
Symbol sequence recovery after failure	$\mu = 50.46$ $\sigma = 7.87$	$\mu = 51.90$ $\sigma = 6.77$	$\mu = 34.49$ $\sigma = 10.63$
Network recovery after rejoin	$\mu = 12.01$ $\sigma = 0.65$	$\mu = 12.80$ $\sigma = 0.96$	$\mu = 29.77$ $\sigma = 2.11$
Symbol sequence recovery after rejoin	$\mu = 62.91$ $\sigma = 0.76$	$\mu = 65.16$ $\sigma = 1.47$	$\mu = 26.80$ $\sigma = 1.96$

Table 6.5: **Average and standard deviation for the recovery times for simultaneous failures based on 10 measurements.**

When cumulatively increasing the number of failed outstations, all three protocol variations managed to keep drivers away from the crossed of lanes as long as possible. The full sequence of symbols during this cumulative failure test is shown in appendix C.

Multiple faults

When multiple faults happen at the same time, the chance that, upon broadcasting, outstations get a sub-optimal configuration assigned to them is increased. In order to test the systems resistance to these faults we disabled the same two outstations as before but this time, at the same moment. We then simulated this case 10 times, all of which resulted in the correct pattern of symbols as shown in figure 6.4. The delay times have been recorded in table 6.5.

The first notable result is the network recovery time both after failure and after rejoin. For the action-share and the measure-share variations, the delay times are very similar with the exception of the network recovery delay after failure for the measure-share protocol, but we expect this to go down with increased sample size. The type multicast system has higher times in both cases, again, due to multiple outstations per failure needing to reconnect with one another.

Symbol sequence recovery delays are close to the results from table 6.3. The reason for this, is that the failure of the outstation at kilometre 2 has the highest recovery delay of the two failures, thus, when both fail, the symbols around kilometre 2 will take the most time to recover.

In chapter 5, we established that one of the reasons our approach outperforms an alternative using a predefined selection of backup neighbours, is the constant time needed to recover from a failure. Figure 6.5 shows the time delay between the moment of failure and the moment the network has recovered for multiple amounts of random simultaneous failures. We note that no reconfiguration delay is longer than 50 seconds for all the protocol variations and that the maximums and minimums of all box plots are close to each other. This indicates that the delay time is constant in its spread.

Also, for all variations, no clear trend emerges from 2 to 6 simultaneous faults. For 7 simultaneous faults, the type-multicast variation box-plot changes considerably. The same happens for the other two variations at 8 simultaneous

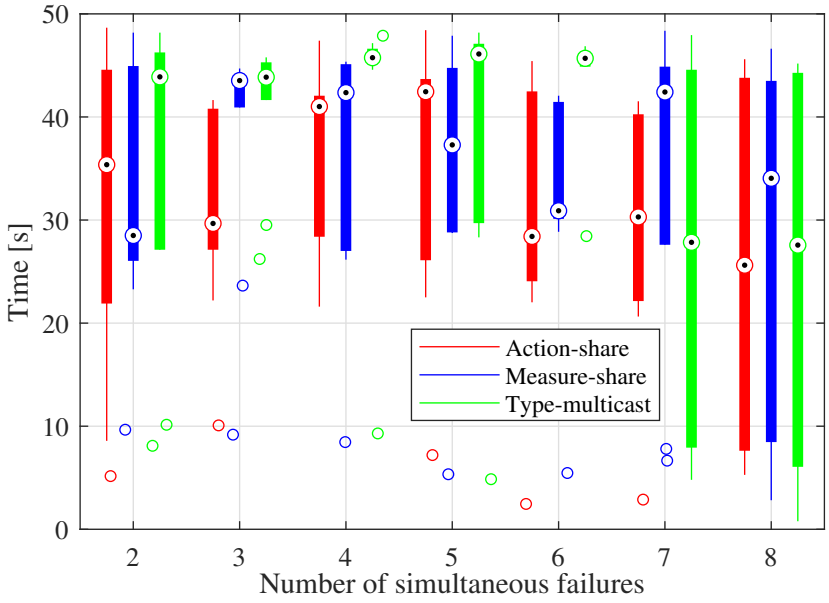


Figure 6.5: **Recovery delay after simultaneous outstation failures for all three protocol variations.**

failures. This change comes down to the fact that, for these amounts of failures, the number of outstations that remain operational is 3 and 2 respectively, which means that the chances of a successful reconfiguration on the first try increase.

We also established that the theoretical maximum amount of broadcasts required upon simultaneous reconnection of multiple outstations increases quadratically with the number of outstations reconnecting. This would impact the time needed for the reconfiguration after multiple outstations (re)join the network. Figure 6.6 shows the time delay between the moment of (re)joining and the moment the network has recovered for multiple amounts of random simultaneous failures. From this figure, a trend is visible for the action-share and measure-share variations: the time needed to recover after multiple outstations (re)join the network increases with the number of outstations (re)joining. This trend is visible due to the box plots beginning with a median of 10 seconds and a 95th percentile under 30 seconds for 2 and 3 simultaneous failures but ending concentrated around 30 seconds. The type-multicast variation, due to its multiple outstation connections, always take around 30 seconds to reconnect. However, the maximum amount of time needed seems to increase with the number of outstations (re)joining. A larger sample size is needed to confirm this hypothesis. In any case, due to the reduced likelihood of the worst case decreasing, no quadratic increase is visible.

All protocol variations perform exactly alike, regarding the symbols displayed, under multiple simultaneous failures, as visible in figure 6.7. The sequence respects the rules until all but 1 outstations fail. When only one outstation remains online, the system is not able to reconfigure and thus, does not know what the correct course of action is. The symbols last shown at that outstation

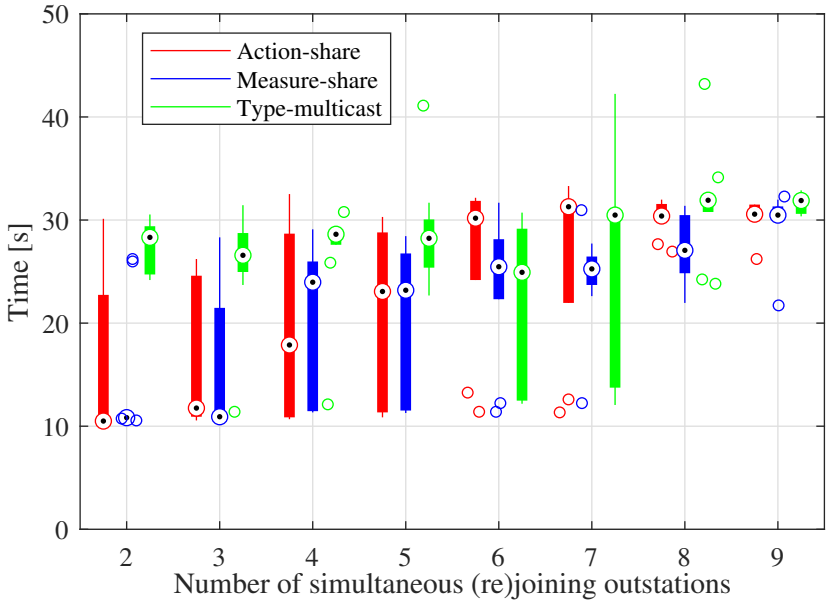


Figure 6.6: **Recovery delay after simultaneous outstation (re)join the outstation network for all three protocol variations.**

stay in place.

6.4 Conclusion

The design, as presented in the previous chapter, performs as expected, both in terms of rule validity (its ability to mimic the original system over a long period of time) and in terms of fault-tolerance. We experimented with the three different protocol variations described in chapter 5: the action-share, measure-share and type-multicast variations.

Overall, all three protocols outperformed the original system in the rule validity test. However, we saw that the action-share variation is indeed over-restrictive in some cases and, therefore, does not respect the current rules.

In terms of network load, the action-share variation is the most lightweight, followed by the type-multicast and the measure-share variation. The measure share-variation revealed itself to be very demanding in terms of peak bandwidth, with messages 10 and 4.5 times larger than the action-share and type-multicast variations respectively.

The second set of tests were focused on the timing of the fault-tolerant aspect of the design. The design is able to recover from any number of failures, as long as one outstation remains operational, and it is able to recover from a single failure within 60 seconds. In these tests, the action-share and measure-share variations performed very similarly since their way of communicating with neighbouring outstations is the same (only the content differs). As the type-multicast variation communicates with more than two neighbours, as soon as a failure is detected the sequence of symbols is able to reorganize quicker.

Regardless of the number of simultaneous failures, the distribution of the time

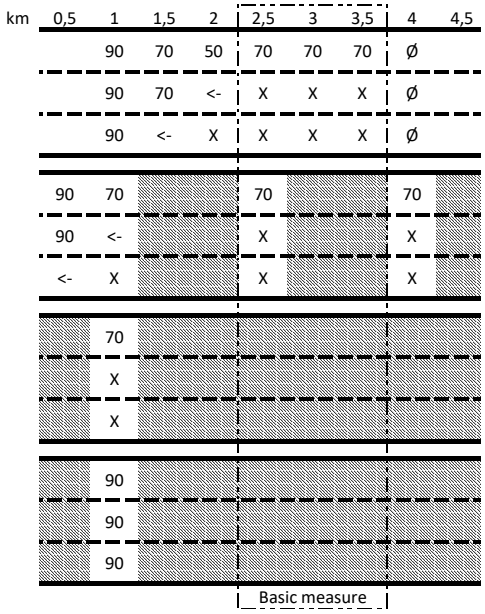


Figure 6.7: **Symbol sequences after simultaneous failure of multiple outstation. The first sequence shows normal operation, the second sequence when 5 outstations fail simultaneously, the third when 8 outstations fail simultaneously and the last when all but 1 outstation fail. Greyed out MSIs represent failed outstations.**

delay between the moment of failure and the moment the network has recovered from the failure remains largely constant. Furthermore, the maximum delay does not exceed 50 second in the current implementation. We can therefore state that our design meets the fault-tolerance requirements set, even for up to 8 simultaneous outstation failures.

Based on the implementation we provided, we would argue that the type-multicast variation is the best choice. The action-share protocol variation has the downside that it is over-restrictive for Rijkswaterstaat, since it is not aware of the origin of the symbols it displays. Moreover, the bandwidth increase that the type-multicast variation demands are relatively low for the benefit of knowing the full measure. The type-multicast variation is also the fastest during symbol sequence recovery and, in terms of network recovery, this variation's worst recovery times are not significantly different from those of the other two variations.

Chapter 7

Conclusions & Future Work

7.1 Conclusions

In this thesis, we presented a design for a fault-tolerant multi-agent distributed flow control system capable of replacing the current motorway traffic management (MTM) system in place at Rijkswaterstaat. Compared to other traffic management multi-agent solutions, our design is tolerant to any number of faults, be it consecutive or simultaneous faults, if at least one agent remains operational.

We started by analysing the faults that cause a problem in the current network of outstations. These faults result in a loss of communication between the central system and the outstation which means that sensor data cannot be received and actuation of the MSIs is not possible. These faults happen on average 3.86 times per hour and concern a vast majority of the outstations under Rijkswaterstaat's control. We were not able to determine one or more definitive causes for the faults that occur. We were, however, able to conclude that, given the significant correlation between the moments the faults occurred and the moment Rijkswaterstaat typically performs midday maintenance, this last activity is a likely origin of a big part of the communication faults. Other possible factors, like old age or outdate communication media, are remedied by the new hardware brought in by the iWKS project.

We established that, currently, there is no MAS suitable for adaptation into a fault-tolerant distributed flow controller MAS. We did, however, find a system called TRYSA_2 whose design methodology was useful in our case. Also, the AMAS technology shows great potential for adaptation into a MTM-MAS. We therefore designed our own MAS and communication protocol using these findings.

This design is based on knowledge units as used in TRYSA_2. We created a MAS where every outstation is associated with an agent and all run the same identical software. Together with the protocol that we designed for this MAS and a configuration file, the resulting system is able to mitigate outstation failures in less than 60 seconds. We created 3 different variations of the communication protocol which vary in terms of content and way of sharing it: action-share, measure-share and type-multicast.

In order to make sure the design and it's variations perform as expected, we

devised 2 sets of tests. The first test proposed a way to check if the systems would perform correct actuations given a historical set of sensor inputs. We compared the results with the historical output of the original system and concluded that the system is able to react faster and correcter than the original system according to the rules set by Rijkswaterstaat.

We also compared the bandwidth requirements for the protocol variations and concluded that the action-share protocol, although more restrictive in actuation, is the least bandwidth demanding of the 3 variations. The goal of the second set of tests was to check how quick the different variations of the protocol react to faults and if the resulting actuations are correct. All three protocols passed the correctness tests and restored their network under 60 seconds even for multiple outstations failing simultaneously. In these tests, the type-multicast protocol came out ahead due to it's ability to very quickly adapt the MSI's when a fault is detected.

Depending on the specific implementation case, a different protocol variation will perform the best. The action-share protocol performs the best when all controller actions are the result of a unique input. However, this often is not the case, which results in wrong actions like we saw with the over-restrictive symbols. The measure-share and type-multicast variations both perform correctly in cases where similar actions have different inputs as a cause. Between these two variations, the type-multicast variation outperformed the measure-share variation in both bandwidth usage and symbol sequence recovery speed.

Overall, we recommand using the type-multicast variation as it performs better than the action-share variation and has lower bandwidth requirements than the measure-share variation.

We see our design as a fault-tolerant platform for future control algorithms. The modelling of the agent using knowledge units allows future implementations of the design to easily change the content of the KUs without altering the functionality. We hope to see Rijkswaterstaat and other system designers explore the possibilities of our fault-tolerant multi-agent distributed flow control system in the future.

7.2 Future Work

The rules KU, as described in this thesis, now holds a copy of the rules already set by Rijkswaterstaat. However, the rules are static and do not evolve over time or change depending on other conditions besides the triggers described in this thesis. It would be possible to modify the content of this KU to integrate more complex rules or even rules that evolve over time using machine learning.

Another improvement that can be made is to remove the central system as a trigger generator. At Rijkswaterstaat, for example, one project called "Slimme camera's" (smart cameras) works towards modifying the current cameras, located alongside the highway, to automatically open and close hard shoulders during rush hour. Research at Rijkswaterstaat [26] also shows that accidents could also be detected using cameras or other local sensors. This would transform a central trigger into a local trigger and remove functionality from the central system.

While flow redirection was not part of this thesis' research, it is a part of flow control. It would therefore benefit the system to add the capability of

redirecting (part of) the traffic in such a way that, when an actuator responsible of the redirection fails, the redirection measure can still be applied to the flow network.

In the introduction of this thesis, we argued that autonomous cars would benefit from communicating with the highway infrastructure. The design, as developed in this thesis, does not provide such communication. However, the given design prepares the infrastructure for such an application. Our agents are capable of reliably acting on there own, without the intervention of a central system. And so, it is possible to equip these agents with the means to communicate with cars so that sensor and actuator information can be shared. New protocols and algorithms would need to be implemented but these can easily be added on top of, or to, the platform we designed thanks to the use of knowledge units.

Bibliography

[1] Jeffrey L. Adler and Victor J. Blue. A cooperative multi-agent transportation management and route guidance system. *Transportation Research Part C: Emerging Technologies*, 10(5):433–454, 2002.

[2] A. L. Almeida, S. Aknine, J. -. Briot and J. Malenfant. Plan-based replication for fault-tolerant multi-agent systems. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, pages 7–13. IEEE, April 2006.

[3] K. Almejalli, K. Dahal and A. Hossain. An intelligent multi-agent approach for road traffic management systems. In *2009 IEEE Control Applications, (CCA) Intelligent Control, (ISIC)*, pages 825–830. IEEE, July 2009.

[4] Carole Bernon, Marie-Pierre Gleizes, Sylvain Peyruqueou and Gauthier Picard. Adelfe: a methodology for adaptive multi-agent systems engineering. In *Engineering Societies in the Agents World III*, pages 156–169, Berlin, Heidelberg. Springer Berlin Heidelberg, 2003.

[5] Antonio Bicchi and Lucia Pallottino. On optimal cooperative conflict resolution for air traffic management systems. *IEEE Transactions on Intelligent Transportation Systems*, 1(4):221–231, 2000.

[6] Romaric Breil, Daniel Delahaye, Laurent Lapasset and Éric Féron. Multi-agent systems for air traffic conflicts resolution by local speed regulation and departure delay. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–10. IEEE, 2016.

[7] P Bublies. Fuel of the future: data is giving rise to a new economy. *Economist (United Kingdom)*, 413, May 2017.

[8] D. Capera, J. -. George, M. -. Gleizes and P. Glize. The amas theory for complex problem solving based on self-organizing cooperative agents. In *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.* Pages 383–388, Linz, Austria. IEEE, June 2003.

[9] José Cuena and Martín Molina. *Ksm: an environment for design of structured knowledge models.* In *Knowledge-Based Systems: Advanced Concepts, Techniques And Applications.* Chapter 7, pages 217–245.

[10] Scott A. DeLoach and Juan Carlos Garcia-Ojeda. O-mase: a customisable approach to designing and building complex, adaptive multi-agent systems. *Int. J. Agent-Oriented Softw. Eng.*, 4(3):244–280, November 2010.

[11] A. M. Garcia-Serrano and D. Teruel Vioque. Fipa-compliant mas development for road traffic management with a knowledge-based approach: the track-r agents. In *the TRACK-R agents, Challenges in Open Agent Systems '03 Workshop*, pages 622–625. IEEE, October 2003.

[12] A. H. Ghods and A. Rahimi-Kian. A game theory approach to optimal coordinated ramp metering and variable speed limits. In *2008 Chinese Control and Decision Conference*, pages 91–96, Yantai, Shandong. IEEE, July 2008.

[13] Zahia Guessoum, Nora Faci and Jean-Pierre Briot. Adaptive replication of large-scale multi-agent systems–towards a fault-tolerant multi-agent platform. In *International Workshop on Software Engineering for Large-Scale Multi-agent Systems*, pages 238–253. Springer, 2005.

[14] Josefa Z Hernández, Sascha Ossowski and Ana Garcıa-Serrano. Multiagent architectures for intelligent traffic management systems. *Transportation Research Part C: Emerging Technologies*, 10(5-6):473–506, 2002.

[15] Jared Hill, James Archibald, Wynn Stirling and Richard Frost. A multi-agent system architecture for distributed air traffic control. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*. American Institute of Aeronautics and Astronautics, August 2005.

[16] Bryan Horling, Brett Benyo and Victor Lesser. Using self-diagnosis to adapt organizational structures. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 529–536, New York, NY, USA. ACM, 2001.

[17] Ramachandra Kota, Nicholas Gibbins and Nicholas R. Jennings. Self-organising agent organisations. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, pages 797–804, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[18] Sehl Mellouli. A reorganization strategy to build fault-tolerant multi-agent systems. In *Advances in Artificial Intelligence*, pages 61–72, Berlin, Heidelberg. Springer Berlin Heidelberg, 2007.

[19] M. Nguyen-Duc, J. -. Briot and A. Drogoul. An application of multi-agent coordination techniques in air traffic management. In *IEEE/WIC International Conference on Intelligent Agent Technology, 2003. IAT 2003.* Pages 622–625. IEEE, October 2003.

[20] Sascha Ossowski, José Cuena and Ana García-Serrano. A case of multiagent decision support: using autonomous agents for urban traffic control. In *Progress in Artificial Intelligence — IBERAMIA 98*, pages 100–111, Berlin, Heidelberg. Springer Berlin Heidelberg, 1998.

[21] Gauthier Picard, Sehl Mellouli and Marie-Pierre Gleizes. Techniques for multi-agent system reorganization. In *Engineering Societies in the Agents World VI*, pages 142–152, Berlin, Heidelberg. Springer Berlin Heidelberg, 2006.

[22] Andrey Popov, Robert Babuška, Andreas Hegyi and Herbert Werner. Distributed controller design for dynamic speed limit control against shock waves on freeways. *IFAC Proceedings Volumes*, 41(2):14060–14065, 2008.

[23] Giannis Roussos and Kostas J Kyriakopoulos. Completely decentralised navigation of multiple unicycle agents with prioritisation and fault tolerance. In *49th IEEE Conference on Decision and Control (CDC)*, pages 1372–1377. IEEE, 2010.

[24] Kagan Tumer and Adrian Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, 255:1–255:8, New York, NY, USA. ACM, 2007.

[25] RT Van Katwijk, B De Schutter and J Hellendoorn. Multi-agent coordination of traffic control instruments. In *2008 First International Conference on Infrastructure Systems and Services: Building Networks for a Brighter Future (INFRA)*, pages 1–6. IEEE, 2008.

[26] Karen van Vianen. *Automatic Incident Detection with Floating Car Data.* Master Thesis, Delft University of Technology, Delft, The Netherlands, 2017.

Appendix A

Calculation of the worst case number of broadcasts after multiple outstations (re)join the network

When multiple outstations (re)join a network of outstations, the case that leads to the worst reconfiguration times, is the one where multiple adjacent outstations join. As an illustration, we take the sequence of outstations shown in figure A.1.

In figure A.1, we assume that the action-share variation is in place and that outstations only communicate with 2 neighbours. A similar analysis as the one we will show here is possible to do with all three protocol variations presented in chapter 5.

The figure shows 5 outstations labelled from A to E, with A located the furthest upstream and E the furthest downstream in the traffic flow. The sequence starts when outstations B, C and D (re)join the network at exactly the same time. This results in all three of them broadcasting their configuration at almost the same time (the specific underlying platform start up time can vary). Since outstations only pick up a new upstream neighbour, clearly the worst case would occur if outstation E selects B as its new upstream neighbour. This would lead to A getting a dump message from E and broadcasting its configuration. Again, the worst case occurs if D were to pickup A's configuration and selecting it as a new upstream neighbour. This is however not the best configuration and, since C's and D's broadcasts might still be in the network, E can pickup C's configuration and switch neighbours. Outstation C therefore sends a dump message to B which, in turn, broadcasts it's configuration. This can be picked up by agent D which then dumps agent A for B as its new upstream neighbour. A will then broadcasts it's configuration again, which might get picked up by agent C. We still haven't reach the optimal configuration, and thus, agent E will pickup D's configuration, dumping C in te process. This time, C's broadcast is pickup by D who dumps B. B broadcasts again which leads C to switch neighbours and dump A. It turn, A broadcasts it's configuration one last time which let's B

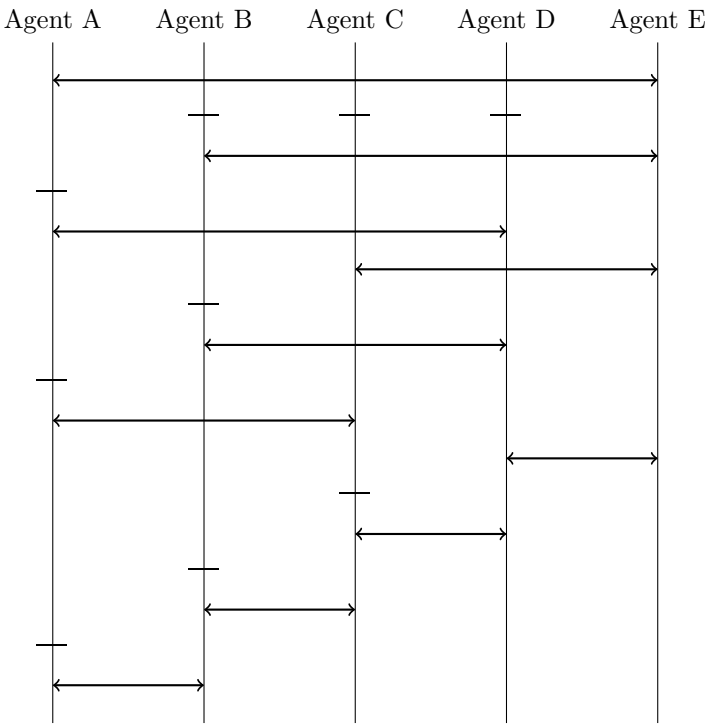


Figure A.1: **Worst case sequence of broadcasts when 3 adjacent outstations (re)join an existing network of agents. A double headed arrow represents an established connection and a dash in an agent's sequence represents a broadcast of that agent.**

select his most optimal upstream neighbour. The system has now reached the optimal overall configuration.

The total number of broadcasts in this example is 9. We can generalise this number for n outstations (re)joining the network by looking at what happens at each outstation during the process described above. Starting with outstation A, we see that from the start of the sequence, where it was communicating with outstation E, to the end where it communicates with B, it has broadcasted its configuration 3 times. As 3 new outstations joined, 3 new options for a downstream neighbour where available. In the worst case, all three get selected before the optimal one is chosen and, since it require a broadcast for an outstation to change its downstream neighbour, 3 broadcasts where needed. If we would have started the sequence with B already communicating with E, outstation B would see 2 new downstream outstations to communicate with and would therefore, in the worst case broadcast it's configuration twice. Similarly, outstations C and D would respectively see 1 and 0 new outstations to communicate with. This effect would continue on, if more outstation would (re)join the network.

Omitting the first broadcasts form the (re)joining outstations, we see that the amount of broadcasts decrease per outstation from upstream to downstream and this subtotal of broadcasts is $3 + 2 + 1 + 0 = 6$. Since this effect continues for outstation also (re)joins at the same time, for every n number of outstations (re)joining, this subtotal can be described by the following equation.

$$B_{sub}(n) = \sum_{k=0}^{n} k \tag{A.1}$$

Equation A.1 is also known as the n^{th} triangle number (which can also be visualized by the sideways triangle created by the broadcasts in figure A.1) for which the sum can be reduced to the following equation.

$$B_{sub}(n) = \frac{n(n+1)}{2} \tag{A.2}$$

Lastly, we need to add the number of broadcasts we omitted previously to have the total number of broadcasts. We omitted exactly n outstations, namely, the number of outstations that (re)joins. We therefore get the following value for $B(n)$, the worst case number of broadcasts for n agents (re)joining the network.

$$B(n) = B_{sub}(n) + n = \frac{n(n+3)}{2} \tag{A.3}$$

The calculation and especially figure A.1 clearly shows that the chance of this worst case happening decreases very quickly when n increases. For example, for the worst case form figure A.1 we need to have the broadcast from A, after being dumped by agent E, to arrive at D before any other broadcast. This is highly unlikely since outstations B and C start at the same time as D and will thus clearly send their configuration before A does. With increasing n the chance that this worst case happens therefore decreases dramatically.

Appendix B

Symbol Priorities

Value	Symbol
0	
1	
2	
3	
4	
5	
6	 [OB]
7	
8	
9	
10	
11	
12	
13	

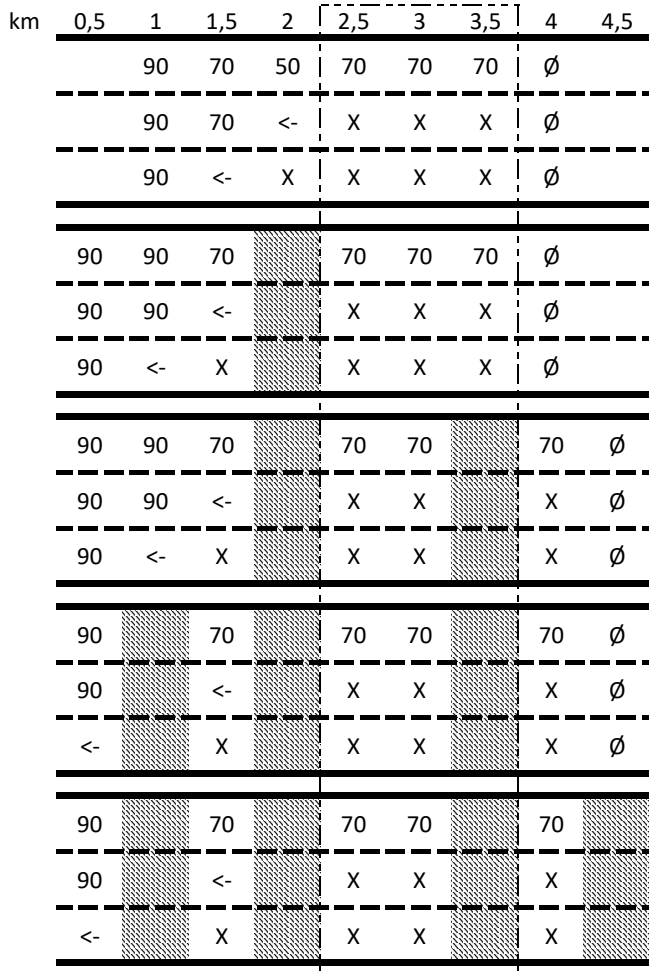
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	

35	
36	
37	
38	
39	
40	
41	
42	
43	
44	

Table B.1: **Table mapping traffic symbols to their priority. The higher the number, the lower the priority. OB, stands for "Overruling blank" which can be used to force a blank MSI.**

Appendix C

Cumulative failures



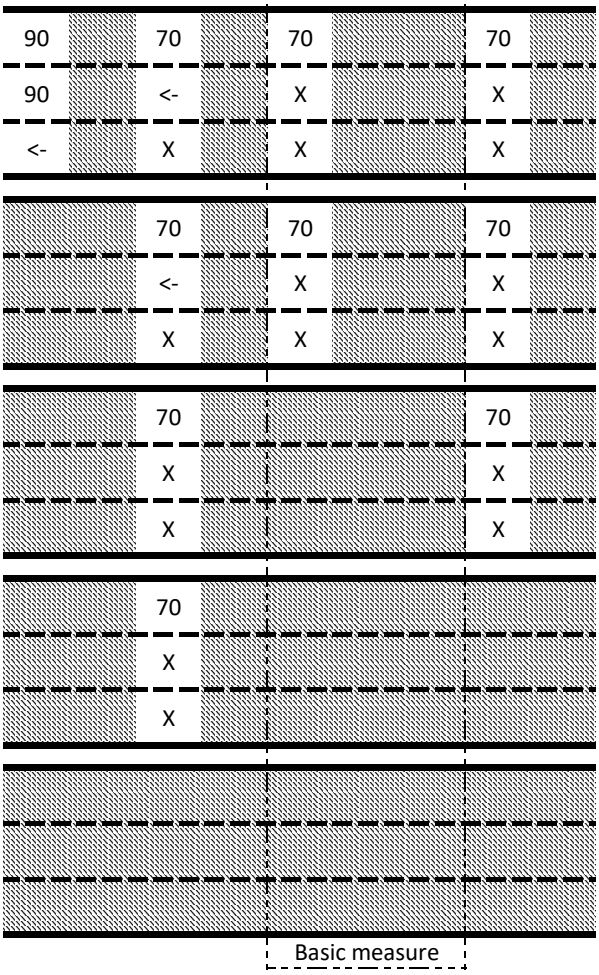


Figure C.1: **Symbol sequences over time (top to bottom) during cumulative failures of all but one outstation. The basic measure still holds throughout the failure sequence.**