NTNU
Norwegian University of
Science and Technology

Virtual (floating) Context Sharing between Vehicles

Generating and Sharing Context Information within an Autonomous Network of Vehicles

Øyvind Risan

Master of Science in Communication Technology

Submission date:	July 2010
Supervisor:	Steinar Andresen, ITEM
Co-supervisor:	Evtim Peytchev, Nottingham Trent University

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Within a network of vehicles there might be groups of vehicles that share some common condition or interpretation about their surroundings. The information should be uniform within the group, and migrate through the whole network. The merging of this information is context dependent since different conditions might need to be merged differently. It is a goal to have one interpretation about the overall status within the whole network.

This report will investigate how information about different conditions might be gathered, distributed and merged to form an overall interpretation about the status in the network. This must include some method for determining the shape of the areas with the different conditions, a method for distribution of information and an analysis of how the different conditions should be treated.

Assignment given: 15. January 2010
Supervisor: Steinar Andresen, ITEM

PREFACE

This document concludes my master’s degree within communications technology at the Department of Telematics at the Norwegian University of Science and Technology (NTNU).

The project has been conducted during the first half of 2010, from January to June. All the work has been done while I was an exchange student at Nottingham Trent University in Nottingham, Great Britain.

I would like to thank my supervisor and mentor at Nottingham Trent University; Dr Evtim Peytchev. His insight and ideas have been a valuable contribution to my work, and helped develop the scope of the thesis. The weekly meetings with Peytchev and his PhD students have given me inspiration and challenges that I’m grateful for.

I’m very grateful for all the help and support I have received from Professor Steinar Andresen at the Norwegian University of Science and Technology. Without his help I would never have gotten such an interesting assignment, or the opportunity to experience another University and the culture of a different country.

A special thanks to Gamati Emadin and Yueyue Li for their contributions in our long and inspiring discussions about the simulators behaviour, functionality and possibilities.

I need to thank my fiancée for her support, as I have spent most of my time in front of the computer completely oblivious to my surroundings. I promise that in the future the house's main computer shall not be unusable bogged down with heavy simulations.

All the material used to write this report is available electronically. This includes the text, simulator codes, simulation result files, java program, figures, graphs and tables. Due to the size and number of simulation trace files it has been impossible to store these. The result files are stripped down and compressed versions of the trace files.

ABSTRACT

This thesis investigates some of the potential within Intelligent Transportation Systems and Inter-Vehicle Communication. The aim is to find a method for spreading information throughout an ad hoc network of vehicles with as little impact on the available resources as possible.

Two problems are identified and investigated. The first problem is determining the border of a known set of vehicle positions. This is solved by producing a list of edges that encircles vehicles which contain the same information. These edges form a border that can be used to reduce the amount of data needed to represent the status in that particular area. By reducing the amount of data the load on the limited transmission capacity can be reduced.

Based on mathematical calculations relating to the relative position of vehicles, an algorithm was devised to detect and order the vehicles that contribute to the border of an area. It is possible to alter the shape of the border by removing selected points on the border. This makes it possible to make the border convex instead of concave.

The behaviour of this border finding algorithm is illustrated by a java program. The graphical representation also displays some of the mechanisms used to determine the border.

The second problem is related to how information can be distributed efficiently to all vehicles within a network.

As there are limited capacities in most vehicle-to-vehicle-networks there is a need to focus on reducing the overall load in such networks. The aim is to distribute the available information to every vehicle in the network as efficiently as possible, without acting at the expense of speed and flexibility.

The choice of information exchange method has a serious effect on the transmission load, and the amount of interference that is present in the network. “Pure flooding” is the most basic and elementary of the methods available. The two main strengths of this method are the reactive behaviour, and the ability to adapt to any network configuration.

This thesis suggests an alternative method for broadcasting information throughout the network, without having to use “pure flooding”. The method is called ICE (“Information Combined and Exchanged”). This method collects the available data and aggregates this to one single new message. To do this a delay is introduced between a vehicle receiving a message and retransmitting it. All messages received during this time are included in the following transmission.

ICE does not limit what kind of information that could be exchanged, but examples of useful information might be warnings, points of interest, infotainment and advertisements.

Based upon the behaviour of ICE a simulator was made to test the performance. The simulator made it possible to compare the performance of both ICE and “pure flooding” by measuring various variables. The main parameters were the size of the introduced delay and the number of vehicles in the simulation area. The combinations of the parameters and broadcast methods lead to 96 simulations from which a great amount of information could be extracted.

The results showed that ICE outperformed “pure flooding” with regards to transmission load, interference and lost messages. At selected delays the time each vehicle is involved in the communication is also superior to “pure flooding”.

The most significant findings are:

- ICE generally performs at its best with a delay of about 50 ms
- Any individual vehicle’s involvement time might be reduced by as much as 74%
- Depending on the number of unique messages the number of sent messages can be reduced by 77% to 86%
- The interference can on average be reduced by 66%

“Pure flooding” is outperformed by ICE in almost all the situations that were tested in the simulator in this thesis. A situation with very sparsely populated networks and long delays is the exception. In this case “pure flooding” is faster, but might waste more resources. Implementation of dynamic delays would make ICE suitable for this scenario as well.

CONTENT

FIGURE INDEX

TABLE INDEX

GRAPH INDEX

EQUATION INDEX

CODE INDEX

ABBREVIATIONS

CALM	Communication Access for Land Mobiles, earlier known as "Communications, Air-interface, Long and Medium range", provides a layered solution that enables communications between vehicles/infrastructure
CVIS	Cooperative Vehicle-Infrastructure System, large EU-funded project for ITS
GNSS	Global Navigation Satellite System, include (GPS, GLONASS, COMPASS, Galileo)
GPS	Global Positioning System, a GNSS build U.S. Department of Defence, provides global coverage
ICE	Information Combined and Exchanged, exchange algorithm to reduce transmission load while spreading information throughout a network
IPv4	Internet Protocol version 4, limited number of free addresses left
IPv6	Internet Protocol version 6, larger address space than IPv4, suitable for mobility
ITS	Intelligent Transportation System, merging of communication equipment and transport infrastructure
IVC	Inter-Vehicle Communication Systems
MANET	Mobile Ad hoc NETwork
NAM	Network Animator, used to illustrate network behaviour in a simulator
NS2	Network Simulator second edition, uses UNIX environment to simulate network behaviour
TCL	Tool Command Language, scripting language used in UNIX
UDP	User Datagram Protocol, best effort network protocol
V2V	Vehicle-to-vehicle
VANET	Vehicle Ad hoc NETwork
WLAN	Wireless Local Area Network

DEFINITIONS

802.11a	54 Mbit/s WLAN using 5 GHz
802.11b	11 Mbit/s WLAN using 2.4 GHz
802.11g	54 Mbit/s WLAN using 2.4 GHz
802.11n	600 Mbit/s WLAN using 5 GHz
802.11p	Referred to as WAVE, WLAN with support for ITS using the licensed band of 5.9 GHz
Ad hoc	Decentralized and self configuring wireless network
Concave	Curving in or hollowed inward
Convex	Curving out or bulging outward
Galileo	A GNSS build by EU and ESA, does not provide global coverage at the moment
Geo-Routing	Routing based upon the geographical position of the receiver, managing the data-stream in the right direction
Polygon	Closed circuit, composed of a finite sequence of straight line segments
Proactive routing	Mapping and evaluation of the available paths before the paths are needed
Pure flooding	Exchange algorithm that retransmits any new messages/packets
Reactive routing	Paths are constructed/discovered when, or if, they are needed
Vehicle-to-Vehicle	V2V, communication between vehicle installations, subject to other challenges than fixed or pedestrian installations
WAVE	802.11p, robust WLAN with improved latency characteristics, designed for use in ITS

1. INTRODUCTION

This chapter presents the theme, structure, methodology and limitations of this thesis. A small portion of background material is presented to set the scope of this thesis and describe the main challenges this thesis will try to illuminate.

1.1. Background

The future is wireless, and new areas of application for wireless communication are emerging at an impressive pace.

Many car manufactures have already introduced wireless communication equipment in their vehicles [1]. Volvo has implemented “Volvo on Call” [2] and BMW has implemented “BMW Assist” [3] in several models [4] to provide help in case of accidents and collisions. BMW has also implemented a system called “BMW Online” [5] to provide real time information for drivers and passengers in some of their models. These systems are based on cell phone technology, using existing base stations, long range and centralised servers. The next step would be to fit equipment in vehicles to provide Inter-Vehicle Communication (IVC) [6]. This development is covered by the term Intelligent Transportation System (ITS) [7]. ITS encompasses the merging of communication equipment with transport infrastructure in an effort to manage traffic and provide new services.

IVC is a research area that receives a fair amount of interest from the automotive industry and from researchers all over the world. CVIS [8] is an example of such a research project that has provided much solid and interesting research.

The desire to build an *“ad hoc network, which consists of a set of nodes (vehicles) that communicate through radio frequencies, without the use of a beforehand deployed infrastructure and without the use of a centralized administration”* [9] is driven by the desire to provide safety, entertainment and comfortable driving to the huge number of individuals that use vehicles on the roads every day.

Vehicle-to-vehicle networks are generally ad hoc based, and broadcast is the most flexible method of information distribution within such networks. The available transmission capacity is limited, and the load on this capacity is dependent on the rate and size of information that is broadcasted. Relevant information for broadcasting might be; accident warnings, driving conditions, weather, announcements, infotainment [10] and even advertisements with audio/video content.

Some information about various conditions might be limited to geographical areas. Weather and driving conditions are examples of this. Information about these areas might be useful to other vehicles within the network, and therefore needs to be exchanged. Determining the borders of these areas can help reduce the amount of data to be transmitted throughout the network.

There is information within the network that needs to be exchanged to all the other vehicles in the network. This information can be of any variety, there is no limit to what kind of information that can be exchanged. The challenge is to do this as efficiently as possible without sacrificing too much of the available resources.

This thesis will try to illuminate the various aspects of area-border-determination and information exchange within vehicle-to-vehicle interaction.

1.2. Problem description

If all vehicles within a geographic area experience[1] the same reality there is no need to spread information about every vehicle inside the area to the rest of the network. It will suffice to spread information about the border of the area. The questions then are:

- "Is it possible to determine the border of an area based upon a list of vehicle positions within the area?"
- "How will the border construction influence the homogeneity of the information?"
- "What benefits can be drawn from border determination?"

Relevant information available within a defined network might be unevenly distributed, thus demanding a need for exchange of information between vehicles. A limiting factor is the bandwidth that can only handle a limited amount of data simultaneously before interference causes information loss. With this in mind the questions then are:

[1] Homogenous interpretation of the environment

- "How can an exchange be conducted to reduce the load on the transmission capacity?"
- "How does the selected method perform against an alternative method in a simulator?"

This thesis will try and answer these questions regarding vehicle-to-vehicle interaction, and provide a platform for further research and development.

1.3. Limitations

This thesis is as technology neutral as possible. The aim is to keep the material as flexible as possible and not enforce technology specific details on the solutions chosen. By doing this the results of this research could be applied to other areas, or systems, without being bound by the chosen technology or architecture such as CVIS [8], CALM [11] or VANET [12]

The simulations are based upon WLAN 802.11 [13]. Considering the high market penetration and the low cost of the equipment this seems a natural choice of communication technology to be used within vehicle-to-vehicle communication.

In the simulations XY-coordinates is used as this is very natural in the simulation environment. XY-coordinates are of course not suitable for real life implementations. GPS [14] or Galileo [15] are the most likely choices for physical implementations, and the results should apply to them as well. Several companies, such as Cisco [16], supply positioning services within wireless networks. These technologies and services are not considered as an option since they rely on centralized servers. The networks considered in this thesis are ad hoc without any demands on infrastructure.

Security and privacy is not considered in this thesis. Security is left out to clarify the explanations and ideas behind the methods used. All communication will be considered "friendly" and all the messages are genuine. Privacy is a topic that will receive much attention as ubiquitous networking develops. The selection of what information to exchange, and possible priorities, is a significant challenge and views might change as the technology develops. These are highly important subjects, but the management of privacy will not influence the behaviour and mechanisms that are presented in this thesis and are therefore disregarded.

1.4. Structure

This thesis has 3 main parts. The first part (chapter 2) explains the theory behind the border-determination and the theoretical foundation regarding exchange of information, as they were presented in chapter 1.1.

The second part (chapter 3) is regarding implementation and proposes possible solutions. In this chapter models are presented, and flowcharts will illustrate the solutions without being bound by choices regarding technology. This chapter will deal with one of the possible solutions to the questions presented in chapter 1.2.

The last of the main parts (chapter 4) is regarding simulation and testing of the solutions, the data is gathered using a simulator. The necessary program code can be found in the Appendixes.

Chapter 5 is the discussion where the data gathered during the simulations from chapter 4 are presented, and relations between the results are investigated.

The conclusion is chapter 6. This chapter tries to answer the questions presented in chapter 1.2 based on the discussion in chapter 5.

In-depth material, and material that is too comprehensive to include in the main text will be presented in the Appendixes.

1.5. Methodology

The methodology used to produce the results in this thesis is based on a loop structure.
The ideas or challenges lead to a hypothesis that is the foundation for the system model. The system model is tested using a simulator and the results are analyzed. Depending on the analysis of the results the hypothesis, or the model, is redesigned.
This is illustrated in Figure 1.

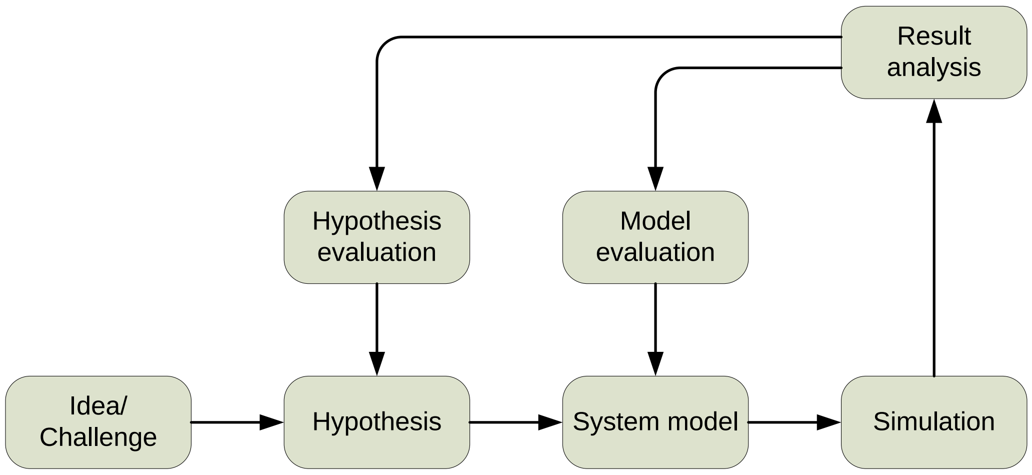


Figure 1 Methodology

Figure 1 explains the methodology used in this thesis. The hypothesis dictates the system model which in turn is tested through simulation of functions proposed in the model. The results given from the simulation lead to alterations of the hypothesis or model which in turn will be tested in the next simulation. The process of making the simulation perform in correspondence with the system model involves hidden iterations where the functionality of the simulator is measured against the proposed model. Only when the simulator is working in correspondence with the model can the simulation results be used to evaluate the hypothesis or the system model. The results presented in this thesis are the outcome of the last iteration through the loop in Figure 1.

2. BACKGROUND & THEORY

This chapter presents the work of authors that have produced similar projects. Some of their findings that relate to this thesis will be emphasised. The chapter also presents some theoretical foundation for the choices that will be made in chapter 3 regarding area-determination, cell construction and information exchange.

2.1. Related work

Inter-vehicular communication is a research field that has received a lot of attention the last years. Several studies have investigated information propagation within V2V networks.

The differences between IVC-networks and regular MANET's is studied in the article by "Blum, Eskandarian and Hoffman" [17], and they discovered some significant differences. Their findings include: limited movement (predictable direction), high speeds give rapid changes in the topology, driver influencing behaviour, frequency fragmentation and limited effect of network redundancies. The nature of the networks in this thesis would be more similar to IVC than to MANET.

The authors "Wischhof, Ebner and Rohling" [18] has conducted a study where electronic maps are used together with WLAN 802.11 [13] and GPS [14] to analyse information based on road segments. This article utilizes the benefits of individual analysis (by the vehicles) to produce an overall system status. This approach is similar to the method explored in this thesis.

In an article by "Zhao and Cao" [19] a proposed solution is discussed where the vehicles store information to be retransmitted when there is a willing receiver in the proximity. This kind of selective retransmission provides valuable opportunities to solve challenging topographical obstacles as well as interference related challenges.

"Nadeem, Shankar and Iftode" [20] have in their article studied how bidirectional mobility on well defined routes affect the information propagation in VANET [12]. The three options are to transmit to the vehicles that move in the same direction, vehicles that move in the opposite direction or to vehicles moving in both directions. All these solutions provide interesting possibilities regarding safety, entertainment and comfort.

Different methods of forwarding has been proposed by “Wu, Fujimoto, Guensler and Hunter” [21] in their article. The three options: opportunistic forwarding, trajectory based forwarding and geographical forwarding gives a wide range of possibilities and very good flexibility.

The aspect of rapid and efficient broadcast of emergency messages is discussed in the article by “Sukdea and Gihwan” [22]. Their solution based upon defer-times that are dependent on the density of available neighbours has a good effect when the communication is point-to-point. Their solution shows a significant improvement compared with “pure flooding”.

The study of intelligent forwarding done by “Korkmaz, Ekici, Özgüner and Özgüner” [23] provides useful methods when infrastructure is available to act as repeaters. As a method for point-to-point communication with infrastructure it is efficient and it is possible to adapt some of the features to work on broadcast within an ad hoc network as well.

The article by “Uthansakul and Uthansakul” [24] gives a comprehensive interpretation of how positioning might be done in a WLAN network, without the use of GPS equipment at each node. Instead of relying on the unstable signal strength, the article suggests using time delay as the measure for distance, and hence the position. The study has been conducted on indoor equipment, and the results are promising. This method is less suitable in an outdoor ad hoc network, as it demands some infrastructure and a fixed reference point.

The study of throughput with moving vehicles done by “Brickley, Shen, Klepal, Tabatabae and Pesch” [25] have established that the speed is a limiting factor on throughput. Their results show a significant drop in throughput as the speed increases (520 kb/s at 50 km/h and 250 kb/s at 100 km/h) when using WLAN. This illustrates that there are significant limitations to the transmission capacity as vehicles move.

2.2. General theory

Vehicles, that are fitted with communications equipment, means of determining their own position and some sort of computing power, can form an ad hoc network without having to rely on infrastructure or other fixed installations. The topology of a network of moving vehicles will change as vehicles change position in relation to each other, some leave the network, and others are introduced. Ad hoc networks are designed to handle these changes in topology as they are self configuring and thus very flexible.

As the vehicles move while communicating the signal propagation will change rapidly during the communication sessions. This movement causes alterations to frequencies, signal strength, interference and delays that in turn affect the throughput and will limit the transmission capacity in the network.

Modern vehicles are equipped with large numbers of sensors as these are needed to maintain normal vehicle operation. Some of these sensors generate data that could be utilized to provide useful services if combined with vehicle position and then shared with the rest of the network.

Each vehicle would have the ability to process data, both received and detected by internal sensors, to provide services to the driver, or passengers, of the vehicle.

2.3. Initial gathering of information

Information gathering in the network can be done in several ways. It would be advisable to choose a method that provides fast and accurate results without using more of the network recourses than necessary.

There are two distinct types of information available; information that is location specific and information that covers a certain area. The information that covers a certain area is made up of empirical data about location specific conditions. These areas will later in this report be referred to as cells.

The difference between information that covers an area and information that is specific to a single location is not very big. The location specific information is just a very small cell, consisting of a single point, while the information that covers an area has a border consisting of one or more locations.

At the time of information exchange the information will be extracted from vehicles and prepared to be spread throughout the network. The information will now represent the conditions a particular vehicle experienced at a specific location and at a certain point in time.

Information can be collected by the vehicle continuously, and then shared with the rest of the vehicles in the network at suitable discrete intervals, or by request.

2.4. Constructing cells

The information gathered must be organized and analyzed to make an interpretation about the status in the network. In most cases it would be interesting to make cells containing various conditions. The size of these cells would depend on the condition in question, different conditions would need different handling.

There is no significant difference between a large and a small cell (except from the geographical area it covers). They both have vehicles that construct a border, and they can both have vehicles that reside on the inside of this border. This means that it is possible to use the same algorithm to construct the border for both small and large cells. The limitations are related to the conditions themselves, and not the size of the cell. This will be discussed further in chapter 3.3.

The shape constructed by drawing a border connecting all the outermost vehicles is often referred to as a Polygon [26], but will in this report be called the cell border, or just cell. The shape of the cell will vary based on the position of the vehicles that constitutes the border, and how the cell-border is constructed. There are two general shapes of cells that need to be addressed. These two shapes are: Concave and Convex [27].

2.4.1. Concave and convex cells

A concave cell will have indentations inwards in the cell, and internal angles that are greater than 180°. This is illustrated in Figure 2A.
A convex cell will not have indentations, and any internal angle will be smaller than 180°. This is illustrated in Figure 2B.

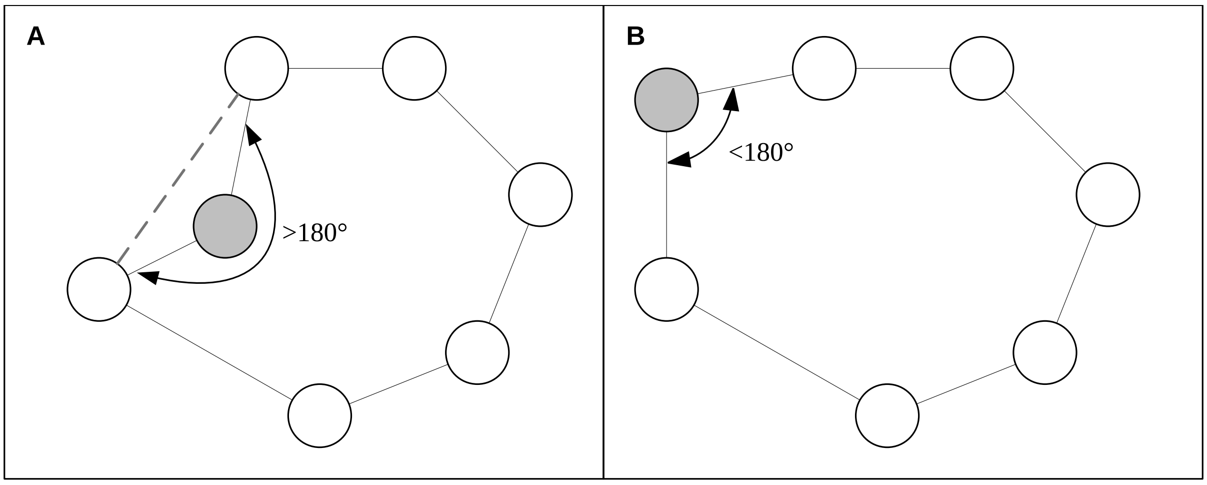


Figure 2 A. Concave, B. Convex

A concave border seems a little unnatural, as the condition would have very sudden changes in direction. A more natural explanation is that we lack information from the points that would have continued the convex border. This lack of information is definitely not the same as to say that the condition does not apply in the space left by the concave angles, but it means that we have not received any confirmations that it applies in that area.

Any concave cell can be decomposed into a series of convex polygons [28]. It is not necessary to split the concave cells into convex sub-cells to get a fair approximation of the convex border. It is possible to remove the concave area by removing the vehicle that causes the sharp angel from the list of vehicles on the border. As illustrated by the grey dotted line in Figure 2A. The mathematical reasoning for this is described in chapter 3.3.3. The removal of these vehicles from the border would cause the concave cell to become convex and probably more truthful. The decision of whether to keep, or remove, concave elements is dictated by the nature of the condition in question. There might be conditions where it would be natural and correct to see/have sharp changes in border directions.

2.5. Merging of cells

When a cell is constructed its borders will consist of a list of vehicles that defines the border of some condition. By exchanging these lists of vehicles it is possible for all the vehicles in the network to learn the shape and positions of the cells.

It is possible for several cells within the same network to present the same conditions independently. Depending on the condition in question merging the information from several small cells to construct one large cell could rationalize the relevant communication to the rest of the network.

Constructing the larger cell is done in the same way as constructing the smaller cells. A border line between the outermost vehicles in all the cells that are to be merged would provide an outline of the condition. This is illustrated in Figure 3. Deciding whether or not to remove the concave elements depends on the context and the conditions in question.

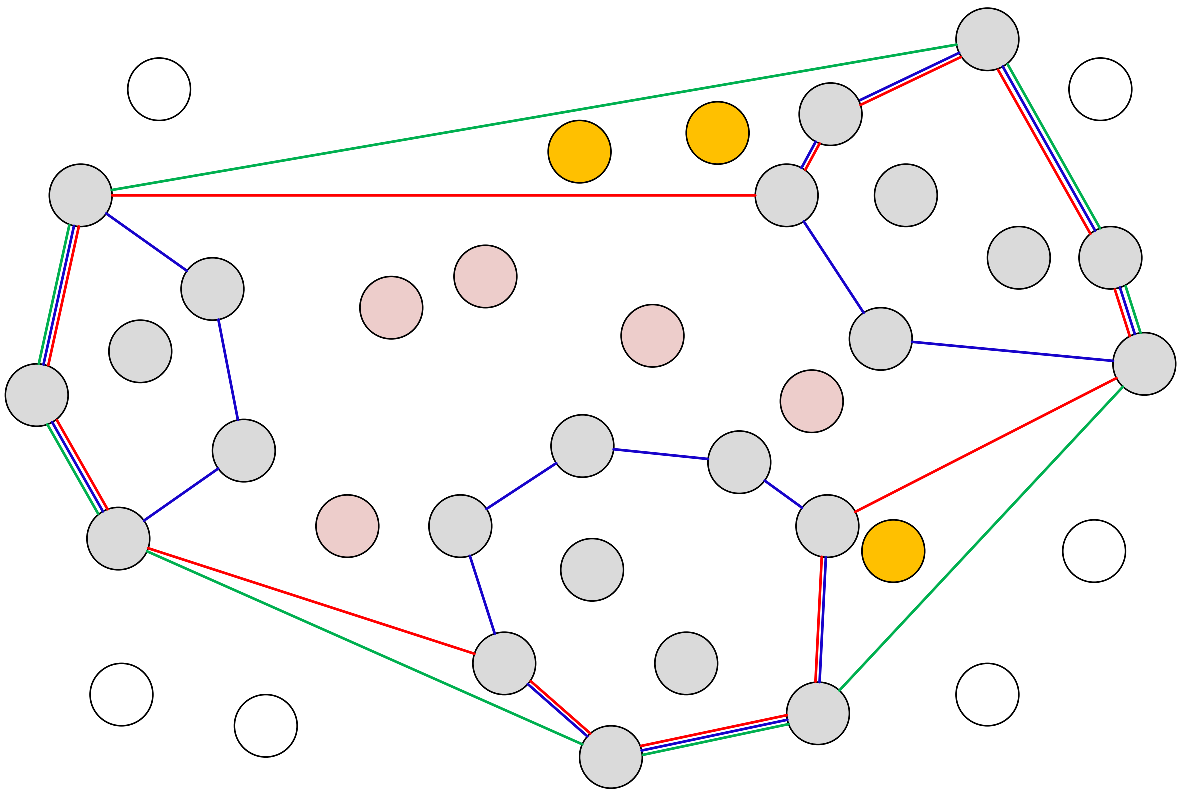


Figure 3 Merging of three distinct cells

- The blue line represents the original cell borders.
- The red line represents the new border, with concave elements.
- The green line represents the new border without the concave elements.
- The gray circles represent the vehicles that have the condition.
- The pink circles represent the vehicles that are included unintentionally inside the new borders.
- The yellow circles represent the vehicles that are included unintentionally as concave border elements are removed.

Merging the three cells into one large affects the pink vehicles; they are suddenly included in a cell that they do not actually belong in. Removing concave elements includes even more non relevant yellow vehicles.

The context, the condition in question and the number of vehicles unintentionally included in a merged cell must be considered before the decision to merge is made.

2.5.1. Different levels of detail

The merging of cells will produce a new larger cell. The list of vehicles that define the large cell border should contain fewer entries than the sum of all the border vehicles of the small cells. The reduction in number of vehicles involved in the borders means that the large cell would contain less data, thus representing less strain on the transmission capacity.

A downside of having large cells is that it provides less detail. The lack of detail is not necessarily a problem, as long as the details are available if needed. This makes it possible to have different levels of detail based upon need. A vehicle that is a long distance away from a condition does not need to know details about every vehicle in each cell, it just needs to get a rough interpretation about what lies ahead. When the vehicle is closing in on the cell, there is need for more detail. The vehicles within a cell would need to have detailed information about their surroundings, but the further away from the cell a vehicle is the less detail is needed. This is illustrated in Figure 4.

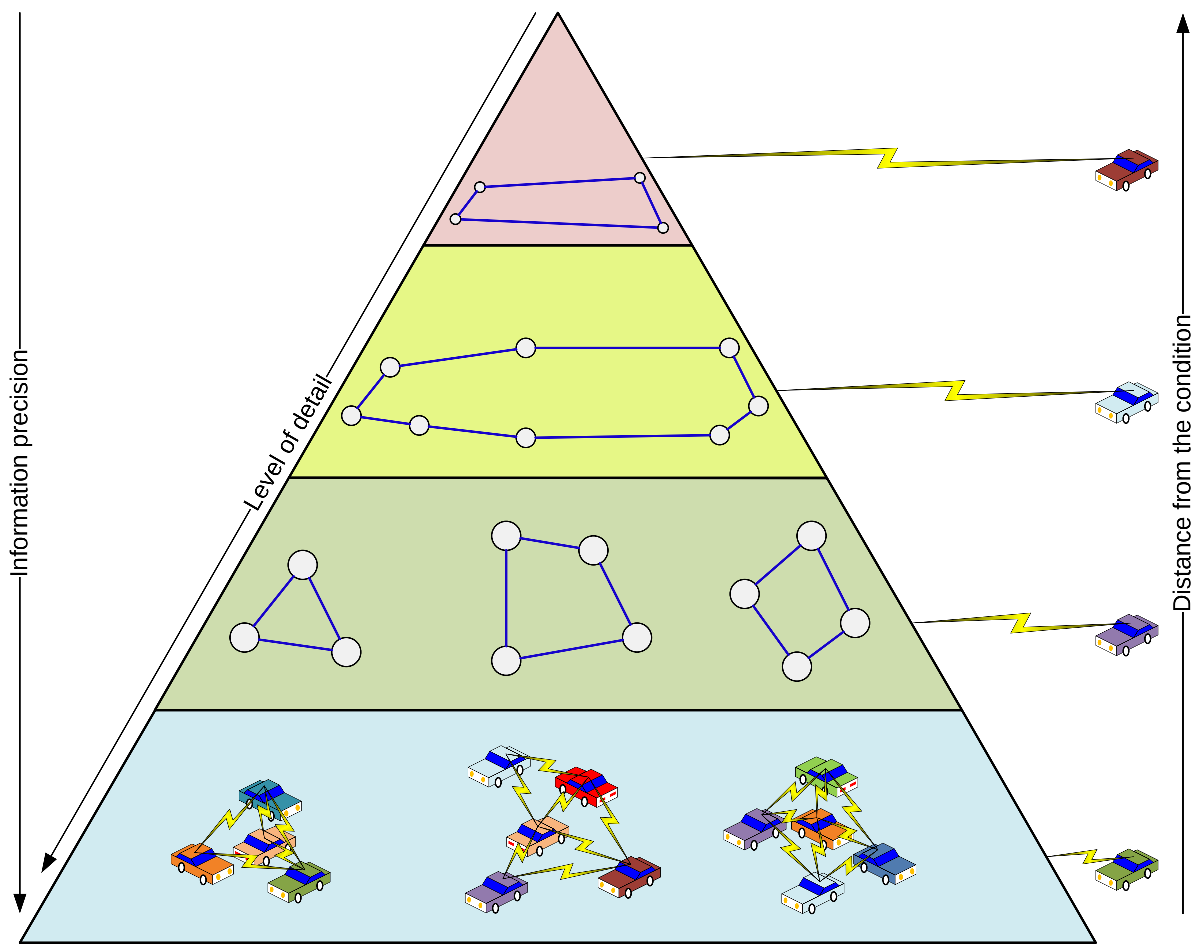


Figure 4 Different levels of detail, information precision and distance from conditions

As can be seen in Figure 4 the information precision decreases as the distance increases. This will cause the vehicles to have slightly different views of the overall status in the network depending on their distance from the condition.

At the blue level in the pyramid in Figure 4 the vehicles know everything about each other. In the green level the internal structure of the cell has been removed, and only the border is announced. At the yellow level the cells are merged together to form a larger cell. In the red level more of the border is removed to give a rougher estimate, and the cell from the yellow level might be merged together with other cells. At which distances the detail levels change, and what the effect of the change is, depends on the context, and should follow rules according to the different conditions.

2.6. Information exchange

2.6.1. Full scale information exchange

The aim is to spread all the information available within the network to all the vehicles in the network in such a way that they can make up their own interpretation about the overall status in the network.

The simplest and most robust way of doing this is by “pure flooding” [29], where every vehicle sends out their interpretation to all the other vehicles. A disadvantage with “pure flooding” is the number of messages generated when every vehicle sends out their interpretation of the status and these messages are retransmitted by all the other vehicles. Equation 1 gives a mathematical representation of these numbers.

$$Number\ of\ messages\ sent = (Number\ of\ vehicles\ involved)^2$$

Equation 1 Number of messages sent in “pure flooding”

If there are more than two transmitters within reach of each other, the vehicles will receive the same message several times. Examining a network where all the vehicles can receive wireless communications from all others, without intermediate vehicles, it is possible to identify the number of redundant messages in the network thus quantifying one of the major disadvantages of “pure flooding”. The number of redundant messages can be calculated using Equation 2.

$$N = number\ of\ vehicles\ within\ reach\ of\ each\ other$$
$$Number\ of\ redundant\ messages = N * (N - 1)$$

Equation 2 Number of redundant messages with “pure flooding”

Depending on the density of vehicles the number of redundant messages might become substantial, and this will unfortunately introduce interference, thus causing loss of messages.

Routing based upon knowing the position of all the neighbouring vehicles is an alternative, but it relies on proactive measures in building a map of the positions of all vehicles in the system. Geo-Routing [30] is such a method; it bases its routing on knowing the start and finishing position of a message. This is fine in point to point communication, but it is not as well suited for broadcasting information. The Geo-Routing approach would introduce another level of overhead when determining the positions, and is not suitable if there are large topographical changes in the network. Due to the speed of vehicles and the rapid changes to the topology, a reactive method would probably be more appropriate.

It would be preferable to find a method of information exchange that is reactive and does not rely on unique messages for each vehicle, but instead produced some sort of aggregated message, which contains processed information. The vehicles would have to make educated decisions on how, and what, to transmit. This would demand more computational power in each vehicle, but it would reduce the load on the transmission capacity.

A vehicle that receives a message containing new information would combine this new information with the information already stored within the vehicle itself to produce a new rationalized and updated interpretation of the network status. This interpretation is sent out as a new message to update the network. For this method to have any benefits over “pure flooding” two or more messages need to be merged in a vehicle before retransmission. The earlier the messages are merged, the more the number of messages in the system is reduced.

Only messages that contribute with information to the network should be forwarded. The flow and merging of information is illustrated in Figure 5. The four colours represent different types of information that need to be spread to all the other vehicles. This information can consist of anything; ranging from warning messages to restaurant reviews. The benefits of merging information before transmission are evident between “hop 2” and “hop 3”. Instead of transmitting four single messages containing only 25% of the information the vehicle transmits a single message containing all the available information. This aggregation has a profound effect on the transmissions that follow in “hop 4” and “hop 5”.

In Figure 5 the dotted lines in “0 hop” illustrates vehicles that are capable of communicating with each other. The solid arrows in “1 hop” to “5 hops” illustrate the flow of messages between vehicles. At “2 hop” and “4 hop” 3 of the lines are dotted, this is to indicate transmissions that do not actually provide new information to the receivers, but the vehicles still need to transmit their new messages.

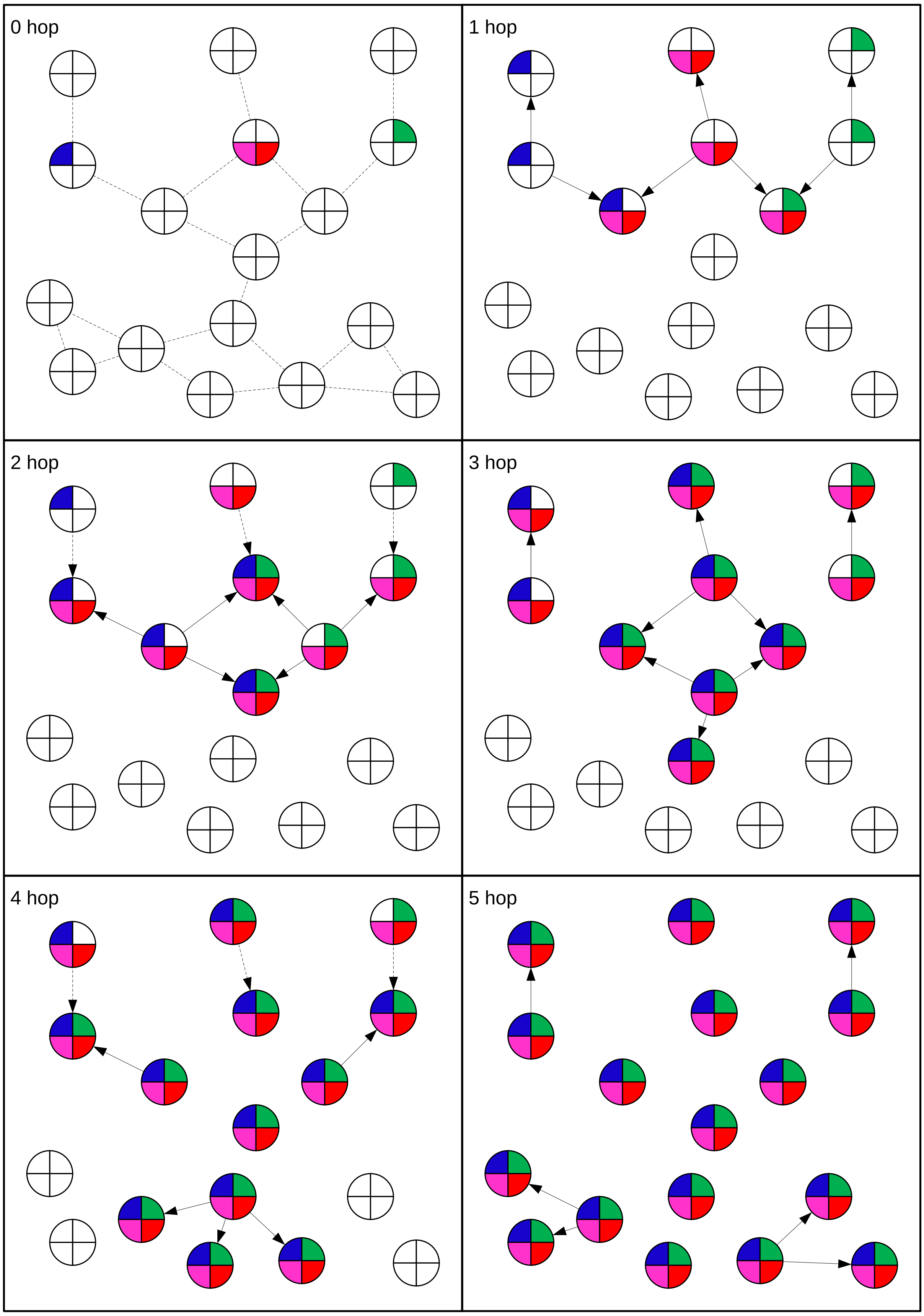


Figure 5 Message exchange with aggregation of information

In Figure 5 the number of transmitted messages is 22. The equivalent method of not aggregating the information would produce 51 messages[2] or 68 messages[3] depending on whether information in the “red & pink” circle in “0 hop” is treated as one message or as two distinct messages.

By transmitting 22 instead of 51 or 68 messages the transmission load can be reduced by between 57% and 68% depending on the handling of the information from the “red & pink” circle.

In Figure 5 the received information is retransmitted within the next hop. If the retransmission is delayed, more information can be aggregated thus further reducing the number of sent messages.

2.6.2. Starting the information exchange

In a floating network of vehicles there is no Master, there is no one to take control and make decisions.

A “Round-robin” scheduling [31] would provide a fair distribution of the responsibility of starting the update. To be able to use “Round-robin” as a selection criteria some extensive knowledge, about which vehicle is present and available for the task, is needed. As vehicles move and become unavailable this is not a solution that provides the necessary robustness and flexibility.

Random selection of the starting point is possible, but this also demands knowledge of the available vehicles. “Round-robin” or random selection does not do much good in reducing the load on the system. When it comes to initiating the spreading of information in the network an educated choice of starting point can provide savings on transmission load.

The ideal starting point would always be the vehicle that has the most complete and correct understanding of the status in the network. There is no reason to limit the number of starting points to just one; several starting points might reduce the time needed to propagate the

[2] 17 vehicles x 3 different messages = 51 messages
[3] 17 vehicles x 4 different messages = 68 messages

information through the network. Several starting points might also provide a more reliant, robust and flexible solution.

2.6.3. Neighbour information exchange

Since vehicles move, there is a need to inform new vehicles about conditions that are present ahead, particularly if “different levels of detail” is implemented. A vehicle that moves towards a cell, but only has a rough estimate about what the cell looks like might be unprepared for what lies ahead. This is an argument for frequent updates of the information within the vehicles, and throughout the network. However exchanging all the information between all the vehicles in the network is a demanding task due to the large number of messages that is needed to do this. With that in mind it might not be advisable to do this too often.

If the time between each complete exchange of information is increased, there needs to be an alternative method of giving new vehicles necessary information without needing to update the whole network.

One method of doing this would be the use of a “Hello message” that indicates the presence of a new vehicle moving towards the cell. This message format is illustrated in Figure 6. Such a message does not need to travel beyond the nearest neighbour, and could contain information about direction of movement, so that only vehicles in front of the sender would respond to the message.

Timestamp	Sender_ID	Position	“Hello”	Direction

Figure 6 Message format for a “Hello message”

“Timestamp” would provide a unique message identifier within the vehicle, which combined with the “Sender_ID”, would uniquely identify the message within the whole network. The “Position” field would give the current position, and the “Direction” field defines the direction of movement of the enquiring vehicle.

A vehicle that receives a relevant “Hello message” could reply by sending its own interpretation of the network status. There should never be a need for a vehicle to respond to more than one “Hello message” from the same vehicle within a certain window of time.

By introducing a “Hello message” it is possible to increase the time between each complete update of information for all the vehicles, and by that reduce the amount of data-traffic generated.

3. IMPLEMENTATION

This chapter builds on the theoretical foundation lain down in chapter 2, and investigates possible solutions that would perform in accordance with the theory. This chapter presents methods and mathematical equations to determine the border around a cell. The chapter also includes suggestions to how the information exchange can be conducted.

3.1. Different types of information

As mentioned in chapter 2 the information available to the network might come from very different sources, and contain different types of data. The main property that distinguishes them from each other is the amount of data to transmit. Warning messages, weather updates and driving conditions can be compressed quite extensively, while data related to advertisement and entertainment generally contain much more data, and with these types of data it is generally less to be gained from compressing the content.

One of the areas where the mechanisms from this thesis can be utilized is within improving road safety and efficiency. The messages mostly used to achieve this are warnings and updated traffic information. These messages are small and can be exchanged rapidly. Advertisements and entertainment data are not part of the system as these are normally linked to centralized servers, and are therefore unobtainable to the network. Although such large messages have been disregarded in this project, the results that are found should apply to these as well.

In transferring large quantities of data in a V2V network, splitting data into much smaller messages ensures that less capacity is wasted by the loss of a single message. There is also a need for an application to request retransmission of lost or damaged messages.

3.2. Initial gathering of information

The information that is available throughout the network can be gathered in different ways. These different ways can be separated into two categories: individual and collaborative gathering of information.

Individual gathering is the easiest and least demanding method. With individual gathering each vehicle uses the onboard sensors to gather as much information as possible about the surroundings of that particular vehicle. This information is then included the next full exchange of information within the network

The alternative is collaborative gathering where the information is gathered by using algorithms and methods for collecting information from other vehicles. The vehicles will still gather as much information as possible about their environment, but this information is shared within a cluster of vehicles to form a more solid foundation before the information is exchanged within the whole network.

This report will only consider one such gathering method called “Intelligent flooding” [32]. This algorithm has since been renamed “Single ripple” [33].

The results found regarding the performance of “Single ripple” [33] indicate that this is an effective method to discover the borders of one or more conditions. One of the benefits of the algorithm is that it does not involve more vehicles than what is strictly necessary. All vehicles[4] within a cell, and any vehicles[5] within one hop of the border, will receive information about the border of the condition. The fact that the information resides in the vehicles just outside the cell is beneficial to the expedient update of a complete network as explained in chapter 3.4.2.

[4] Green circles in Figure 7
[5] Red circles in Figure 7

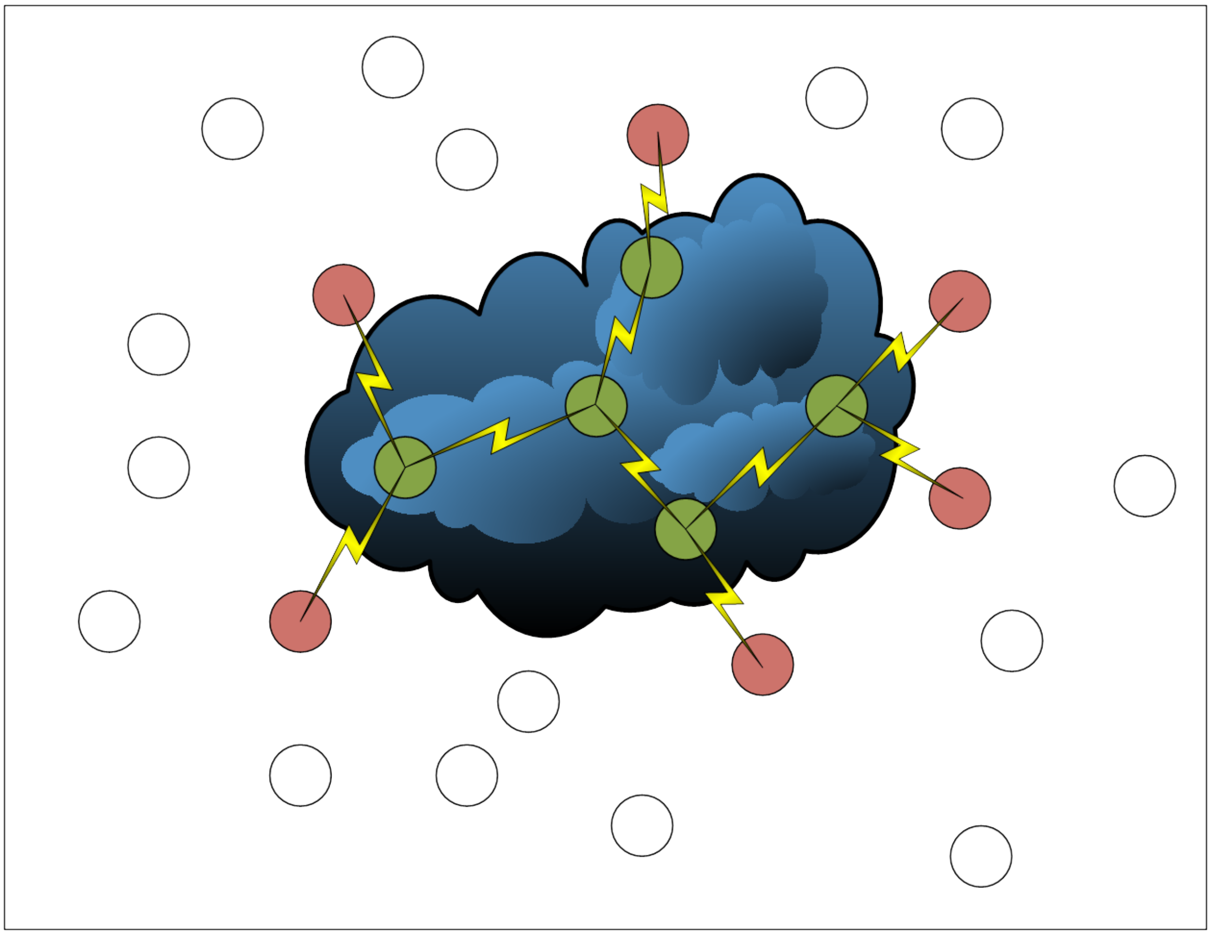


Figure 7 Result of "Single ripple"

The "Single ripple" method reduces the number of vehicles involved in the communication as any vehicle that does not possess the condition in question will refrain from transmitting to vehicles further out and only reply negatively to the original message. As can be seen from Figure 7 only the vehicles inside the cell, and the vehicles just outside the cell are involved in the communication, and these vehicles will have gathered all the available information related to the cell when "Single ripple" is finished.

To this thesis it is not important how the information was gathered, but a completed "Single ripple" is a suitable starting point for further investigations about how information can be exchanged trough the network.

Each vehicle should at any given time have information about their internal and external status, and any vehicle involved in "Single ripple" should contain an updated table of information regarding their surroundings when "Single ripple" finishes.

Table 1 illustrates how such information can be organized in each vehicle. This table would be updated and filled with entries as the information is exchanged between the vehicles.

Internal				External		
Source	**Description**	**Value**		**Entry**	**Description**	**Value**
Sensor 1	ABS	On		Cell A	Fog	{a,b,c,d,e,f}
Sensor 2	Temperature	16˚C		Cell B	Ice	{d,h,l}
Sensor 3	Fog lights	On		Cell C	Ice	{m}
Sensor 4	Traction control	Off		Cell D	Snow	{{g,h,l},{m,n,o}}
Sensor 5	Speedometer	90km/h		⋮	⋮	⋮
⋮	⋮	⋮		⋮	⋮	⋮

Table 1 Vehicle status information

Table 1 is separated into internal and external information. The internal represents the various sensors that are available to the vehicle, and the external represents the information gathered through collaboration with other vehicles. In the external part of the table the values could be sets of vehicle positions or a collection of subsets representing several cells or single points.

3.3. *Constructing cells*

3.3.1. Determining the border elements

The gathered information must be processed in order to be useful. The construction of cells reduces the amount of data needed to be exchanged. The interior of the cell could be considered homogenous, and does not provide any new information. The border of the cell is the only information that needs to be exchanged.

The process of defining cell borders begins with a list of all the positions reporting the condition in question. In this report positions are defined by XY coordinates but other positioning systems such as GPS would work using the same methods presented here.

From a list of positions it is clear that the entry with the smallest and largest X-coordinates would need to be included in the border. In Figure 8 this first pair is named “min_0” and “max_0” and is connected by a pink line.

The entries are sorted by their position relative to the pink line[6]. The following examples focus on the entries in the upper half (above the pink line). This is just a simplification to clarify the illustrations; the following methods would behave in the same manner when working with the entries in the lower half (entries below the pink line).

In the “upper-half” the process starts again: find the largest and smallest X-coordinate, and discard any entries below the line between these points. The new points are named “min_1” and “max_1” and are connected by a blue line in Figure 8. How to determine whether the entries are in the “upper-” or “lower-half” is described further in chapter 3.3.2. This process will repeat until there are no more entries left in the “upper-half”. If there is just one entry left this is added as a border element as well. An example of this would be “min_5” in Figure 8.

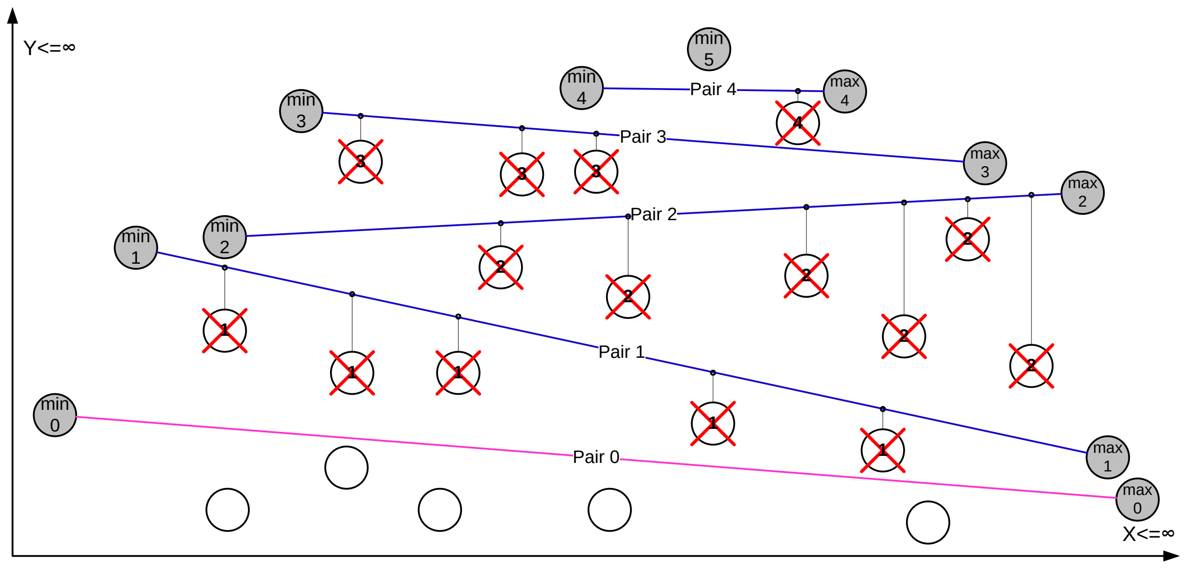


Figure 8 Determining the border by using smallest and largest X-coordinate

In Figure 8 the white circles with red crosses represents positions in the “upper-half” that are not part of the border. The number inside each of these circles indicates which min/max-pair that defined their rejection. Any position found between two lines is not a min- or max-point, and will be discarded as not part of the border. The operations in the lower-half are identical, just mirrored around the pink line between “min_0” and “max_0”.

When all the min and max points are found they can be combined to make up the border. The border is constructed by adding the upper min-points to a new list in an ascending order, and

[6] Lists with “Upper-half” and ”Lower-half”

then the upper max-points are added in a descending order. The result is a clockwise ordered list of all the border elements in the “upper-half” of the available entries. The complete list of border elements is produced by merging the lists of border elements from the “upper-half” and the equivalent list of “lower-half” border elements.

The actions that can be taken to make the shape of the cell convex instead of concave are explained in chapter 3.3.3. The complete procedure for determining the cell border is described in chapter 3.3.4.

3.3.2. Discarding non min/max points

To determine the relative position of a point to the min- and max-points it is not sufficient to just compare the Y-coordinates of the entries. As the line between min- and max-points can be at any angle from -90° to +90° relative to a hypothetical horizontal X-axis originating in the min-point, an entry that has a smaller Y than one of the min- and max-points might very well be on the upper side of the relevant line and should not be discarded as it could be a min- or max-point further out in the cell.

By constructing a linear equation for the line between the min- and max-point it is possible to determine what the relative Y-coordinate on the line would be at the same X-coordinate as the point we want to determine the relative location of. Equation 3 [34] uses the known coordinates of the min- and max-points, and the X-coordinate of the point in question to make a new Y-coordinate that can be compared to the Y-coordinate for the point in question.

$$y' = f(x) = \frac{(y2 - y1)}{(x2 - x1)}(x - x1) + y1$$

Equation 3 Y-coordinate on a straight line

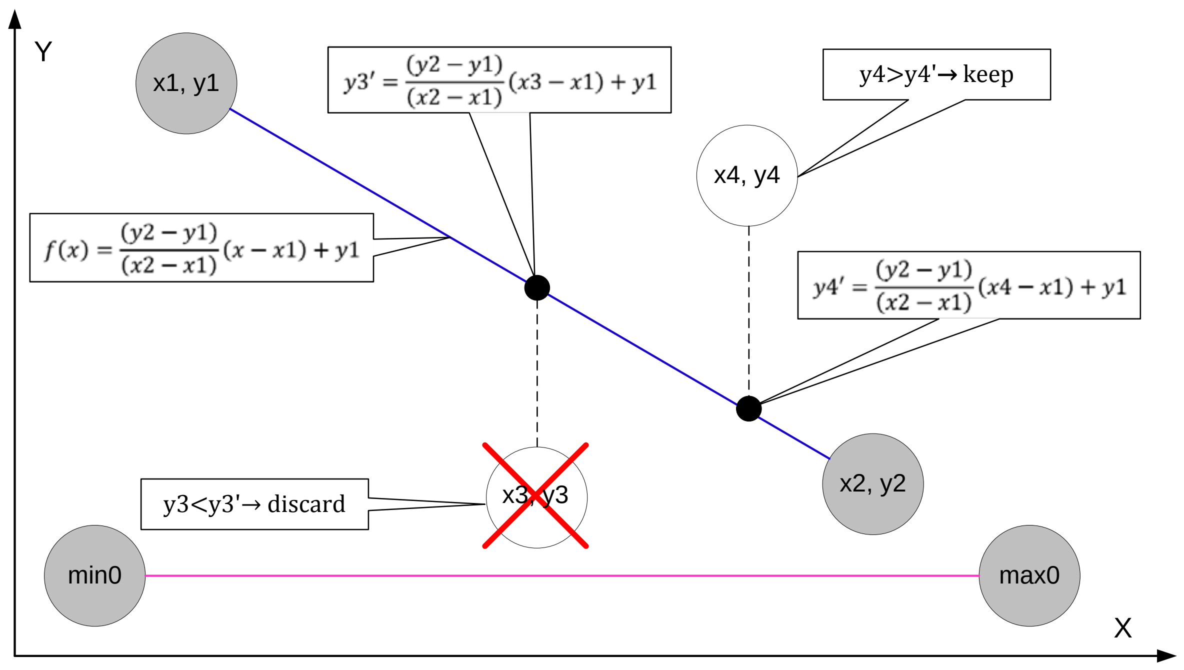

Y
x1, y1
$y3' = \frac{(y2-y1)}{(x2-x1)}(x3-x1)+y1$
y4>y4'→ keep
x4, y4
$f(x) = \frac{(y2-y1)}{(x2-x1)}(x-x1)+y1$
$y4' = \frac{(y2-y1)}{(x2-x1)}(x4-x1)+y1$
y3<y3'→ discard
x3, y3
x2, y2
min0
max0
X

Figure 9 Determining position and discarding non-border elements

Figure 9 illustrates how Equation 3 can be used to determine the relative positions of the entries when the entries are split into upper or lower halves relative to the min/max-pair in question. Figure 9 illustrates the definition of the upper half border, thus #3 is removed while #4 is kept.

3.3.3. Removing concave elements

When the list of border elements is compiled, based on the method explained in 3.3.1, there might be remaining elements that alter the shape of the cell in a significant way. As discussed in 2.4.1 this might not be desirable.

There are at least two possible methods of determining whether an element contributes to make a concave or convex cell shape. One method is calculating the angle between neighbouring points in the ordered list of border points. This is illustrated in Figure 2. An angle >180° indicates concavity and an angle of <180° indicate convexity. This would involve the “Law of cosines” [35] as quoted in Equation 4.

$$\beta = \cos^{-1}\left(\frac{a^2 + c^2 - b^2}{2ac}\right)$$

Equation 4 “The law of cosines”

The variables in Equation 4 are illustrated by Figure 10. Circle B represents the point that is going to be checked for concavity. The circles A and C represent the neighbours on either side of B in the list of border elements.

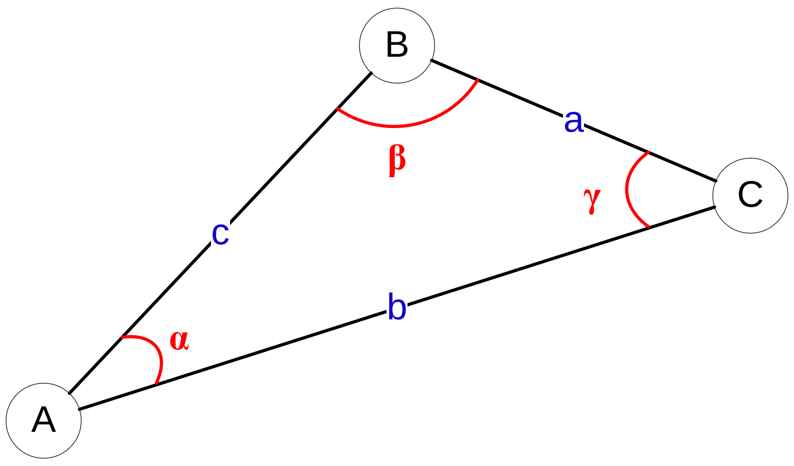


Figure 10 Parameters for “The law of cosines”

There are some problems related to this approach as there is no fixed point to compare the angles against, this means that the angle will always be calculated at the inside of a triangle, and therefore always will be less than the 180° that we are looking for. It is possible to use a fixed point as basis, and then use the trigonometric functions with this as a reference. However this would be unnecessarily complex as there is a simpler method.

Another way of discovering concave elements is by using the method explained in 3.3.2. By calculating the relative position of the element it is possible to determine if the element is a source of concavity or not. Comparing the Y-value of a border node with its corresponding Y-value (Y’) on the line between its neighbouring nodes any node with Y>Y’ in the “upper-half” is kept and the node is discarded if it is in the “lower-half”. If Y<Y’ for a node in the “upper-half” the node is discarded, whilst a node in the “lower-half” would be kept. Figure 11 illustrates how Equation 3 can be used to determine the presence of concave elements.

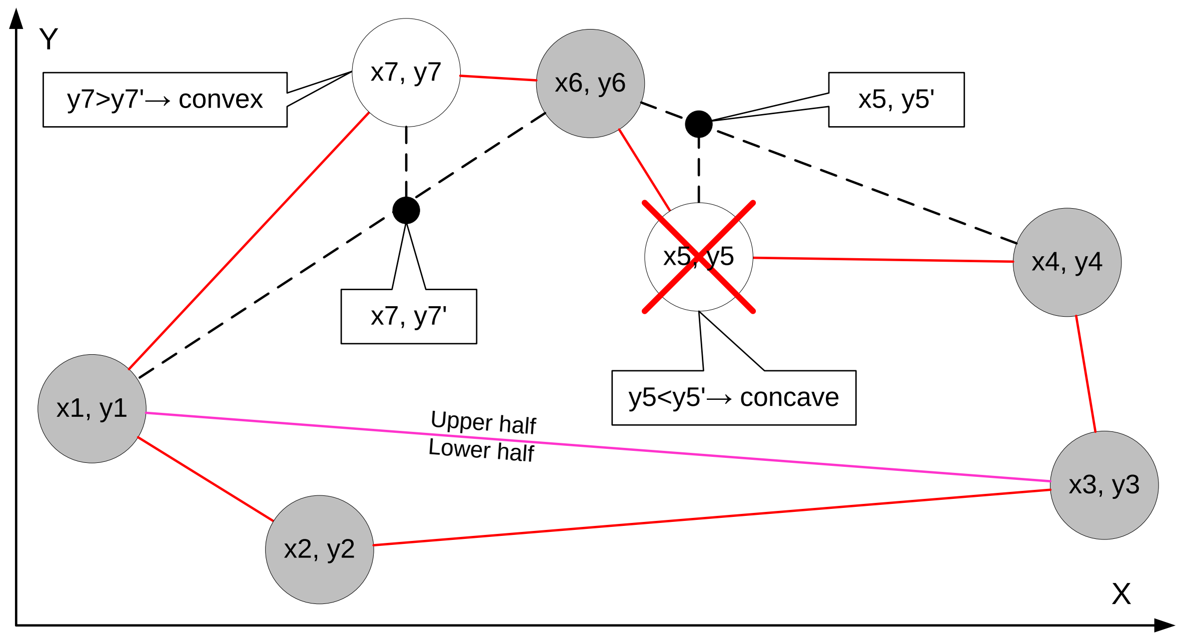


Figure 11 Determining, and removing, concave elements

In Figure 11 #5 is inside the line between #4 and #6 so this element has to be removed to make the cell convex. #7 should not be removed, since it is outside the line between #6 and #1 and contributes in a positive manner to the convex shape of the cell.

Based on these results Equation 3 seems a simpler method to determine concave elements than the “the law of cosines” (Equation 4).

3.3.4. Procedure for constructing the cell border

When the preceding chapters are combined it is possible to produce a flowchart, such as Figure 12, that describes the actions needed to make a suitable list of border-elements.

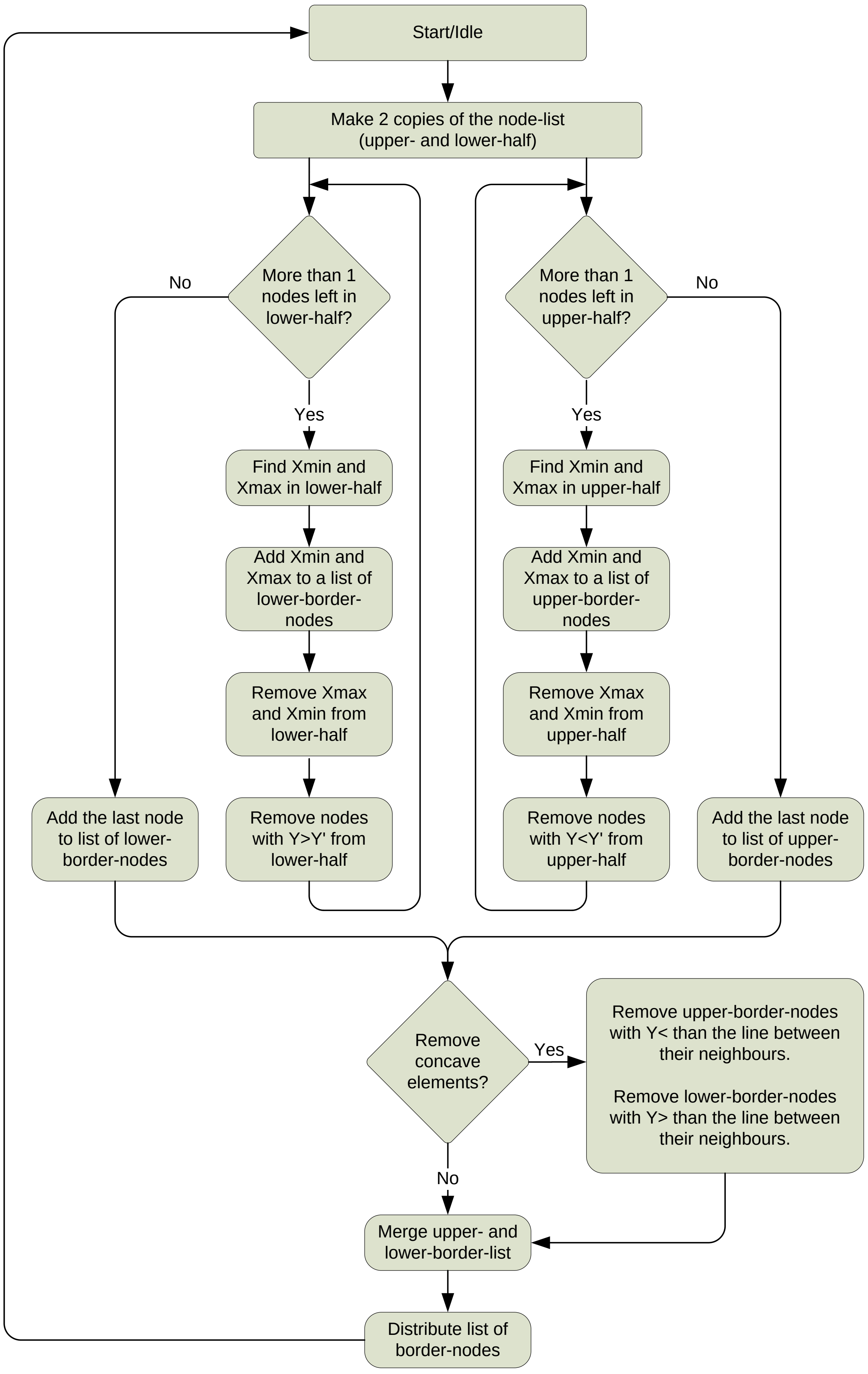

Start/Idle
Make 2 copies of the node-list (upper- and lower-half)
More than 1 nodes left in lower-half?
No
Yes
More than 1 nodes left in upper-half?
No
Yes
Find Xmin and Xmax in lower-half
Add Xmin and Xmax to a list of lower-border-nodes
Remove Xmax and Xmin from lower-half
Remove nodes with Y>Y' from lower-half
Add the last node to list of lower-border-nodes
Find Xmin and Xmax in upper-half
Add Xmin and Xmax to a list of upper-border-nodes
Remove Xmax and Xmin from upper-half
Remove nodes with Y<Y' from upper-half
Add the last node to list of upper-border-nodes
Remove concave elements?
Yes
Remove upper-border-nodes with Y< than the line between their neighbours.
Remove lower-border-nodes with Y> than the line between their neighbours.
No
Merge upper- and lower-border-list
Distribute list of border-nodes

Figure 12 Flow chart for determining the border elements

By following Figure 12 it should be possible to take any number of elements and produce a border around them, including removal of concave elements. The results when presented in a XY-coordinate system would look like Figure 13.

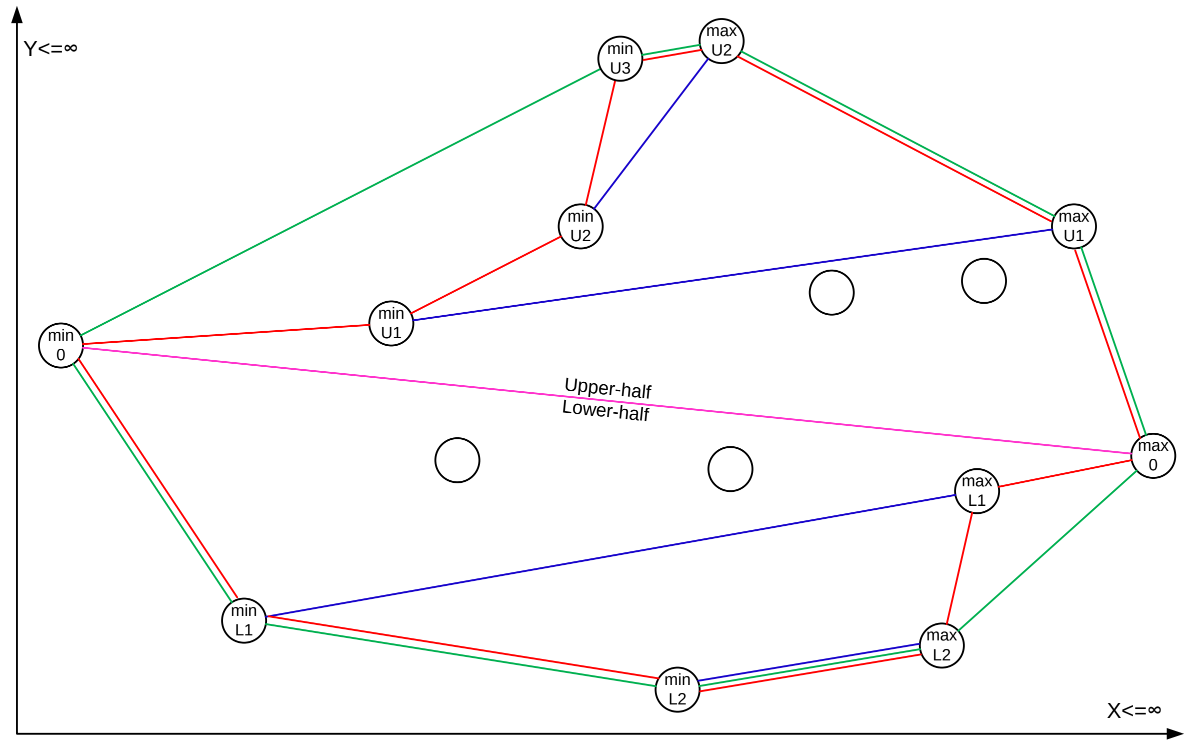


Figure 13 Result from determining the cell border

In Figure 13:

- The green line is the border without concave elements.
- The red line is the border with the concave elements.
- The blue and pink lines are intermediate lines between pairs of min- and max-points.

3.4. Exchange

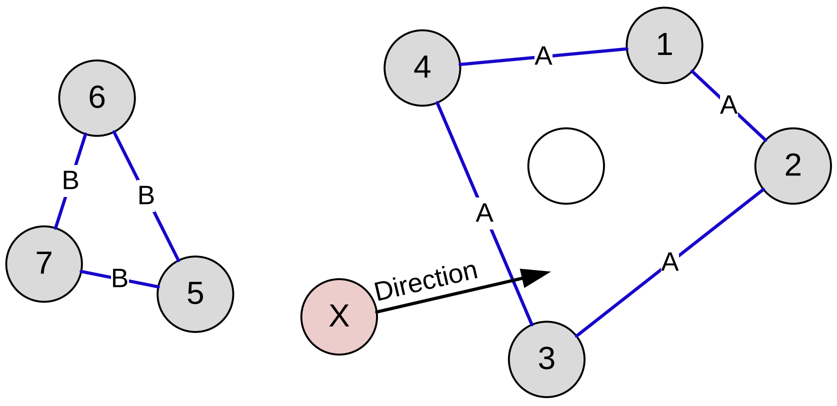
To form a unified understanding of the status in the network the information needs to be exchanged as efficiently as possible. This thesis proposes two methods. The first, and least expensive, solution would be that the vehicles moving around would request information. The solution of complete exchange of all the information available would be much more expensive, but would at the same time make the network status unified.

A combination of these two solutions would give the desired flexibility and reliability in the information exchange. A full exchange can be conducted at large intervals, while the less demanding “Neighbour update” could be used at much smaller intervals.

The following chapters are not limited to cases where “Single ripple” has been used to collect the information, but “Single ripple” is included in the illustrations as a source of information, and a starting point. The methods presented can be used regardless of how information was gathered, and what kind of information that is being exchanged.

3.4.1. Neighbourhood update

Spreading information to neighbours is a useful tool to avoid involving all the vehicles in the network. Only the nearest neighbours would be involved, keeping the transmission load at a minimum. A “Hello” message would trigger a response-message containing the available information in front. By including a vector describing the direction of movement it is possible to reduce the number of replies, ensuring that only the vehicles that are in front of the initiator originate a response, as these are the vehicles that are most likely to have new and relevant information. This is illustrated in Figure 14.



X->All: Hello, “Direction”
3->X: Condition A: 1(pos),2(pos),3(pos),4(pos)

Figure 14 Neighbour update based on direction

In Figure 14 the red circle X is a vehicle moving towards #3. Vehicle X sends out a “Hello” message to anyone within reach, this message is received by #3 and #5. Only #3 will respond as this is the only receiver that lies in the direction of the movement. This behaviour is illustrated in Figure 15.

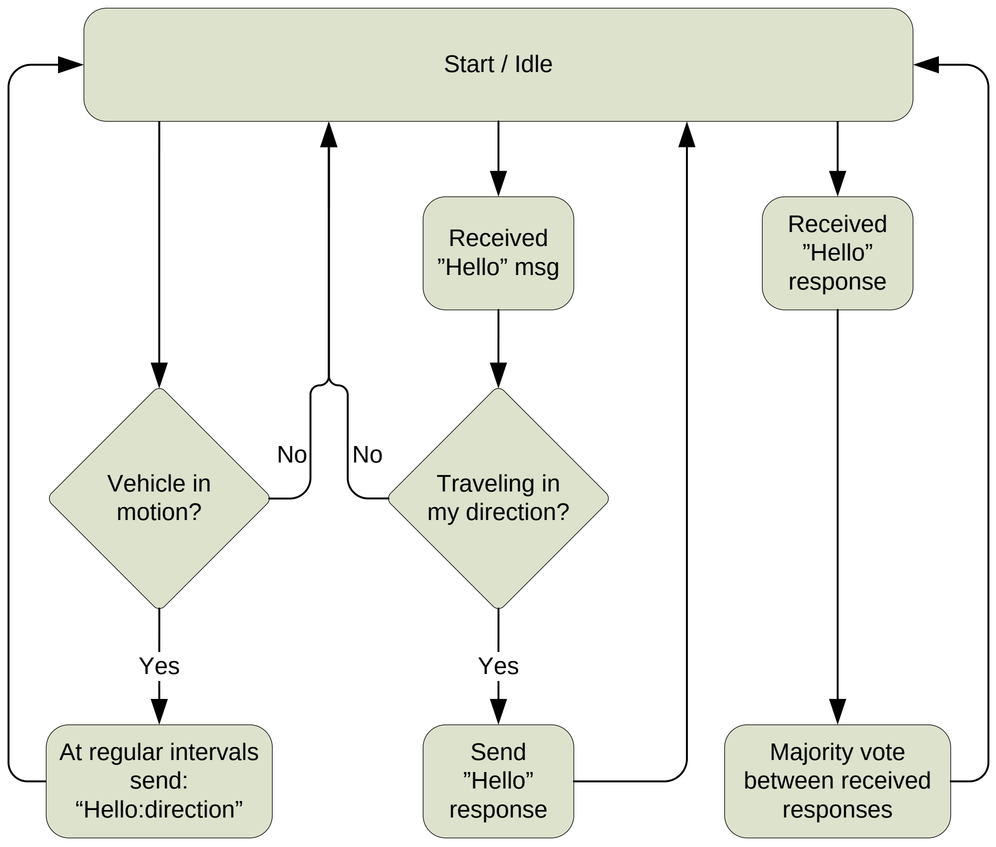

Start / Idle
Received ”Hello” msg
Received ”Hello” response
Vehicle in motion?
No
No
Traveling in my direction?
Yes
Yes
At regular intervals send: “Hello:direction”
Send ”Hello” response
Majority vote between received responses

Figure 15 Flowchart for neighbour update

The flow chart in Figure 15 has three branches.

- The first branch is related to movement and sending of the “Hello” message. If a vehicle is not in motion there is no need to send a “Hello” message.
- Branch two is the actions taken by vehicles that receive a “Hello” message. If their position is in the direction of movement they will respond with a message containing all the relevant information.
- The third branch displays the actions taken by the originator of the “Hello” message when it receives a reply.

3.4.2. Starting the update

When initiating the broadcast of information in the network an educated choice of starting point can provide savings on transmission load. If possible the vehicles with the most complete interpretation of the status in the network should start the exchange. This is normally not known and the choice has to be made based other criteria.

Using a collaborative method, like “Single ripple” to gather information about conditions leaves vehicles to constitute the border of the cell. These border vehicles will contain the same data as all the vehicles within the cell. Since all vehicles within the cell already contains the data there is no reason to transmit the information inward in the cell.

By letting the border vehicles start the update most of the vehicles inside the cell are shielded from the messages that contain the information that was gathered during the “Single ripple”. If the cells are huge, this might prove to be a relatively great saving in transmission load.

In an arbitrary system it cannot be known which vehicles hold the most information and the vehicles would have to decide by themselves to exchange the available information. Such an exchange should happen on a regular basis. The use of timers and internal clocks will suffice to regulate this exchange.

There are no limitations to how many vehicles that can initiate the exchange of information, or when they do it. However there is a benefit in reduced transmission load to be derived from several vehicles transmitting within a short period of time, as there is a potential for aggregating the information into single messages. This aggregation of information is illustrated in Figure 5.

3.4.3. Full scale exchange

As proposed in 2.6.1 there is a lot to gain by aggregating messages, as one message would substitute many single messages there is no need to transmit several messages when one would be satisfactory.

Figure 16 illustrates a flowchart of how an exchange algorithm might be constructed. The algorithm has been given the name ICE, an anagram for “Information Combined and Exchanged “.

As discussed in chapter 2.6.2 any selection scheme could be used to determine who should initiate the exchange, the wise choice would always be the vehicle with the most information to share. The flowchart in Figure 16 consists of two main branches. The right branch shows the initiation of the exchange based on the result of a “Single ripple”; however the use of

"Single ripple" is not mandatory for the functionality. The right branch describes the actions taken by receiving vehicles.

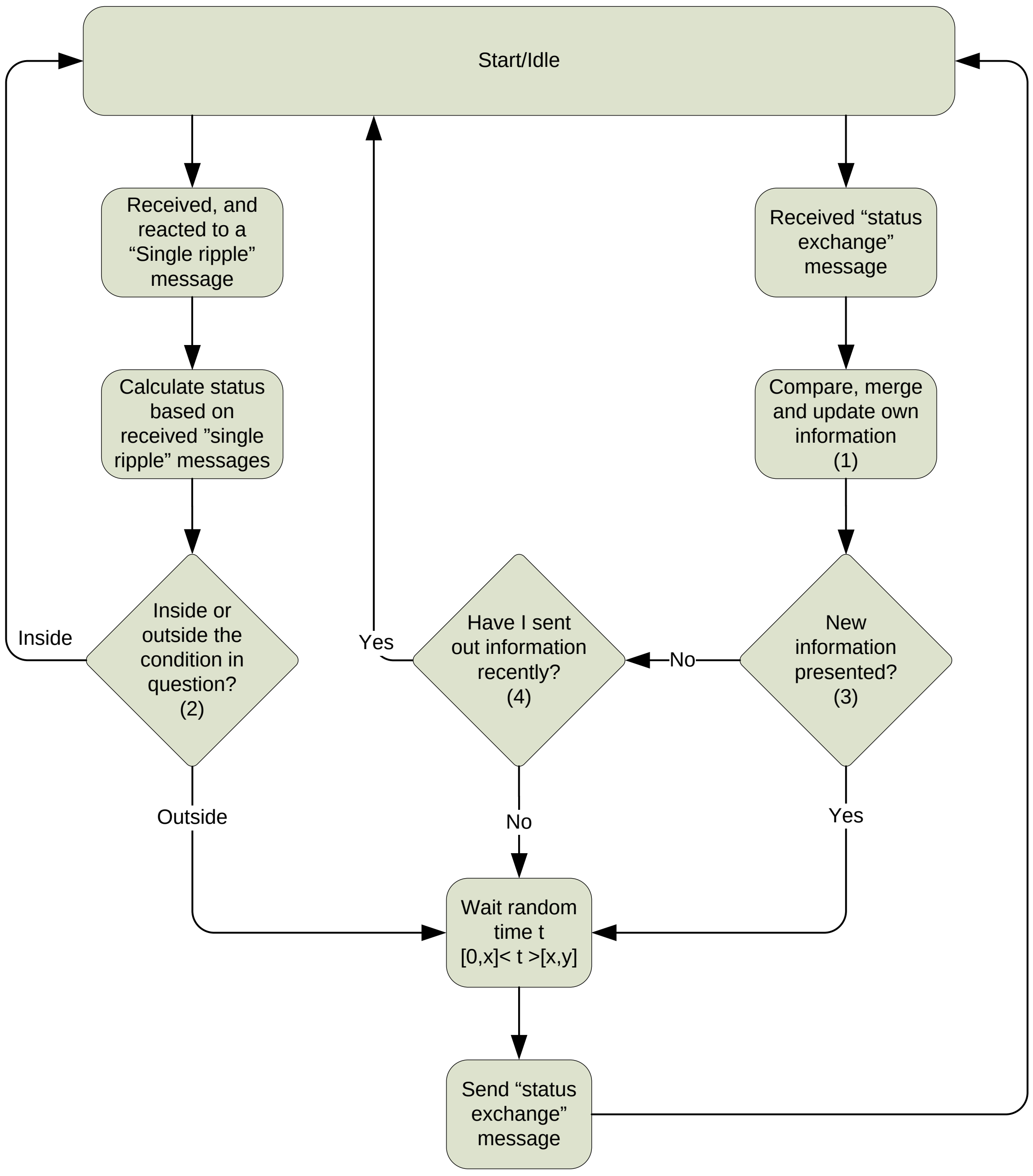

Start/Idle
Received, and reacted to a "Single ripple" message
Calculate status based on received "single ripple" messages
Inside or outside the condition in question? (2)
Inside
Outside
Yes
Have I sent out information recently? (4)
No
Received "status exchange" message
Compare, merge and update own information (1)
New information presented? (3)
No
Yes
Wait random time t [0,x]< t >[x,y]
Send "status exchange" message

Figure 16 Flowchart for the ICE algorithm

The most important parts of the algorithm in Figure 16 are described below.

- The block marked (1) is responsible for the internal handling of the received information. Merging and “different levels of detail” must be implemented in this block. This block needs to behave according to the current context and the information present.
- The choice block marked (2) is “Single ripple” specific, and ensures that vehicles on the outside of a cell start the exchange.
- The choice block marked (3) is responsible for detecting if there is any new information in the message received. If so it triggers a retransmission of the available information.
- The choice block marked (4) checks if the message contains information already received, or information the vehicle already possesses. This block ensures that the exchange is triggered in all vehicles to prevent crucial vehicles stopping a chain of updates.

A good example of such a crucial vehicle could be found in the middle of each frame in Figure 5, the vehicles are positioned in a shape that resembles the number eight and the vehicle in the middle could stop the information migrating between the two halves.
In Figure 17 the red vehicle acts as a crucial vehicle between the green and the blue vehicles. The red vehicle needs to continue the exchange even if it does not receive any new information, because there might be vehicles within the blue area that has not got the information from the green area. A timer could be used to ensure that the exchange is repeated at reasonable intervals.

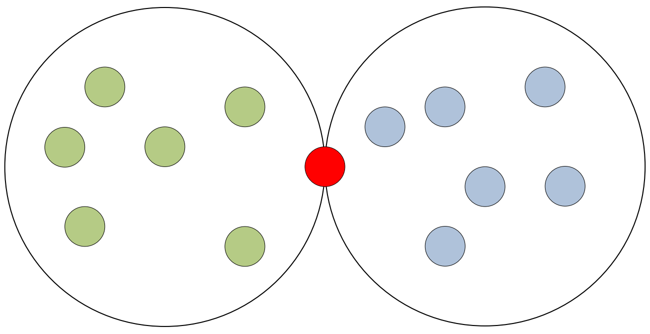


Figure 17 Representation of a network with a crucial vehicle

When the vehicle decides there is a need to send out information a waiting period can delay the transmission to, if possible, gather more information before sending. This waiting period is given a random element, to reduce the probability for several vehicles transmitting at the same time. Without this random element vehicles that receives a message simultaneously would transmit simultaneously, and possibly both messages might end up destroyed by interference.

ICE algorithm does not, unlike “pure flooding”; rely on timestamps or any form of message identification to distinguish between new and old messages. In ICE it is the content of the message that decides if the message should trigger a response or not.

Another difference between ICE and “pure flooding” would be the ability to accommodate different levels of detail. In “pure flooding” all vehicles exchanges their personal information. In ICE it is possible to implement a calculation before transmission to accommodate different levels of detail. If the vehicle that receives a message is distant from the information in question it is possible to just transmit a reduced set of the information, and not include the information that can be considered redundant.

The information needs to be received, processed and forwarded according to the rules of the information and the context involved. This includes different levels of detail and timeout periods. By doing this it would be possible to reduce the transmission load compared to “pure flooding”.

3.5. Different levels of detail

The use of different levels of detail could be beneficial to the load on the system. By removing information that is not strictly necessary would help keep the message sizes down. To implement this, a set of rules would be needed according to each type of information and context.

The rules should be based on distance from the original source of information. The further away, the less information should be kept. To make this work the rules would have to be identical within the different vehicles, to ensure correct handling and response.

When using distance from the information as a parameter for selecting the amount of detail, it is possible to expand the complexity by including speed as a criterion. Distance and speed combined would give the estimated time of arrival at the point of interest. This would make it possible for vehicles to adapt their information burden according to when the information will be necessary. A vehicle travelling at 90 km/h would need more details sooner than a vehicle that only travels at 30 km/h.

Including speed as a variable is challenging as this could vary greatly from vehicle to vehicle and also for any vehicle from time to time. Averaging the speed over a period of time reduces this problem, as long as it can be assumed that that the vehicles have the same average speed at a given section of road.

The calculations and adaptations to the context would be done at each vehicle individually, but to avoid problems all the vehicles would have to use the same set of rules in the same situation. A counter representing the different levels of detail can be used to keep track of what kind of information that should be presented to a vehicle. By doing this a receiving vehicle could know what kind of information that might have been left out of the message by the source.

The neighbour update, as described in chapter 3.4.1, would complement the use of different levels of detail by providing information as the vehicles move forward. In this way the amount of detail will increase as the vehicle move towards an information source, and be reduced when the vehicle move away from an information source.

If the information exchanged is related to multiple cells they can be merged to reveal the different levels of detail as displayed in Figure 4. This can be done by using the method described in chapter 3.3, and especially in Figure 12, on all the information points in the cells eligible for merging.

4. TESTS & SIMULATION

This chapter is devoted to testing and simulation of some of the solutions proposed in chapter 3. This chapter is split in two parts. The first part looks at the construction of cells described in chapter 3.3. The second part explains the information exchange described in 3.4, and specifically the ICE method presented in 3.4.3.

4.1. Constructing the cells

Hypothesis:

It is possible to define a convex border that surrounds all entries within a list of positions.

This hypothesis can be tested by making a program that visualizes the placements of random positions, and then follow the method explained by Figure 12.

4.1.1. Installation and programming Java

I have used a standard Java JDK v6.18 [36] installation and NetBeans version 6.8 [37] on a computer using Windows XP SP3.

The program uses “Graphics2D” functions to draw the positions/elements as circles, and connect them with the appropriate lines. The circles represent the location specific information that is available within a cell. The locations are stored as entries in a list.

The circles are placed at random within the frame of the screen. Each circle contains XY-coordinates representing their centre. The different lines between the circles are drawn with a small offset to make them more visible in case they overlap. Most of the programming is done using the Java manual [38], a graphic tutorial [39] and web searches.

The functionality of the Java program is as close to Figure 12 as possible, and should not differ in any significant way.

The code is presented in APPENDIX B.

4.1.2. Variables

The positioning of the elements is completely random within the frame of the area. To make the result more visual the frame (700px X 700px) is a little bigger than the actual area (650px X 650px) occupied by the elements.

The number of elements is set to be 20. This number is chosen because it accurately represents the functionality of the method without introducing unnecessary clutter. Any number could be used, but the visual effect is reduced as the number of elements is increased. With very few elements it is difficult to display all the different scenarios that the method is handling.

4.1.3. Results from the Java program

Figure 18 shows a screenshot of the program where 20 elements are placed at random.

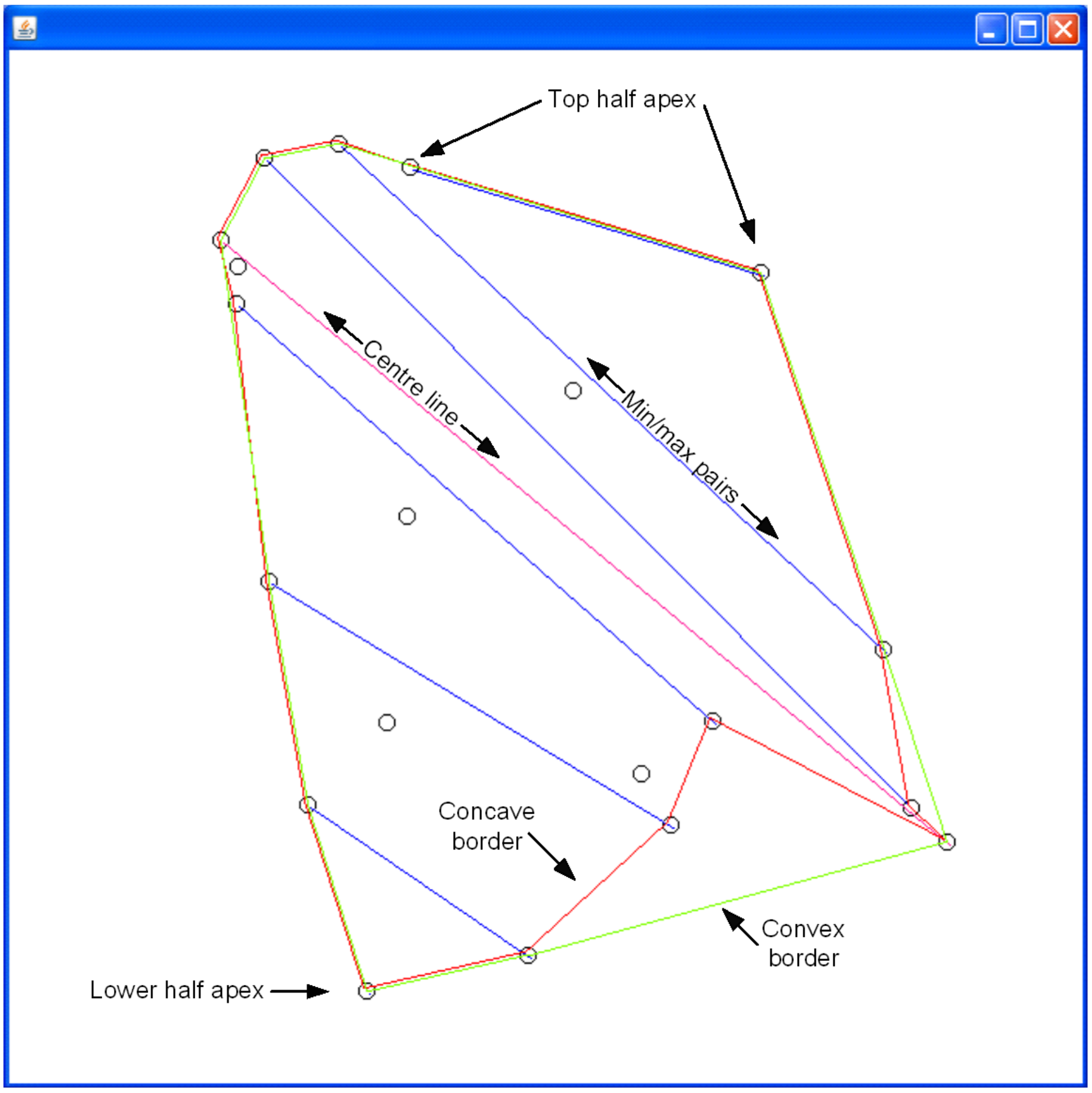


Figure 18 Screenshot of the border determining method, with 20 elements

In Figure 18, as in Figure 13, the pink line represents the centre line between the first min- and max-points. The blue line represents the connection between the following min- and max-points. The red border is the border as it is constructed with concave elements, and the green border is the same border with the concave elements removed[7].

Figure 18 shows several pairs of min- and max-points, what the border looks like when it includes concave elements and the result when the border has been made convex. The apex in the top half of the border is formed by a pair of elements, whilst in the lower half a single element is the apex.

[7] And by that the border is made convex

Empirical studies reveal that the method described in Figure 12 works excellent on any number of elements. Given a list of positions the method will return a list presenting the border-elements in a clockwise orientation[8].

4.2. Simulation of ICE

Hypothesis:

ICE will exchange information across the network with significantly reduced transmission load compared to "pure flooding".

This hypothesis can be tested using a simulator that emulates wireless equipment, movement and interactions between vehicles.

4.2.1. Installation and programming NS2

The platform for the simulation was similar to the one used to develop and simulate "Single ripple" [33]. The simulations were done through a Cygwin [40] installation on a computer using Windows XP SP3. The environment for the simulation was a standard NS2 [41] installation, version 2.29-Allinone. The alterations related to the newer releases of NS2 should not have any impact on the results of the simulations. The installation was done according to instructions that can be found on the internet [42] [43].

This project was forced to use an older version of Cygwin, as the newest versions do not presently support NS2. To install an older edition of Cygwin with Python it is necessary to follow the instructions given by "Smallko" [42] and then install Python into Cygwin by using "Fruitbat Cygwin Legacy" [44] as the package source.

The language used in NS2 is Tcl [45]. Tcl is suitable for simulations because of its flexibility and dynamic behaviour. The code is written according to previous perfected standards [46] [47] [48]. Forums and tutorials found on the internet have been the main sources of information when facing problems.

[8] In a circular list the last and first elements are neighbours

A UNIX installation might have been better suited as platform for running the simulations. A platform based on Windows is not as efficient because Cygwin is needed to emulate the necessary UNIX-environment for the simulator.

The complete simulator code for ICE can be found in APPENDIX C.

4.2.2. Scenario and variables

One scenario could be that the information is provided by border vehicles in a cell defined through collaborative gathering such as “Single ripple”. However the simulator used in this project does not use information from border vehicles, as the aim is to generate more universal results and show that ICE can be used on different scenarios, not only those present after “Single ripple”. The simulator is programmed to act as described in Figure 16.

The simulation is based on standard 802.11 communications, as this is the most widespread technology, and a very likely candidate for the communication between the vehicles [25].

The aim of the simulations is to compare the performance of ICE, as described in Figure 16 with the performance of a standard “pure flooding”. The code for “pure flooding” can be found in APPENDIX D.

To test performances the simulator had to be set up with a large selection of different parameters and record their effect on the performance of the two methods. The scenario for the simulator is an area of 1 km x 1 km where vehicles can move around freely. Streets are not included in the simulations as this would limit the number of different available scenarios when movement and positioning of vehicles is limited to 4 directions[9]. A simulator similar to TraNS [49] might provide useful data as it incorporates some driving patterns and also maps/streets, but the benefit is reduced by the very limited availability of different scenarios. Table 2 describes the most important variables in the simulator and their values.

[9] North, south, east and west

Variable	Values	Definition
opt(X)	1000	Length of area (m)
opt(Y)	1000	Width of area (m)
optNodes	25, 50, 100, 200	Defines the number of vehicles
optActive	10, 20	Defines the number unique information's available
DefaultWait	0, 5, 10, 25, 50, 100	Max delay before sending content (ms) (Randomized)
optRounds	100	Number of complete exchanges within each simulation
opt(interval)	10	Time between each round (s)
opt(speed)	10	Max speed of the vehicles (m/s) (Randomized)

Table 2 Variable declaration and values

The simulations have been carried out with different numbers of vehicles. A wireless transmitter can be considered to have a useful range of approximately 150 meters [50] in ideal outdoor conditions. In an area of 1km x 1km there is room for 36 vehicles positioned no less than 150 m apart. See Equation 5 for mathematical details.

$$\#of\ vehicles = \left\lfloor\left(\frac{length}{range}\right)\right\rfloor * \left\lfloor\left(\frac{width}{range}\right)\right\rfloor$$

Equation 5 Maximum of non communicating vehicles

- With 25 vehicles the area is very sparsely populated.
- With 50 nodes the area is sparsely populated.
- With 100 nodes the area is densely populated.
- With 200 nodes the area is very densely populated.

These chosen numbers of vehicles[10] simulates four different scenarios. The positioning of each vehicle is totally random, so there is a possibility that vehicles might be positioned out of reach of other vehicles.

The movement of each vehicle is random and limited to a maximum speed of 10 m/s (36 km/h). This is a fair estimate based on the fact that the average speed in major cities in the UK is less than 30 km/h [51].

[10] “optNodes” in Table 2

A set of unique messages is distributed randomly between the vehicles. One vehicle might hold more than one message before the exchange starts. The simulations are conducted with 10 and 20 such messages[11].

One of the key features in ICE is the delay before the transmission. This delay makes it possible to collect more information before transmission. The duration of the delay is a random value smaller than the “DefaultWait” in Table 2.

During a single run of the simulator the ICE algorithm is executed several times[12]. This is done to ensure that any extraordinary situation is not shifting the result. Between each run of ICE there is a delay[13] and the gathered information is cleared. New information is assigned to new vehicles, and the simulation process is repeated. The movement of the vehicles is independent of the rounds, and the vehicles will move throughout the whole simulation.

4.2.3. Randomness in the simulator

Randomness is used in the simulator to test as many scenarios as possible, and to make the simulations as realistic as possible. The following is a list of the random functions, and their purpose, that are used in the code.

Positioning:

The vehicles are placed randomly within the area.

Movement

When the simulations begin a route, or range, of movements is assigned to each vehicle. These routes are calculated randomly for each vehicle before each run of the simulator using the function “setdest.exe” [52].

Transmission delay

The time a vehicle delays its transmission is a random value smaller than the “DefaultWait”. This value contributes to the gathering and aggregation of information before transmission.

[11] ”optActive” in Table 2
[12] “optRounds” in Table 2
[13] “opt(interval)” in Table 2

Information distribution

The different messages are assigned to randomly chosen vehicles. This function is strictly not necessary as the vehicles themselves are randomly placed within the area. The outcome of two random numbers acting on each other might be just the same as another random number.

There are two arguments for having random distribution of messages in the system.

1. It allows the simulator to use the same random movement file without necessarily ending up with identical results.
2. It allows the simulation to contain multiple independent executions of the exchange algorithm. Each execution of the exchange algorithm is referred to as a round. Each round acts independently of each other.

Initial delay

The start of each round begins with all the vehicles that have something to exchange preparing to send their interpretation of the system status. A small random delay is inserted here to ensure that all the vehicles do not transmit at the same time, as this would be unrealistic, and might cause congestion or message loss.

4.2.4. Selection criteria for the number of rounds

Within each simulation the ICE algorithm is run repeatedly. The results of each round is gathered, and averaged on all the vehicles in the simulation. According to the "Law of large numbers" [53] [54] the average of several independent random numbers will trend toward the expected value[14] of the random numbers. This will have an impact on the result of the simulations as extreme values will be cancelled out and the result will be more stable as the number of rounds increase.

Empirical tests and analysis of this can be found in APPENDIX A. Based upon the results of the empirical tests it seems that 100 rounds should be enough to get a stable result. This is confirmed by the "Monte Carlo method" [55] which states that multiple values will average towards a stable value.

[14] Expected value = E(X)

4.2.5. Comparable parameters

To be able to compare the different results some of the variables would have to be equal, to avoid situations that would shift the results in one direction or the other.

The movement pattern for the vehicles is kept identical for both ICE and "pure flooding", for each of the different number[15] of vehicles. This is done to ensure identical working conditions, and comparable results between the methods. The movement pattern is also identical between the different delays[16] to show the effect the different delays have on the system.

In Figure 19 the blue blocks display which simulator configuration that has uses the same movement pattern.

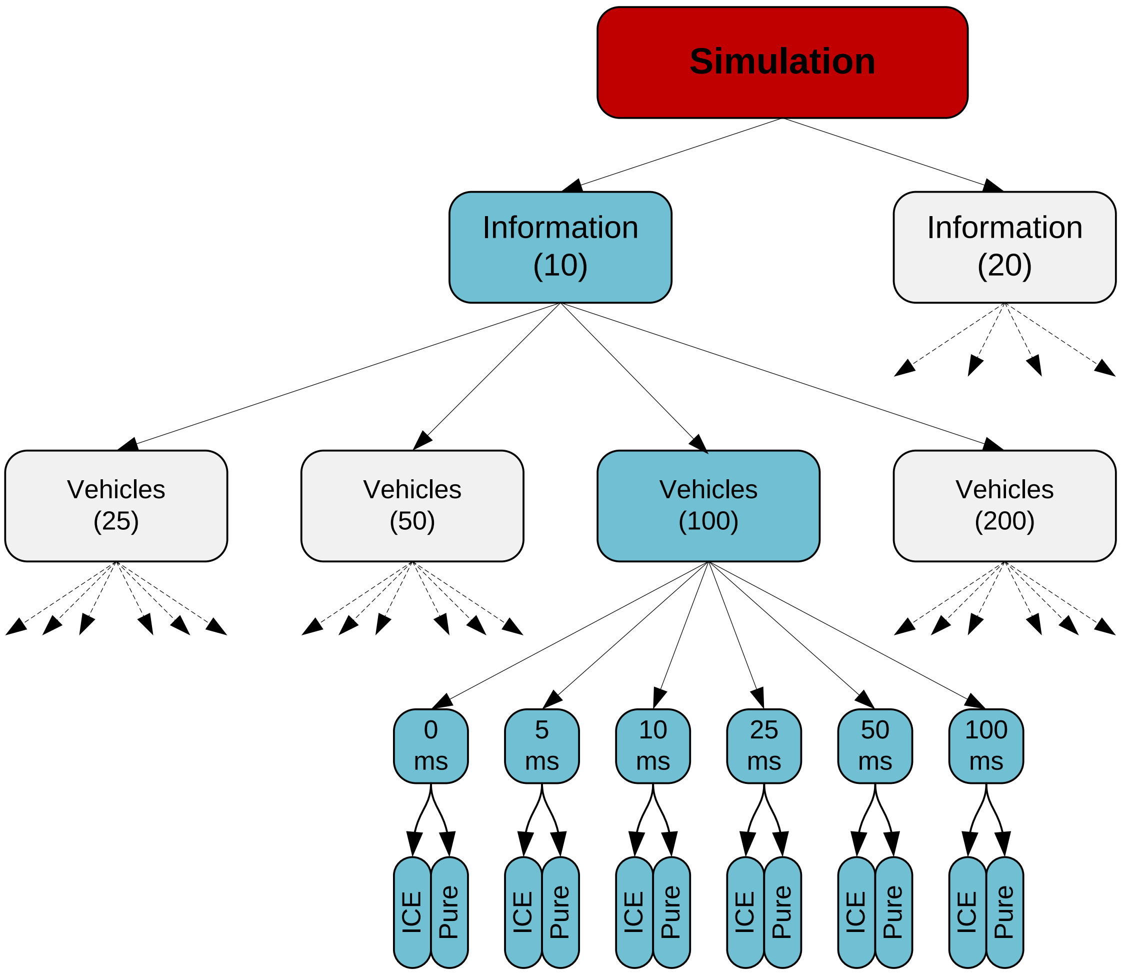


Figure 19 Simulation parameter layout

[15] 25, 50, 100, 200 vehicles
[16] 0ms, 5ms, 10ms, 25ms, 50ms, 100ms

ICE and "pure flooding" should perform concurrent regardless of the number of unique information[17] available. By using different movement patterns for the two different message numbers the simulations could provide information whether the methods are dependent, or influenced, by the randomly generated movements.

4.2.6. Simulation results

The results gathered from the simulations are displayed in Table 3.

The table is divided into vertical blocks based on the various combinations of; method used for information exchange[18], number of nodes present[19], and initial information available[20]. There are six horizontal blocks displaying information regarding each of the different delays[21] that were used.

- "Sent pr node/round" is the average number of messages sent by each node in each round. This describes transmission load, and should be as low as possible.
- "Time pr node/round" describes the average time between first and last message received at each node in each round. This should be as low as possible.
- "Result pr node/round" describes how much of the initial information available that on average was received at each node in each round. This should be as high and close to the "Unique messages" as possible.
- "Received pr node/round" is the average number of messages received at each node in each round. This is related to the numbers of messages "Sent pr node/round", "Dropped pr node/round" and the node density.
- "Dropped pr node/round" is the average number of messages lost due to interference at each node in each round. A smaller number describes fewer collisions between messages in the network.

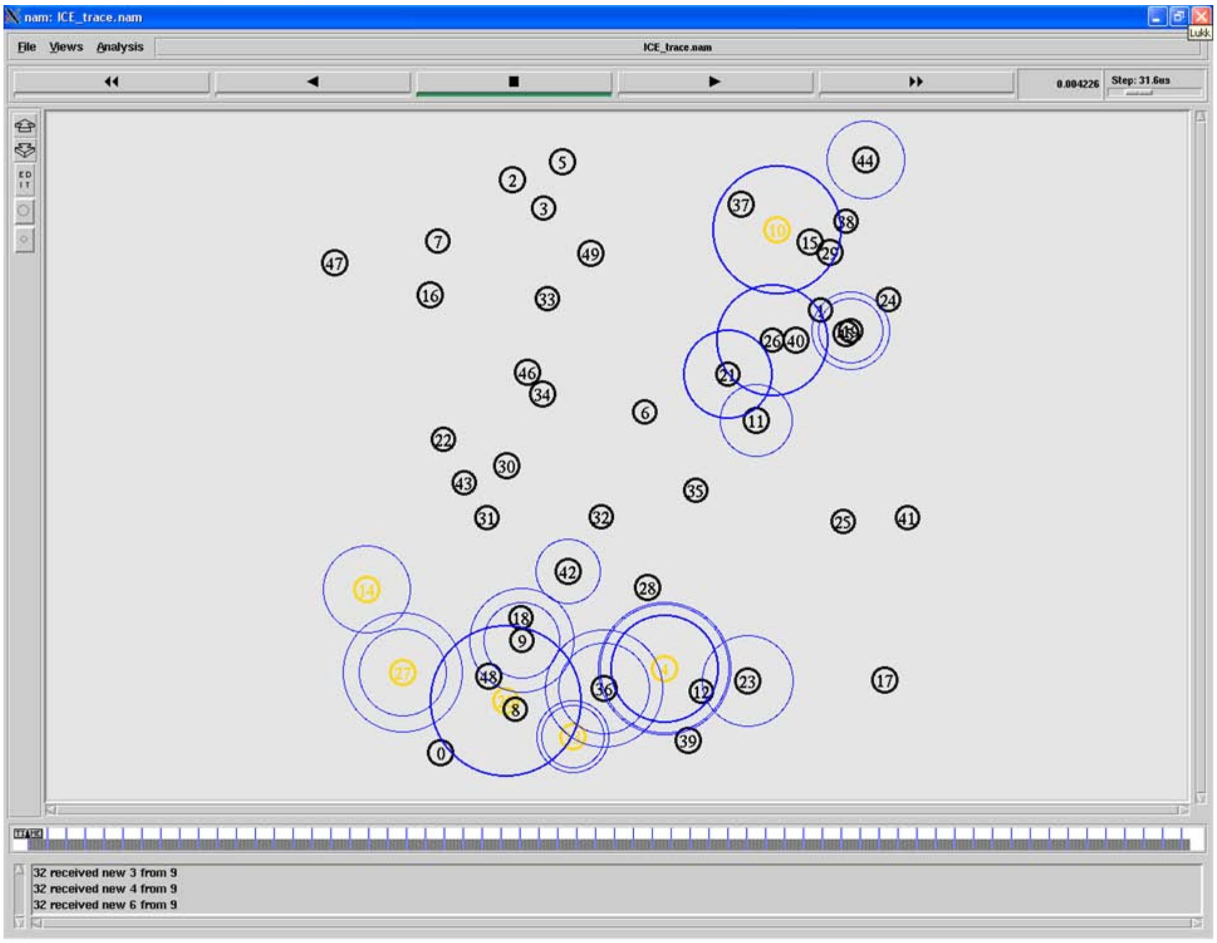
[17] 10 and 20 unique messages injected into the system
[18] ICE or "pure flooding"
[19] 25, 50, 100 or 200 vehicles
[20] 10 or 20 initial information
[21] 0, 5, 10, 25, 50 and 100ms

		ICE		Pure flooding		ICE		Pure flooding		ICE		Pure flooding		ICE		Pure flooding	
	Nodes (vehicles)	25		25		50		50		100		100		200		200	
	Unique messages	10	20	10	20	10	20	10	20	10	20	10	20	10	20	10	20
Delay 0ms	Sent pr node/round	4.91	7.77	6.72	14.41	6.82	10.16	9.19	18.22	7.65	11.63	9.73	19.43	8.16	12.47	9.73	19.56
	Time pr node/round	33.45	60.01	48.32	115.51	76.34	117.01	104.39	213.09	122.56	190.41	155.44	319.10	179.40	282.03	215.39	437.26
	Result pr node/round	8.21	17.49	6.72	14.41	9.89	19.66	9.19	18.22	10.00	19.99	9.73	19.43	10.00	20.00	9.73	19.56
	Received pr node/round	16.50	35.42	22.55	61.06	44.94	71.98	58.10	121.65	81.58	127.44	100.68	204.10	127.30	201.09	151.16	308.13
	Dropped pr node/round	6.70	14.59	9.03	27.17	28.53	47.41	38.02	82.20	80.93	130.57	102.22	213.63	214.06	357.93	252.30	550.55
Delay 5ms	Sent pr node/round	4.29	6.34	6.60	14.36	5.52	7.88	9.27	18.21	6.32	9.32	9.65	19.50	7.04	10.39	9.72	19.35
	Time pr node/round	38.86	55.32	47.48	116.21	70.40	98.23	105.09	215.19	111.86	160.54	155.00	320.36	169.22	244.19	214.03	433.71
	Result pr node/round	8.61	17.66	6.60	14.36	9.93	19.70	9.27	18.21	10.00	20.00	9.65	19.50	10.00	20.00	9.72	19.35
	Received pr node/round	18.24	31.39	21.92	60.94	41.60	60.17	58.68	122.23	75.50	108.40	100.02	205.64	121.80	176.21	150.17	305.76
	Dropped pr node/round	3.77	9.93	8.95	26.90	20.34	33.93	38.55	81.78	63.38	100.34	101.55	214.37	180.66	291.94	252.07	543.95
Delay 10ms	Sent pr node/round	3.64	5.17	6.57	14.47	4.37	6.21	9.09	18.17	5.16	7.37	9.76	19.48	6.09	8.69	9.73	19.61
	Time pr node/round	40.80	51.68	47.56	117.95	61.44	83.01	102.19	213.52	96.04	131.83	156.97	322.31	151.08	209.41	214.77	436.81
	Result pr node/round	8.63	17.73	6.57	14.47	9.92	19.65	9.09	18.17	10.00	20.00	9.76	19.48	10.00	20.00	9.73	19.61
	Received pr node/round	15.79	26.53	21.94	62.26	34.39	48.92	57.58	121.38	63.73	88.28	100.89	206.24	108.84	151.31	150.54	309.38
	Dropped pr node/round	2.48	6.62	8.87	26.63	14.21	24.59	37.59	81.69	48.99	75.94	102.65	213.81	152.46	239.39	252.27	552.66
Delay 25ms	Sent pr node/round	2.84	3.55	6.81	14.60	2.84	3.79	9.21	18.34	3.21	4.20	9.74	19.48	3.71	5.04	9.69	19.62
	Time pr node/round	60.21	61.02	49.61	118.65	60.78	74.42	104.55	216.88	78.84	95.68	155.87	322.03	112.06	140.47	215.22	438.29
	Result pr node/round	8.72	17.61	6.81	14.60	9.89	19.59	9.21	18.34	10.00	20.00	9.74	19.48	10.00	20.00	9.69	19.62
	Received pr node/round	13.24	20.39	22.95	62.06	26.29	35.02	58.85	123.18	46.28	57.85	100.98	205.46	77.09	100.16	151.00	309.28
	Dropped pr node/round	0.82	2.00	9.11	27.38	5.21	9.41	38.09	82.12	23.69	35.43	102.62	213.96	81.64	125.35	251.06	553.00
Delay 50ms	Sent pr node/round	2.68	3.11	6.66	14.73	2.24	2.76	9.38	18.19	2.10	2.61	9.68	19.43	2.26	2.73	9.77	19.55
	Time pr node/round	107.34	101.53	48.14	119.97	86.35	98.78	106.42	214.18	87.88	101.32	155.29	320.64	102.81	114.87	216.32	434.69
	Result pr node/round	8.61	17.66	6.66	14.73	9.91	19.66	9.38	18.19	10.00	20.00	9.68	19.43	10.00	20.00	9.77	19.55
	Received pr node/round	12.71	18.54	22.16	63.02	23.08	29.21	59.66	121.67	37.28	45.40	100.27	205.53	61.08	71.65	151.55	307.92
	Dropped pr node/round	0.39	0.90	9.08	27.32	1.93	3.06	38.63	81.46	8.58	12.45	101.68	213.26	35.69	50.80	253.63	550.61
Delay 100ms	Sent pr node/round	2.54	2.97	6.71	14.44	2.06	2.55	9.24	18.40	1.68	2.01	9.66	19.49	1.54	1.79	9.76	19.50
	Time pr node/round	197.59	194.76	48.52	116.90	158.08	182.31	105.47	216.36	137.13	154.32	156.12	322.98	135.67	149.41	216.57	435.20
	Result pr node/round	8.65	17.80	6.71	14.44	9.92	19.63	9.24	18.40	10.00	20.00	9.66	19.49	10.00	20.00	9.76	19.50
	Received pr node/round	12.25	17.72	22.57	60.96	21.87	28.31	59.19	123.53	34.06	40.64	100.51	206.31	54.11	63.72	151.85	307.34
	Dropped pr node/round	0.22	0.65	8.97	26.99	0.83	1.39	37.77	82.77	2.92	4.18	101.04	213.69	12.66	17.16	252.77	549.04

Table 3 Results from the simulations

It has not been possible to store the 96 complete trace files produced during the simulations as some of the simulations[22] have produced files in excess of 3.4GB. The extracts and manually collected data are available electronically, including the movement files produced with “setdest.exe”.

Figure 20 is a screenshot from “nam” [56] as it simulates the information exchange between 50 nodes. At this point in time 6 of the nodes have been assigned the task as ICE-initiators[23] and they have started the broadcast. The aim is to get every node fully updated with all available information.



```
32 received new 3 from 9
32 received new 4 from 9
32 received new 6 from 9
```

Figure 20 Screenshot from Network Animator, “nam”, during simulation

[22] ICE 200 nodes ≈ 1.4GB - Pure 200nodes ≈ 3,4GB
[23] Yellow circles

4.2.7. Support scripts for the simulator

Two support scripts written in Python [57] can be found in APPENDIX E. There is a script to automate the execution of the simulator with multiple parameters, and there is a script for extracting information from the large trace files that is produced by the simulator.

The scripts themselves are only used for support with the unattended execution of multiple simulations and extraction of data from trace files. Retrieving data from the trace files is difficult since the files tend to get very large as the number of vehicles and messages increase. Many programs import the complete file, and will not allow access to the content on a line by line manner. The script will do this and count the number of the various message actions within the file. This is very time saving, as there are several million lines[24] within each trace file.

[24] One line each time an action is made to a message. Such as send, receive and drop. “Pure flooding” with 200 nodes ≈ 25 millions messages transactions during a 100 round simulation.

5. DISCUSSION

This chapter will analyze the results from chapter 4 and present correlations between the various values to measure the performances.

This thesis is dealing two main themes. Border determination and information exchange.

The border determination tried to find a set of nodes that would form a border around the remaining nodes. The theory presented in chapter 2.4 and partly chapter 2.5, combined with the mathematical background from chapter 3.3 gives a possible solution to the border determining problem, as illustrated in Figure 12. The functionality and validity of this solution was investigated using a Java program as presented in chapter 4.1.

As seen by the illustration in Figure 18 the borders encapsulates all the nodes, this will be the case in any trial using the java code in APPENDIX B. The program is performing according to the proposed solution, and the result is a border that is either concave or convex.

There is a problem related to the concave border[25] as it might produce very sharp and unnatural angles. This is not a problem with the convex border[26] as it always uses the shortest path between the nodes, without leaving any node on the outside of the border.

Nodes that do not have the condition might be included under false circumstances and thus classified as being within the border when this is not the case. This could also happen when concave elements are removed, as this encapsulates a bigger area that might contain other irrelevant nodes.

[25] Red border in Figure 13 and Figure 18
[26] Green border in Figure 13 and Figure 18

There are at least two reasons for a node not to be in the list that is used to find the border.

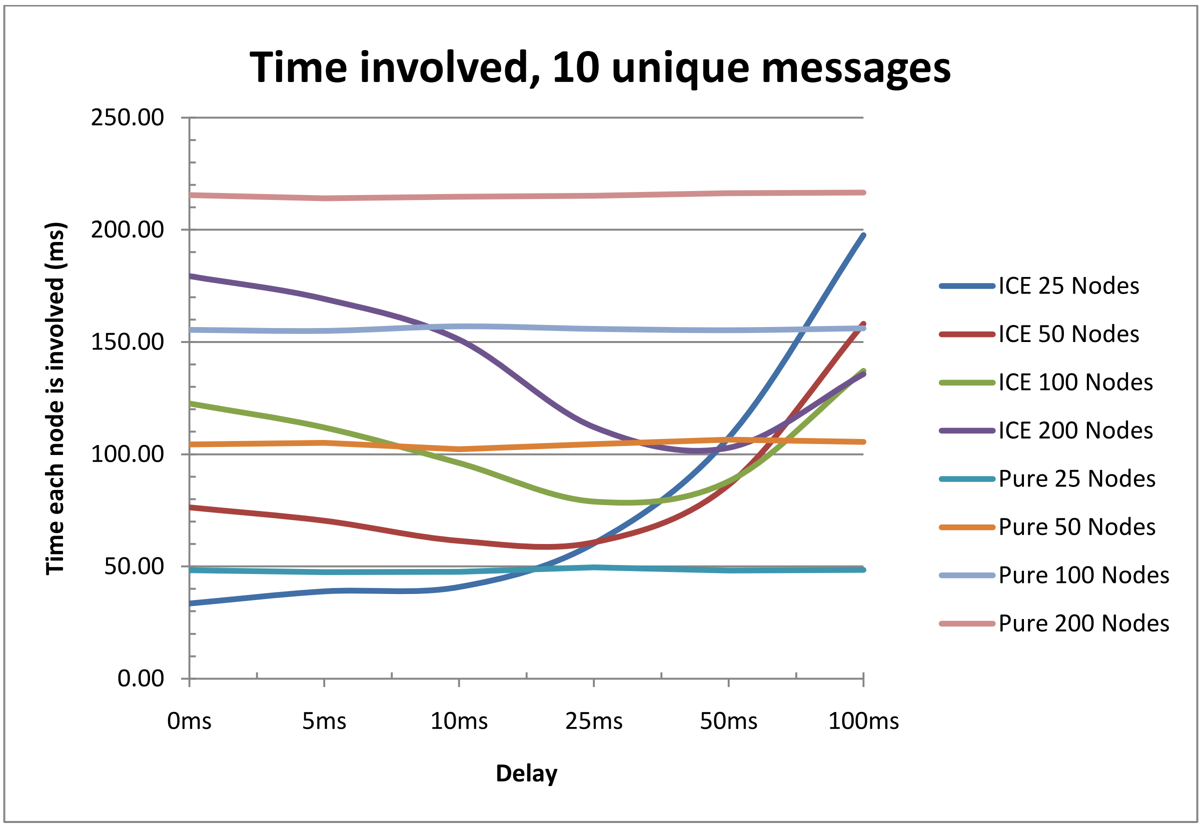
- They might not be equipped with the right sensors or communication equipment, in which case it does not do any harm to include them inside the border, as it cannot be proven that they do not belong within the border.
- They might not have conditions to qualify them to be included inside the border, but this is dependent on the nature of the condition. As an example, there might be places within a raincloud where it does not rain, but that might just be a coincident. The absence of a condition does not mean that the condition does not apply at the position.

The information exchange method named ICE was tested using an NS2 simulator. Results gathered from a simulator will generally be inferior to results gathered from real-life experiments since there probably will be factors that are not considered or even anticipated. A simulator is a good method to gather huge amount of data that can be analysed to say something about what behaviour might be expected. The result of a simulation will normally be averaged values that indicate what kind of performance a system might deliver. Because of this it is necessary to test the system with as many different parameters, and scenarios as possible, to get the most accurate results.

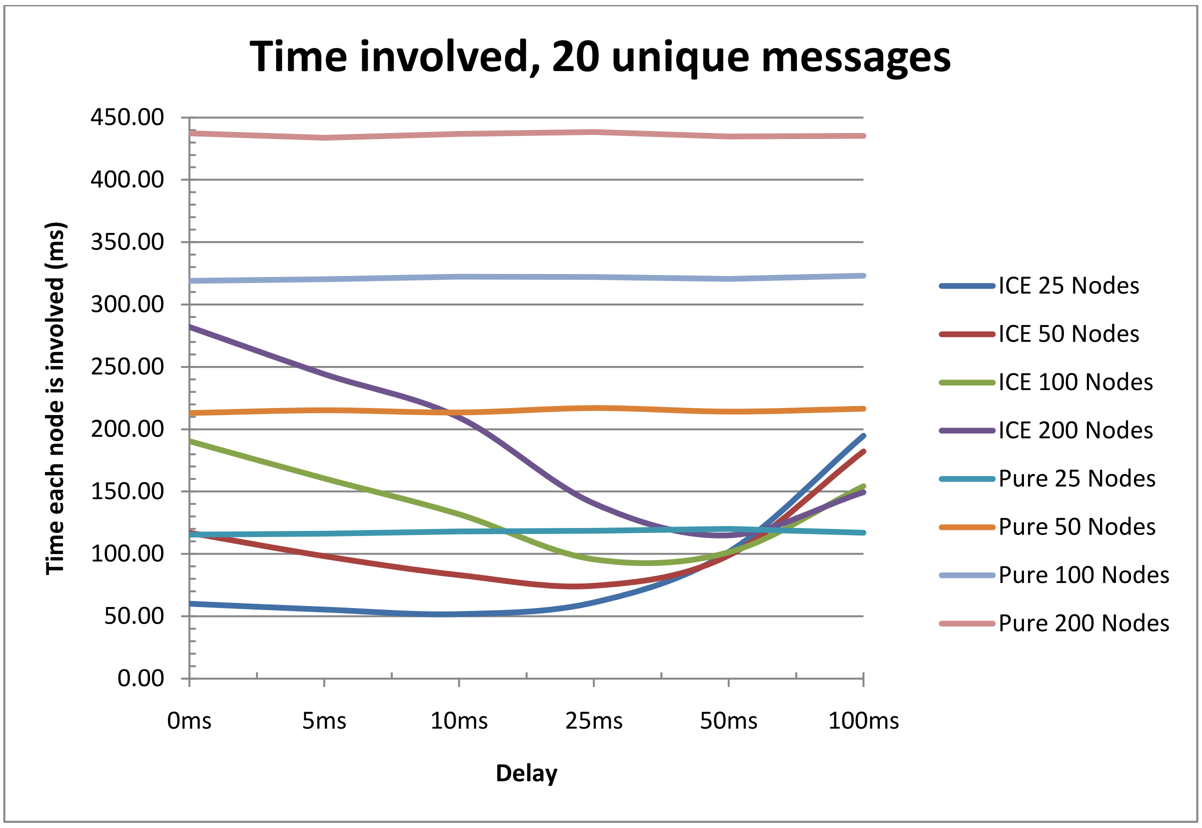
The results form ICE and “pure flooding”, as they are represented in Table 3, can be organized into graphs to make the correlation between the two methods and their different parameters more visual.

Graph 1 and Graph 2 illustrates how long each node is involved in the transmission as a function of the different delays, and with different number of nodes. The time a node is involved is calculated as the time difference between the first and the last message received at each node.

The basic parameters for the data displayed in Graph 1 and Graph 2 are identical except for the movement patterns and the amount of information that is available in each system. 10 unique messages are available to the network in Graph 1 and 20 unique messages are available in Graph 2.



Graph 1 Average time each vehicle is involved in the communication, 10 messages



Graph 2 Average time each vehicle is involved in the communication, 20 messages

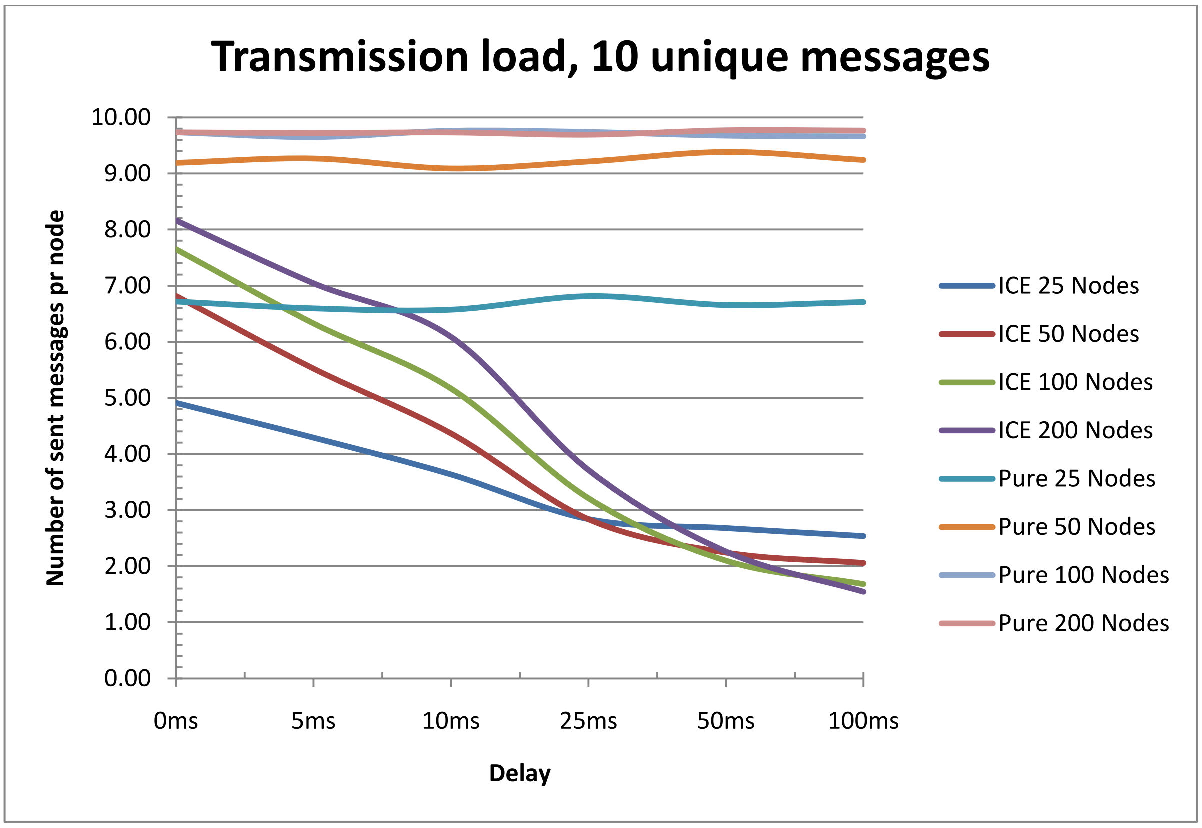
The shapes of Graph 1 and Graph 2 are very similar except for the placements on the Y-axis. This indicates that the involvement time does not influence the behaviour; it just alters the range and internal placement of the lines.

“Pure flooding” is by nature not influenced by the delays in transmission. In the code presented in APPENDIX D the delays are omitted altogether, causing the transmission to happen as soon as the message processing is finished. If delays had been included it would not have made any difference since all the messages would have been subjected to the same delay, and thereby cancelling out the effect. This causes “pure flooding” to be presented as relative straight lines in all these graphs. The consistent result of “pure flooding” is a good indicator that the movement pattern for nodes is fair and consistent during the simulations. The consistency of the “pure flooding” data emphasise the influence time delays have on the curves describing ICE behaviour.

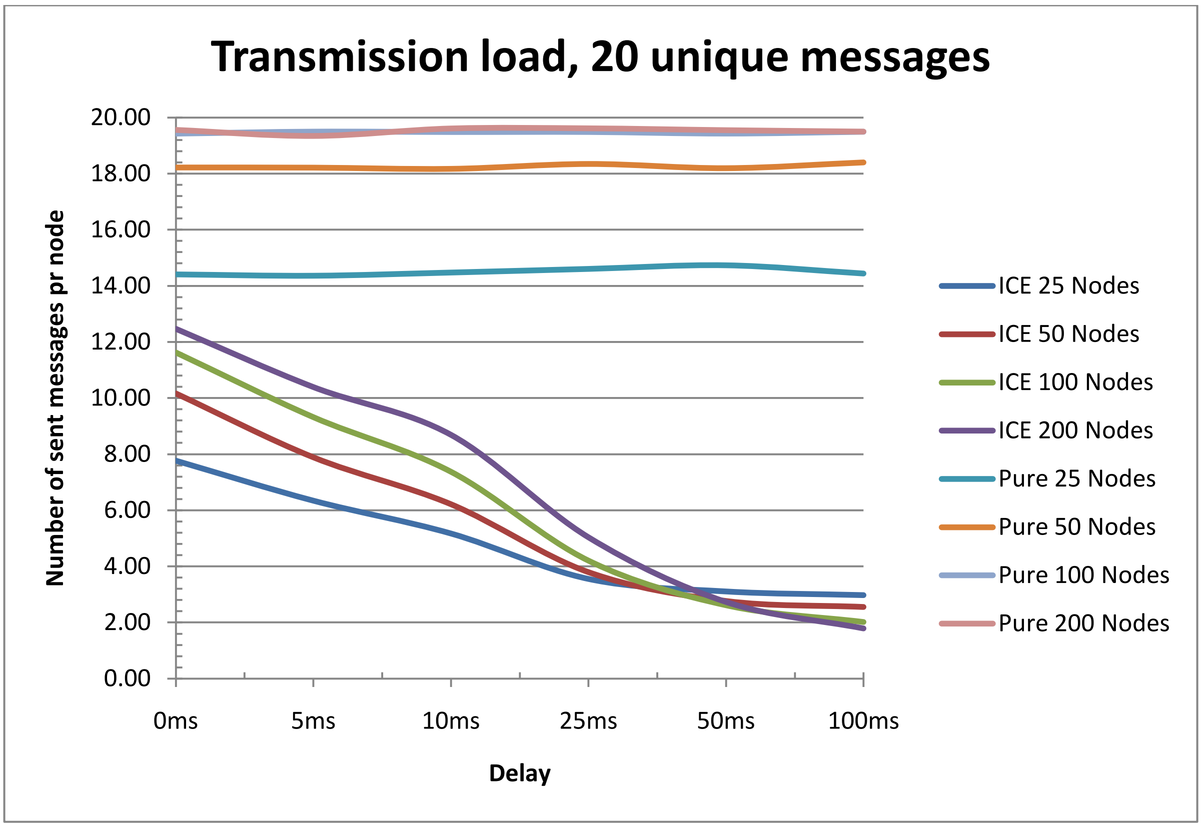
From Graph 1 and Graph 2 it is clear that with ICE the time between first and last message decreases as transmission delay is increased towards 25 and 50 ms, beyond this critical transmission delay time involvement time increases. This reveals a possible weakness in ICE, as it is influenced by the duration of the delays. The involvement time of ICE is longer and less efficient than “pure flooding” when the network is very sparsely populated and delays are increased beyond the critical transmission delay time.

The decrease in involvement time is explained by the fact that more messages are aggregated reducing transmission load leaving fewer messages to be transmitted as the delays increase, thus less time passes between the first and the last message at each node. The increasing curve after the apex indicates that time is wasted waiting for messages to arrive. The aim is for each node to spend as little time as possible involved in transmission. When considering involvement time it seems that the ideal delay is somewhere around 50 ms. At this point the involvement time related to 200 vehicles using ICE is reduced by as much as 74% compared to the 200 vehicles using “pure flooding”.

Graph 3 and Graph 4 illustrates the transmission load represented by the number of messages sent by each node at the varying delays and different number of nodes present.



Graph 3 Transmission load from each vehicle, 10 messages in the network



Graph 4 Transmission load from each vehicle, 20 messages in the network

In Graph 3 and Graph 4 it is evident that in ICE the number of messages needing to be transmitted decreases as the delays increase. These two graphs are very similar which supports the assumption that the system is acting the same way independently of the number of unique messages that is available to the nodes.

The number of sent messages is a good measure of the system load as it is directly linked to the amount of communication present at any given time. The fewer messages the system transmits the less likely the occurrence of collisions and lost messages. An ideal value would be as small as possible and it seems that in ICE the number of transmitted messages is decreasing as long as the delay is increasing. This is a predictable behaviour as each node gathers more and more information before a message is transmitted. The value will tend towards 1, but it is unrealistic to aim for such a low value, since this would mean that the nodes would waste very much time waiting.

There is a change in both graphs when the delay is around 50 ms as all the lines representing ICE cross each other; this indicates that this is an ideal delay regarding sent messages. At this point ICE sends an average of 2.32 out of 10, and 2.80 out of 20 unique messages. This represents a significant reduction compared to the 10 and 20 messages that “pure flooding” transmits.

There are some concerns regarding the calculation of time each node is involved if the number of transmitted messages drops as low as 1. The involvement time is calculated as the difference between the first and the last message. Nodes that only receive 1 message would not be counted as involved, and might skew the result. The simulations in this thesis have not got long enough delays for this to be a problem.

When “pure flooding” behaves as designed any unique message should be retransmitted as soon as it is received. Thus “pure flooding” should, in Graph 3 and Graph 4, present straight lines at 10 and 20 transmitted messages respectively. This is not the case, leading to the conclusion that messages are lost. This will be investigated further in Graph 7 to Graph 10 and in Table 5.

To get a good measure of how ICE performs it is necessary to compare it against “pure flooding”. The fact that “pure flooding” loses some of the available information reduces the transmission load. To get a more accurate comparison it is possible to put ICE up against an idealized version of “pure flooding”. When “pure flooding” is working perfectly and there is no loss of messages each node will transmit the exact same number of messages as there are unique messages available[27] in the system.

Based on Equation 6 and Table 3, Table 4 shows in percent how much the number of transmitted messages is reduced on average at each node in ICE compared to how an ideal “pure flooding” would perform, with different number of nodes and varying delays.

$$Improvement = \frac{(unique\ messages - sent\ messages)}{(unique\ messages)}\%$$

Equation 6 Improvement calculations

		ICE							
Nodes		25		50		100		200	
Messages		10	20	10	20	10	20	10	20
Max delay (ms)	0 ms	50.88 %	61.15 %	31.84 %	49.22 %	23.55 %	41.86 %	18.40 %	37.66 %
	5 ms	57.07 %	68.29 %	44.78 %	60.59 %	36.75 %	53.40 %	29.59 %	48.04 %
	10 ms	63.63 %	74.14 %	56.34 %	68.95 %	48.35 %	63.13 %	39.15 %	56.55 %
	25 ms	71.59 %	82.24 %	71.63 %	81.05 %	67.89 %	78.99 %	62.87 %	74.81 %
	50 ms	73.20 %	84.47 %	77.58 %	86.19 %	79.00 %	86.94 %	77.40 %	86.33 %
	100 ms	74.62 %	85.13 %	79.43 %	87.26 %	83.17 %	89.94 %	84.57 %	91.05 %

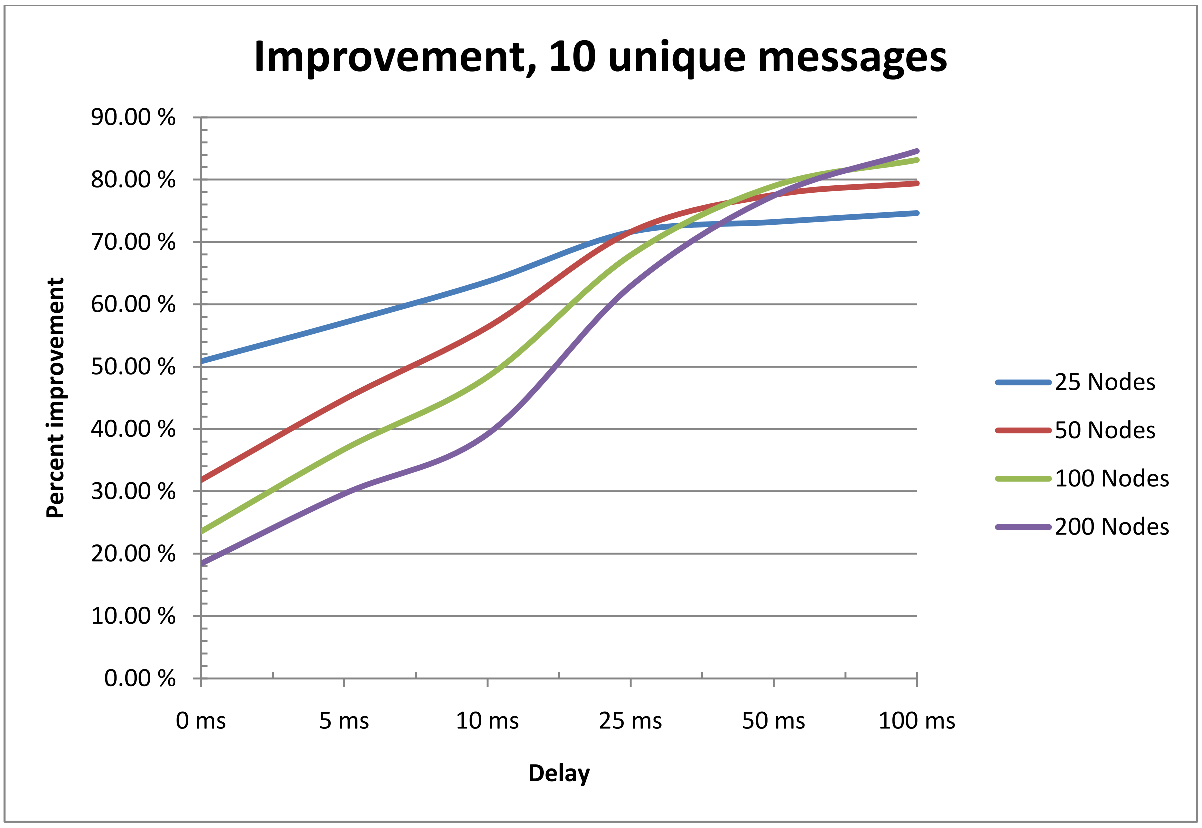
Table 4 Improvement of ICE compared with ideal “pure flooding”

Table 4 shows that transmission load decreases as the delays increase, and the best values are obtained with a long delay and many nodes in the system.

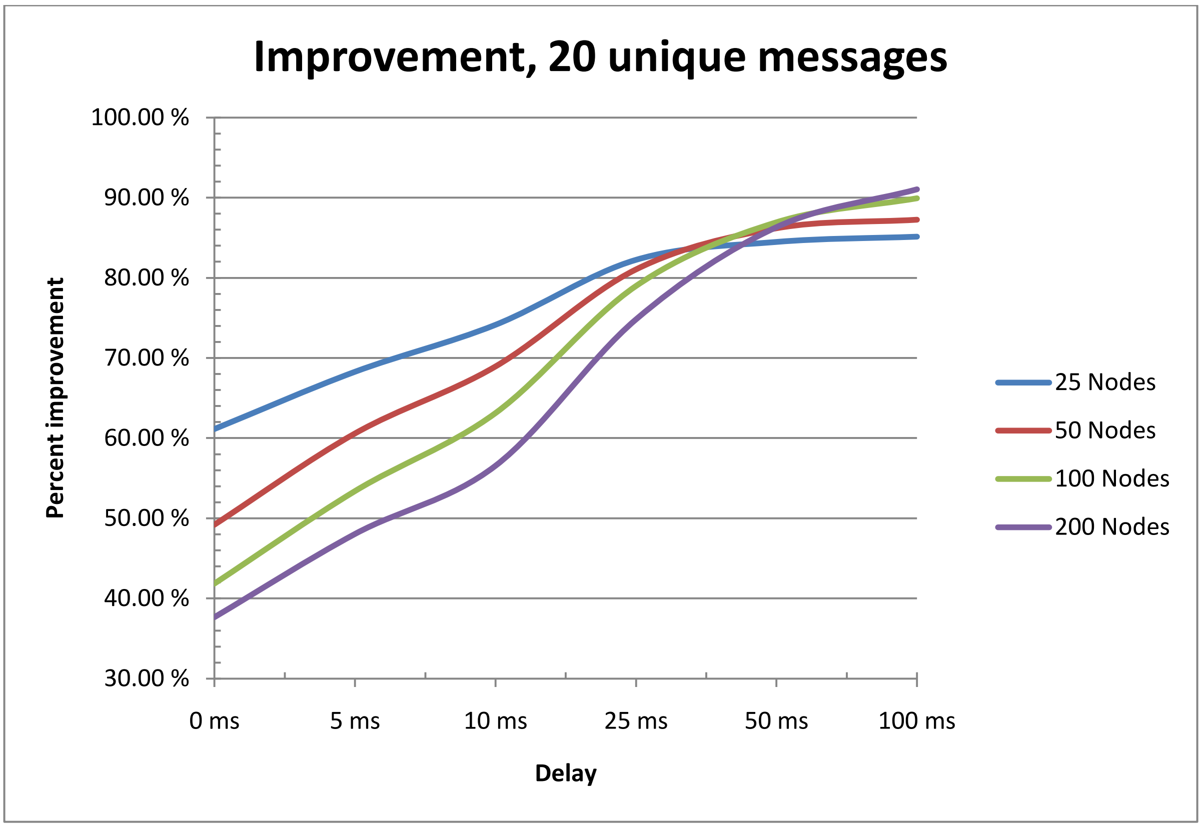
The results in Table 4 are illustrated by Graph 5 and Graph 6. These show that transmission load reduction varies from just 18% to above 90% depending on the number of nodes in the system, the delay imposed before transmission and the number of unique messages available.

A large reduction is preferable, but it comes at a price, the delay in the whole system increases, and resources is wasted on other areas. If there is a demand for rapid exchange, sacrificing some of the transmission capacity might be necessary to make the communication more flexible.

[27] In this thesis there are either 10 or 20 unique messages available.



Graph 5 ICE compared with ideal "pure flooding", 10 messages in the network



Graph 6 ICE compared with ideal "pure flooding", 20 messages in the network

Graph 5 and Graph 6 clearly display the effect of including a delay before transmission, and gathering more information into a single message before transmission. These two graphs have a similar shape indicating that the shape of improvements experienced with ICE is independent of the number of unique messages available. The metrics will differ slightly as there are greater improvements to be detected as the number of unique messages to be transmitted increase.

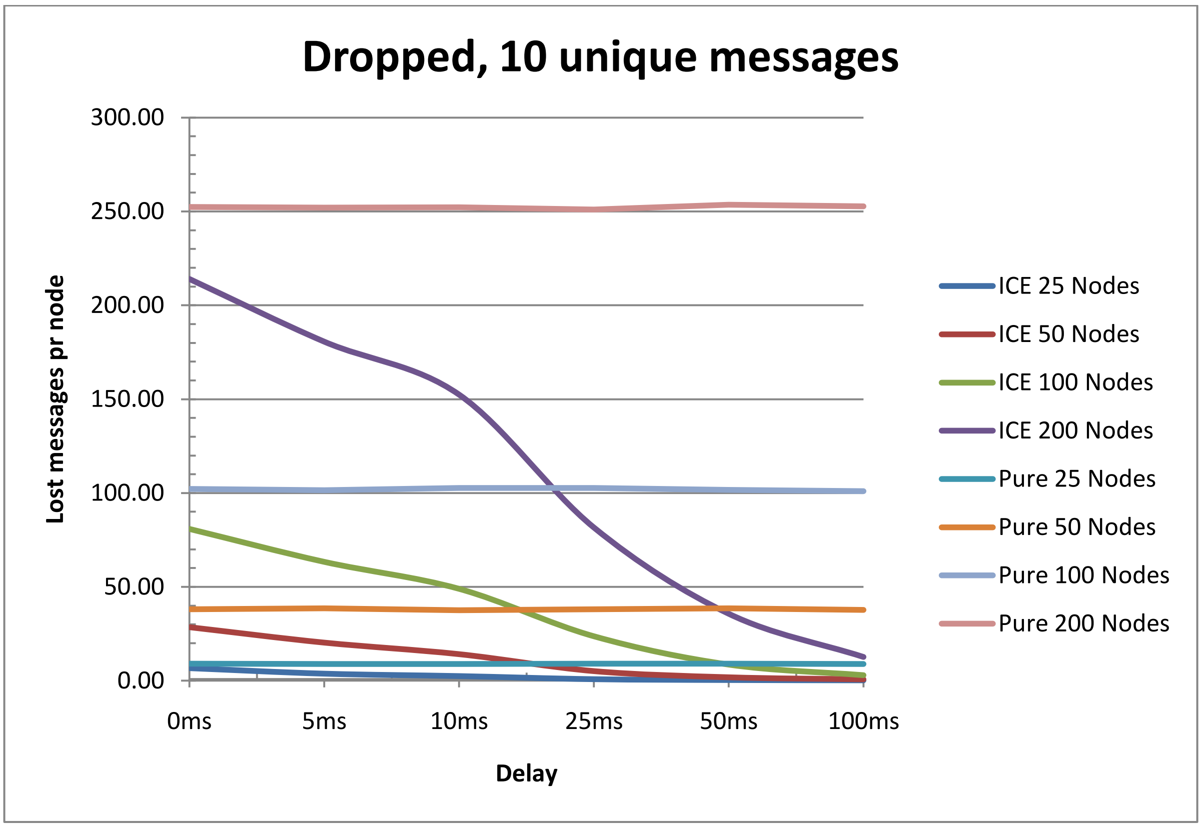
In both graphs the lines cross each other at around 50 ms, this is easily explained by the similar behaviour in Graph 3 and Graph 4. The average reduction in number of sent messages seems to be 77% with 10 unique messages, and 86% at 20 unique messages in the network at this point. These reductions are even better than the estimate (between 57% and 68%)[28] made in chapter 2.6.1 regarding the transmission load in the system described by Figure 5.

Graph 7 and Graph 8 illustrates the number of dropped messages at each node as a function of the number of nodes, and the transmission delays.

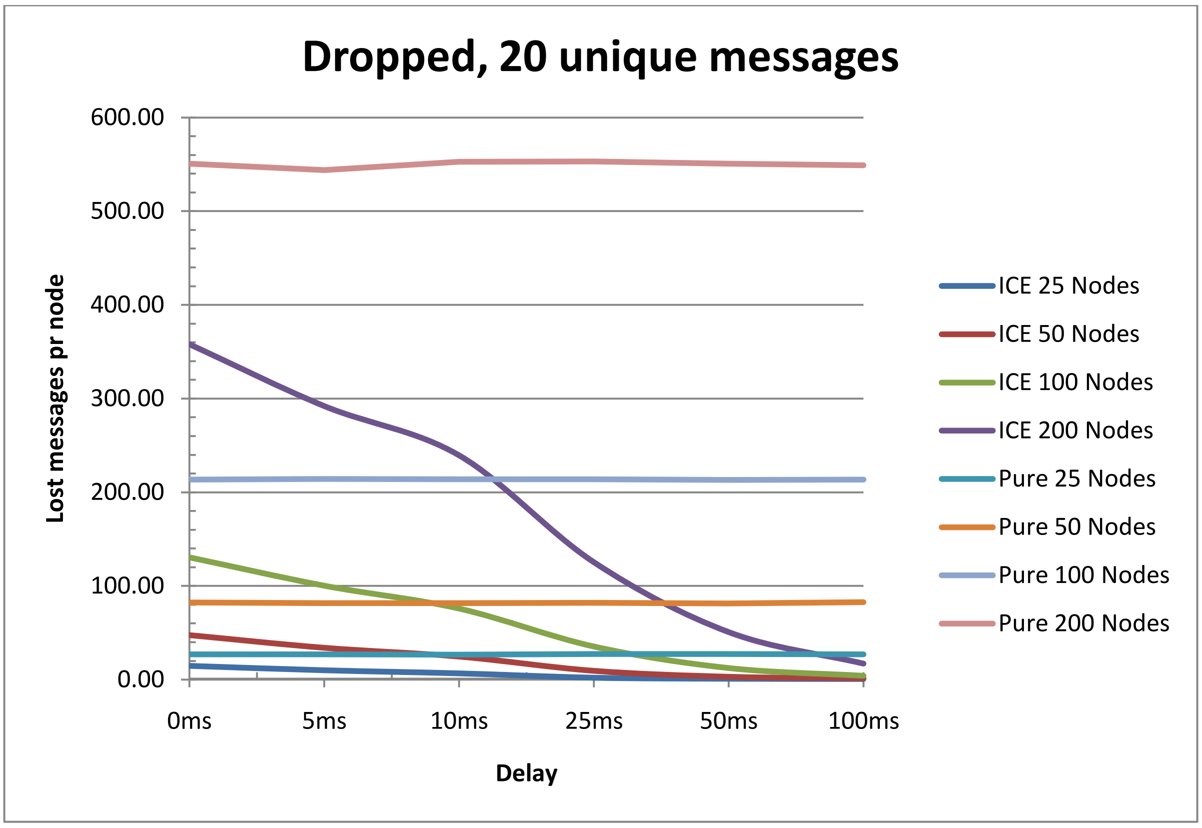
The number of dropped messages in ICE is always lower than the corresponding values for “pure flooding”. With higher delays the difference is quite noticeable. This can be attributed to the decreasing probability of collisions when the number of messages is reduced.

The “best effort” nature of UDP as transmission method does not generate retransmissions with failed messages, but there is still a need to reduce the number of lost messages as they might consume significant amounts of computation power. In the worst cases the last bits of a message is distorted, or lost, and the message is rendered useless. The consequence of this is that some vehicles might not receive all the available information, and by that make a wrong interpretation of network status.

[28] In the simulation “pure flooding” is constructed to treat all stored information as one message (best case from Figure 5)



Graph 7 Average number of lost packets pr vehicle, 10 messages in the network



Graph 8 Average number of lost packets pr vehicle, 20 messages in the network

Graph 7 and Graph 8 give a picture of the number of dropped messages at each node, but the graphs do not take into consideration the reduced number of messages in the system, thus skewing the graph in favour of ICE.

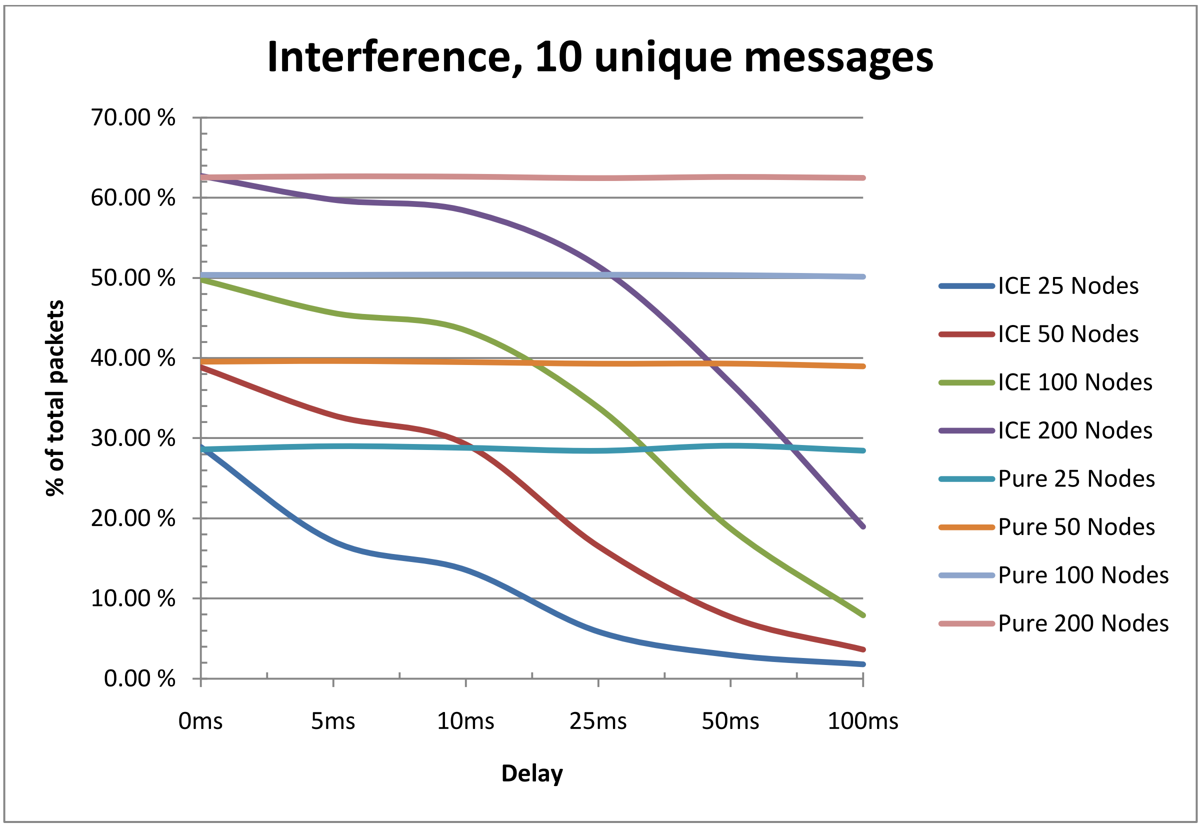
To get an accurate estimate of how much interference there is in the system it is necessary to not only focus on the dropped messages. When the total number of messages in the system is taken in to consideration the graphs get more detailed. The sum of dropped and received messages would provide a much better basis for interference calculations, and avoid the unjustified appraisal of ICE over “pure flooding”.

The values in Table 5 are based upon the “Received pr node/round” and “Dropped pr node/round” values from Table 3. They give a fair assessment of how much interference each method introduces.

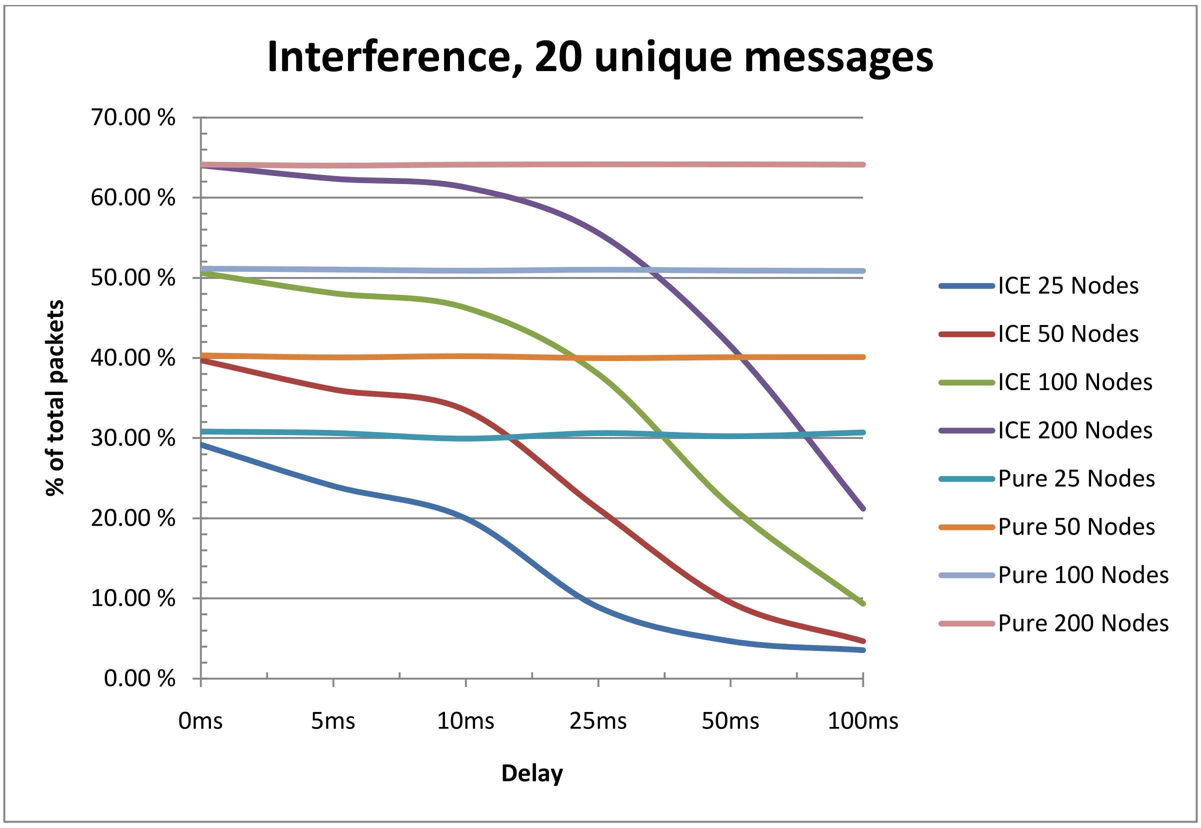
			Delay					
	Vehicles	Messages	0ms	5ms	10ms	25ms	50ms	100ms
Ice	25	10	28.88 %	17.12 %	13.56 %	5.86 %	2.95 %	1.79 %
		20	29.18 %	24.02 %	19.98 %	8.91 %	4.65 %	3.53 %
	50	10	38.83 %	32.84 %	29.23 %	16.53 %	7.70 %	3.64 %
		20	39.71 %	36.06 %	33.46 %	21.17 %	9.49 %	4.67 %
	100	10	49.80 %	45.63 %	43.46 %	33.86 %	18.71 %	7.89 %
		20	50.61 %	48.07 %	46.24 %	37.98 %	21.52 %	9.33 %
	200	10	62.71 %	59.73 %	58.35 %	51.43 %	36.88 %	18.96 %
		20	64.03 %	62.36 %	61.27 %	55.58 %	41.49 %	21.21 %
Pure flooding	25	10	28.59 %	29.00 %	28.80 %	28.42 %	29.06 %	28.44 %
		20	30.80 %	30.62 %	29.95 %	30.61 %	30.24 %	30.69 %
	50	10	39.55 %	39.64 %	39.50 %	39.29 %	39.31 %	38.95 %
		20	40.32 %	40.09 %	40.23 %	40.00 %	40.10 %	40.12 %
	100	10	50.38 %	50.38 %	50.43 %	50.40 %	50.35 %	50.13 %
		20	51.14 %	51.04 %	50.90 %	51.01 %	50.92 %	50.88 %
	200	10	62.53 %	62.67 %	62.63 %	62.44 %	62.60 %	62.47 %
		20	64.12 %	64.02 %	64.11 %	64.13 %	64.13 %	64.11 %

Table 5 Interference for ICE and “pure flooding”

Graph 9 and Graph 10 is used to illustrate the content of Table 5. It is evident that the percentage values are very similar, in both graphs. This indicates a consistent behaviour independent of the number of unique messages in the system.



Graph 9 Average interference pr vehicle, 10 messages in the network



Graph 10 Average interference pr vehicle, 10 messages in the network

At short delays in Graph 9 and Graph 10 ICE perform at a level similar to that of “pure” flooding. When the delays increase the interference caused in ICE drops significantly.
At 100 ms even the traffic generated by 200 nodes using ICE is not causing as much interference as “pure flooding” with just 25 nodes. The curves for the different number of vehicles indicate that the greatest reduction in interference occur with delays between 10 and 50 ms, but the decline continues downwards even with longer delays.

ICE with 25 nodes seems to reach its minimum when the delay exceeds 50 ms, as the curve levels out. It is likely that the other curves will level out as well if the delay is increased past 100 ms.

At 50 ms the reduction in interference is between 35% and 89%, depending on the context. The average reduction of interference is 66%, and this represents a significant improvement. The reduced interference alleviates some of the pressure on the bandwidth and by that enables more efficient use of the resources in the network.

In every graph that includes “pure flooding” its performance is depicted as a straight line. This is not particularly strange as “pure flooding” is not influenced by the various delays, but it also underlines that the results of the simulations are consistent. ICE and “pure flooding” have used the same movement pattern for the nodes at the various delays, and this indicates that the sample is large enough to get a significant result.

The movement patterns are constructed to allow the nodes to move around at random with a top speed of 30 km/h. During a 1000 seconds simulation each car might cover a distance in excess of 8000 meters. The average distance each node would travel should be about half this maximum distance, i.e. 4000 meters. This equals 4 times the size of the area used in the simulation. On the basis of this the probability that a wide variety of scenarios will have occurred in one form or another during the simulations is significant. With several scenarios in each simulation the results are stable and consistent.

It is of special interest that the curves representing ICE in the graphs show a change in tendency when the delay is in the area of 50 ms.

- In Graph 1and Graph 2 there is a change of direction, from less time involved, to more time involved pr node.
- In Graph 3and Graph 4 all the curves cross each other indicating a change in the performance. This change is also visible in Graph 5and Graph 6 as these graphs are based mostly upon the same data.
- The changes in Graph 7 and Graph 8 are more subtle as the curves start to level out, this is can also be assumed to be the case for Graph 9 and Graph 10, as these are based upon the same data.

The delay is a random value in the subset [0, max].With a sufficiently high number of samples the average delay will trend towards half of the max-value. This means that if the delay is set to 50 ms then each vehicle will wait on average 25 ms before transmitting any messages. The delay needs to be randomly selected to avoid congestion and collisions. A fixed delay would only cause all the transmitters to transmit simultaneously, and the behaviour would be similar to having 0 ms delay.

Simulations will never be able to give a completely accurate interpretation of the real world, but they might provide reasonable metrics to begin the trial with real hardware. The NS2 simulator has been developed over the years to become one of the most realistic simulators available to the public.

The results from the simulations in chapter 4 were as expected, and the results show a significant reduction in transmission load, and interference, compared to “pure flooding”.

6. CONCLUSION & FUTURE WORK

This chapter draws conclusions based on the analysis performed in chapter 5, and relates the findings to work done on similar themes. The chapter also contains some suggestions for further work that could benefit the development of V2V communication.

6.1. Conclusion

The java program in APPENDIX B shows that it is possible to construct, or determine a border based upon the position of individual nodes. The choice of whether the border should be concave or convex is dependent on the conditions that are involved when constructing the border. There is a real chance of catching nodes within the border that does not belong there, but this is acceptable to some degree. If the number of vehicles that are included within the border by mistake gets to high there should be a mechanism to increase the level of detail so that as few vehicles as possible are included within the border by mistake.

The inclusion of irrelevant vehicles will have an impact on the homogeneity of the information if the border information is exchanged through the network. This might cause different interpretations about what the status is within the network. This will not be a problem for surrounding vehicles as long as the range of detail in information is increased as the vehicle’s distance to the border decreases.

The border determination does not exclude the possibility of having different semi-overlapping borders; the borders can be treated as individual entities.

By determining the border it is possible to reduce the amount of information needing to be exchanged. The whole cell might be characterised by the elements that constitute the border, instead of exchanging information about every element within the cell. This results in a significant reduction in transmission load. The bigger the cell, the less data is needed to describe the conditions of the vehicles within the cell.

The use of min/max-pairs to determine the border is simple and fast, but it is not easy to adjust the granularity of the border. It is also difficult to determine the actual shape of the border in any geometrical sense.

Another method of determining the border is being developed by Gamati Emadin [58]. The development of his method uses some of the work done in this thesis, as a foundation. The method includes 4 nodes instead of 2 nodes when determining the border. The use of two dimensional structures makes it possible to alter the granularity of the border by including several semi-overlapping structures. This would produce a border that has multiples of 4 elements in the border, and by that have a predictable shape. The shape would start off as a square, and can be altered into an octagon by including 4 more positions. The possibility of having a known degree of polygons might give more possibilities for mathematical analysis. A problem with this arises when the number of nodes is small and there no longer are enough elements to make multiples of 4.

The method used for distribution of information throughout the network has a profound effect on the efficiency and transmission load in the network.

By implementing neighbour update with “Hello” messages it is possible to avoid involving all the vehicles in the network when a new vehicle requires new information. The inclusion of directionality in the neighbour update scheme ensures that the vehicles receive the most relevant information available, as the vehicles in the same area tend to need, or detect, the same information. This is supported by an article of “Yildiz, Pagliari, Ozdaglar and Scaglione” [59]. Their work is a detailed analysis of how the information is migrating based on neighbour updates. Adopting the interpretation of one neighbour, or when there are more responses, using a majority vote to determine the status, will over a period of time, provide some uniformity in the overall status interpretation. To ensure that every vehicle in the network has the same interpretation about the status a complete exchange of information is needed.

The result from the simulations of ICE show significant reductions in transmission load, lost messages, interference and involvement time compared to the corresponding results of ”pure flooding”. The different delays used in ICE have different effects on the results. Long delays give a positive result regarding lost messages, interference and sent messages as the amount of information that is aggregated and combined into a single message is increasing. The delay has a very different effect on the involvement time for the vehicles. The optimum delay as seen by the graphs is about 50 ms.

Delays larger than 50 ms cause unnecessary waiting and wastes recourses. The work done by Yueyue Li [60] investigates how much the messages are delayed when using various routing protocols in an inter-vehicle communication and how predictable vehicles like buses or trams can be utilized as information carriers. By including ICE in this setting it is possible to reduce the time each vehicle is involved in the communication even further. In this thesis ICE is simulated with a “dumb agent” protocol that does not specify destination or route.

Based on the optimum delay derived from the involvement time in Graph 1 and Graph 2 it seems feasible to get a significant reduction in the number of sent messages, and therefore reduction of transmission load. With a 50 ms delay the involvement time for 200 vehicles is reduced by 74% with ICE. The number of sent messages and the transmission load can been reduced by as much as 86%. This reduction has another benefit as it reduce the interference by 66% in average, and between 35% and 89%, depending on how many vehicles and unique messages there are in the network, compared to “pure flooding”. The significance of aggregating information by introducing delays before transmission has been investigated by Gamati Emadin [58], and his ongoing work with “N-hop information gathering” experiences a similar effect when the delays are increased.

The simulations of ICE is made as realistic as possible, by making the vehicles that have the initial information send their first message at a random time. This avoids them all transmitting at the same time, and imitates the vehicles discovering the information by themselves. The notion of 10 (or 20) vehicles discovering many different conditions that are worth exchanging, within a small window of time might be unrealistic, but it makes it possible to analyze the effect large transmission loads have on the system.

As a broadcast method ICE is superior to “pure flooding” in concerns of transmission load and interference. With a suitable choice of delay time ICE performs better regarding involvement times as well.

“Pure flooding” is easier to implement, but is outperformed by ICE in almost all the situations that were tested by the simulator in this thesis. The exception is a situation with very sparsely populated networks and long delays. In this case “pure flooding” is the sensible solution. By implementing dynamic changes of delay ICE would be suitable for this scenario as well.

6.2. Future work

The construction of the cells as it is described in chapter 3.3 is not ideal for avoiding the inclusion of irrelevant vehicles. The method will construct the border based on elements that shall be on the border, but it is not able to bend the border around vehicles that do not belong in the cell. Further investigation might provide a solution to solve this problem of wrongfully included nodes in the cells.

There is a need for more research regarding what kind of cell-conditions that are best described using concave or convex borders, and what effect false nodes have on the cell when the border is changed from concave to convex.

Merging of various cells is a field that might benefit from more studies as this directly influences the number, and size, of messages that needs to be broadcasted, and thus the transmission load. The simulations in this thesis have not taken different levels of detail into consideration, but further studies might find ways to utilize this feature to improve the method. Combining ICE with the work currently being done by Gamati Emadin [58] could help limit the number of involved vehicles.

A more detailed study of the effect of delays around the optimum delay from Graph 1 and Graph 2 would make it possible to improve the performance of ICE further. Ideally some investigation should be made into how to determine the optimal dynamical delay time as a function of the number of vehicles. During the work on this thesis some attempts have been made at how to make ICE intelligent enough to learn what would be the optimal delay in any given context. The attempts revealed that there is potential to improve the method by dynamically changing the delay time depending on previous behaviour. A study of dynamical delays might provide further improvements to ICE.

The ICE simulations are conducted with 802.11b/g as transmission technology. The draft standard IEEE 802.11p/Wave [61] shows interesting properties, especially for rapid information exchange. The performance as previously analyzed [62] and confirmed [63] shows promising results regarding rapid delivery of warning messages to vehicles. A simulation that tests the performance of ICE with an 802.11p implementation could provide a useful contribution to the emerging standard.

ICE is currently just tested with simulations. A test with real vehicles and real equipment might reveal problems, or advantages, not detected in the simulations. Testing facilities such as the CVIS test sites [64] might be an ideal environment to conduct such experiments.

The content of this thesis is kept as technology independent as possible, but to make further experiments relevant to new standards any future experiments should make sure to use IPv6 [65] and not IPv4 [66]. It will be important to utilize the mobile functionality that is included in IPv6, as well as conforming to the standards put forward by the CVIS project.

REFERENCES

1. **Sister.** WirelessCar. [Online] ERTICO – ITS Europe, 01 March 2010. [Cited: 01 March 2010.] http://www.sister-project.org/en/about_sister/sister_consortium/wirelesscar.htm.

2. **Kolberg, Jon Einar.** Automatisk redning i Norge. [Internett] Nettavisen, 21 May 2007. [Sitert: 01 March 2010.] http://www.nettavisen.no/motor/article1058077.ece.

3. **BMW.** BMW Assist. [Online] BMW, 14 November 2008. [Cited: 02 March 2010.] http://www.bmw.com/com/en/owners/navigation/assist_1.html.

4. **The Auto Channel.** ATX Launches Automatic Collision Notification for BMW. [Online] The Auto Channel, 01 11 2009. [Cited: 02 March 2010.] www.theautochannel.com/news/2009/01/11/357504.html.

5. **BMW.** BMW Online. [Online] BMW, 2008 December 2008. [Cited: 02 March 2010.] http://www.bmw.com/com/en/owners/navigation/online.html.

6. **Physorg.com.** Inter-vehicle communications may save lives. [Online] 10 August 2005. [Cited: 24 March 2010.] http://www.physorg.com/news5731.html.

7. **RITA.** RITA | Intelligent Transportation Systems (ITS). [Online] U.S. Department of Transportation, 05 June 2010. [Cited: 05 June 2010.] http://www.its.dot.gov/index.htm.

8. **CVIS Project.** CVIS Project. [Online] ERTICO – ITS Europe, 19 February 2010. [Cited: 19 February 2010.] http://www.cvisproject.org/.

9. **Khaled, Yacine, Menouar, Hamid and Challal, Yacine.** Reactive and adaptive protocol for inter-vehicle communications (RAP-IVC). *Information and Communication Technologies: From Theory to Applications, 2004.* Univ. de Technol. de Compiegne, France : IEEE, 2004. Vol. 2004.

10. **Wikipedia UK.** Infotainment - Wikipedia. [Online] Wikipedia, 13 February 2010. [Cited: 04 March 2010.] http://en.wikipedia.org/w/index.php?title=Infotainment&oldid=343651840.

11. **HA EU Watch Project Team.** CALM standards and CVIS. [Online] 01 November 2006. [Cited: 10 May 2010.] http://www.haeuwatchits.info/press/press_detail.asp?pid=135&aid=433.

12. **Wikipedia UK.** Inteligent VANET - Wikipedia. [Online] 19 April 2010. [Cited: 31 May 2010.] http://en.wikipedia.org/w/index.php?title=Intelligent_Vehicular_ad-hoc_Network&oldid=356974446.

13. **IEEE.** 802.11 Standards. [Online] 01 January 2007. [Cited: 01 April 2010.] http://standards.ieee.org/getieee802/download/802.11-2007.pdf.

14. **Wikipedia UK.** Global Positioning System - Wikipedia. [Online] Wikipedia, 16 March 2010. [Cited: 17 March 2010.] http://en.wikipedia.org/w/index.php?title=Global_Positioning_System&oldid=350161239.

15. —. Galileo (satellite navigation) - Wikipedia. [Online] Wikipedia, 17 March 2010. [Cited: 18 March 2010.] http://en.wikipedia.org/w/index.php?title=Galileo_%28satellite_navigation%29&oldid=350444321.

16. **Cisco.** Cisco Wireless Location Appliance. [Online] Cisco, 27 March 2010. [Cited: 30 April 2010.] http://www.cisco.com/en/US/products/ps6386/index.html.

17. **Blum, J.J., Eskandarian, A. and Hoffman, L.J.** Challenges of intervehicle ad hoc networks. *Intelligent Transportation Systems, IEEE Transactions.* December, 2004, Vol. 05, 04.

18. **Wischhof, L., Ebner, A. and Rohling, H.** Information dissemination in self-organizing intervehicle networks. *IEEE Transactions on Intelligent Transportation Systems.* March, 2005, Vol. 06, 01.

19. **Zhao, Jing and Cao, Guohong.** VADD: Vehicle-Assisted Data Delivery in Vehicular Ad Hoc Networks. *IEEE Transactions on Vehicular Technology.* May, 2008, Vol. 57, 03.

20. **Nadeem, Tamer, Shankar, Pravin and Iftode, Liviu.** A Comparative Study of Data Dissemination Models for VANETs. *Mobile and Ubiquitous Systems - Workshops, 2006. 3rd Annual International Conference.* July, 2006, Vol. 03, 01.

21. **Wu, Hao, et al.** MDDV: A Mobility-Centric Data Dissemination Algorithm for Vehicular Networks. *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks.* 2004, Vol. 01, 01.

22. **Sukdea, Yu and Gihwan, Cho.** A Selective Flooding Method for Propagating Emergency Messages in Vehicle. *Hybrid Information Technology, 2006. ICHIT '06. International Conference.* November, 2006, Vol. 01, 02.

23. **Korkmaz, Gökhan, et al.** Urban multi-hop broadcast protocol for inter-vehicle communication systems. *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks.* 2004.

24. **Uthansakul, P. and Uthansakul, M.** WLAN positioning technique based on measured time delay distribution. *Communications, 2008. APCC 2008. 14th Asia-Pacific Conference.* October, 2008.

25. *A Data Dissemination Strategy for Cooperative Vehicular Systems.* **Brickley, Olivia, et al.** Cork, Ireland : Cork Institute of Technology, 2007, Vol. 2007.

26. **Wikipedia UK.** Polygon - Wikipedia. [Online] Wikipedia, 28 January 2010. [Cited: 01 February 2010.] http://en.wikipedia.org/w/index.php?title=Polygon&oldid=340604886.
27. —. Convex and concave polygons - Wikipedia. [Online] Wikipedia, 04 January 2010. [Cited: 30 January 2010.] http://en.wikipedia.org/w/index.php?title=Convex_and_concave_polygons&oldid=335893688.
28. **Chazelle, Bernard and Dobkin, David P.** Optimal convex decompositions. [book auth.] G T Toussaint. *Computational geometry.* North Holland : Elsevier Science Publishers B.V, 1985. http://www.cs.princeton.edu/~chazelle/pubs/OptimalConvexDecomp.pdf.
29. **Wikipedia UK.** Flooding - Wikipedia. [Online] Wikipedia, 25 January 2010. [Cited: 31 January 2010.] http://en.wikipedia.org/w/index.php?title=Flooding_algorithm&oldid=339926466.
30. **Wattenhofer, Roger.** Geo-Routing. [Online] 28 September 2009. [Cited: 09 March 2010.] http://www.dcg.ethz.ch/lectures/asn/lecture/2/chapter02georouting4.pdf.
31. **Wikipedia UK.** Round-robin scheduling - Wikipedia. [Online] Wikipedia, 20 January 2010. [Cited: 01 February 2010.] http://en.wikipedia.org/w/index.php?title=Round-robin_scheduling&oldid=338862246.
32. **Risan, Øyvind and Peytchev, Evtim.** Status sharing in a vehicle to vehicle context. Nottingham : NTNU / NTU, 2009. Vol. 2009.
33. —. A Vehicle-to-Vehicle communication protocol for collaborative identification of urban traffic conditions. *Accepted ADHOCNETS 2010 (Victoria, BC, Canada).* Nottingham/Victoria : ADHOCNETS 2010 Victoria, 2010. Vol. 2010. http://www.adhocnets.org.
34. **Otken, Phil.** Math Help - Algebra - Linear Functions and Straight Lines. [Online] Technical Tutoring, 09 May 2005. [Cited: 20 January 2010.] http://hyper-ad.com/tutoring/math/algebra/Linear%20Functions%20and%20Straight%20Lines.html.
35. **Wikipedia UK.** Triangle - Wikipedia. [Online] Wikipedia, 30 January 2010. [Cited: 30 January 2010.] http://en.wikipedia.org/w/index.php?title=Triangle&oldid=340938247.
36. **Sun Developer Network (SDN).** Java SE. [Online] Oracle, 09 February 2010. [Cited: 09 February 2010.] http://java.sun.com/javase/downloads/index.jsp.
37. **NetBeans Community.** Netbeans. [Online] Netbeans, 08 February 2010. [Cited: 08 February 2010.] http://netbeans.org/.
38. **Sun Developer Network (SDN).** Java SE API. [Online] Oracle, 31 August 2009. [Cited: 30 January 2010.] http://java.sun.com/javase/7/docs/api/.

39. **RoseIndia.Net.** How to Create Circle In Java. [Online] 05 February 2010. [Cited: 05 February 2010.] http://www.roseindia.net/java/example/java/awt/how-to-create-circle-in-java.shtml.
40. **Red Hat Cygwin Product.** Cygwin information and installation. [Online] 14 June 2008. [Cited: 10 January 2010.] http://cygwin.com/.
41. **The VINT project.** Usedr Information -NsNam. [Online] 25 August 2009. [Cited: 19 January 2010.] http://nsnam.isi.edu/nsnam/index.php?title=User_Information&oldid=4573.
42. **Chih-Heng, Ke.** Windows + Cygwin + myNS2. [Online] 26 October 2006. [Cited: 19 February 2010.] http://hpds.ee.ncku.edu.tw/~smallko/ns2/mysetup_en.htm.
43. **HNS.** Cygwin and ns2 installation. [Online] 03 December 2009. [Cited: 15 February 2010.] http://ns2-hns.blogspot.com/2009_05_01_archive.html.
44. **Castro, Peter.** Cygwin Time Machine. *Cygwin legacy.* [Online] Fruitbat, 02 May 2005. [Cited: 05 February 2010.] ftp://www.fruitbat.org/pub/cygwin/circa/2004/02/08/082010/.
45. **Wikipedia UK.** Tcl - Wikipedia. [Online] Wikipedia UK, 11 March 2010. [Cited: 15 March 2010.] http://en.wikipedia.org/w/index.php?title=Tcl&oldid=349196316.
46. **The VINT Project.** The ns Manual. [Online] 06 January 2009. [Cited: 23 February 2010.] http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf.
47. **Greis, Marc.** Marc Greis' Tutorial for the Network Simulator ns. [Online] 02 December 2000. [Cited: 15 January 2010.] http://www.isi.edu/nsnam/ns/tutorial/index.html.
48. **Yoon, Jisun and Soo, Kim Il.** How to use NS for Wlan. [Online] 15 July 2005. [Cited: 15 January 2010.] http://nislab.bu.edu/sc546/sc546Fall2002/NS80211/course.html.
49. **TraNS.** TraNS (Traffic and Network Simulation Environment). [Online] TraNS. [Cited: 04 May 2010.] http://trans.epfl.ch/.
50. **Wikipedia UK.** IEEE 802.11 - Wikipedia. [Online] Wikipedia, 07 March 2010. [Cited: 15 March 2010.] http://en.wikipedia.org/w/index.php?title=IEEE_802.11&oldid=348281583.
51. **Massey, Ray.** Daily Mail UK Newspaper. [Online] 26 July 2007. [Cited: 30 January 2010.] http://www.dailymail.co.uk/news/article-471022/Increased-congestion-means-average-speed-towns-17-8mph.html.
52. **Wen, Shushan.** Setdets parameters. [Online] 04 January 2006. [Cited: 05 January 2010.] http://winet.ece.ufl.edu/~wen/setdest_para.html.
53. **David, S Moore and George, P McCabe.** Law of Large Numbers. *Introduction to the Practice of Statistics, 4ed.* [Online] W.H.Freeman & Co, 24 March 2004. [Cited: 10 March 2010.] http://bcs.whfreeman.com/ips4e/cat_010/applets/expectedvalue.html.

54. **Wikipedia UK.** Law of large numbers - Wikipedia. [Online] Wikipedia, 08 March 2010. [Cited: 09 March 2010.] http://en.wikipedia.org/w/index.php?title=Law_of_large_numbers&oldid=348476996.

55. —. Monte Carlo method - Wikipedia. [Online] Wikipedia, 07 March 2010. [Cited: 11 March 2010.] http://en.wikipedia.org/w/index.php?title=Monte_Carlo_method&oldid=348304647.

56. **The VINT Project.** Nam: Network Animator. [Online] 04 February 2005. [Cited: 25 April 2010.] http://www.isi.edu/nsnam/nam/index.html.

57. **Pilgrim, Mark.** *Dive Into Python.* s.l. : CreateSpace, 2010. Older edition avalible at: http://diveintopython.org/. 978-1441437136.

58. **Gamati, Emadeddin.** PhD Studentship Regsitration Document, preliminary work. *Research Work Titled: "Intelligent Node Design for Urban Traffic Wireless Mobile Ad-Hoc Networks (MANETs)".* Nottingham : Gamati, Emadeddin, September 2009.

59. **Yildiz, Mehmet E., et al.** Voting models in random networks. *Information Theory and Applications Workshop (ITA), 2010.* La Jolla, CA, USA, 2010, Vol. April, 01.

60. **Li, Yueyue.** PhD Studentship Regsitration Document, preliminary work. *Research Work Titled:" Dynamic Task Oriented Wireless Mobile Network Architectures for Distributed Collaborative Real-Time Information Generation and Control".* Nottingham : Yueyue Li, July 2009.

61. **Weigle, Dr. Michele.** Standards:WAVE / DSRC / 802.11p. [Online] 05 February 2008. [Cited: 19 March 2010.] http://www.cs.odu.edu/~mweigle/courses/cs795-s08/lectures/5c-DSRC.pdf.

62. **Gallardo, J.R, Makrakis, D and Mouftah, H.T.** Performance Analysis of the EDCA Medium Access Mechanism over the Control Channel of an IEEE 802.11p WAVE Vehicular Network. *Communications, 2009. ICC '09. IEEE International Conference.* June, 2009, Vol. Dresden , 01.

63. **Martinez, F.J., et al.** A performance evaluation of warning message dissemination in 802.11p based VANETs. *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference.* October, 2009, Vol. Zurich.

64. **CVISproject.** CVIS Test Sites. [Online] 01 May 2010. [Cited: 01 May 2010.] http://www.cvisproject.org/en/cvis_subprojects/test_sites/.

65. **ipv6.org.** IPv6: The Next Generation Internet! [Online] 06 May 2010. [Cited: 06 May 2010.] http://www.ipv6.org/.

66. **Wikipedia UK.** IPv4 - Wikipedia. [Online] Wikipedia, 03 May 2010. [Cited: 06 May 2010.] http://en.wikipedia.org/w/index.php?title=IPv4&oldid=359930207.

67. —. Cartesian coordinate system - Wikipedia. [Online] Wikipedia, 01 February 2010. [Cited: 03 February 2010.] http://en.wikipedia.org/w/index.php?title=Cartesian_coordinate_system&oldid=341336826.

APPENDIX A Empirical tests for the number of rounds

This appendix gives empirical reasoning behind how many rounds each simulation should contain before the results are expected to be stable and accurate.

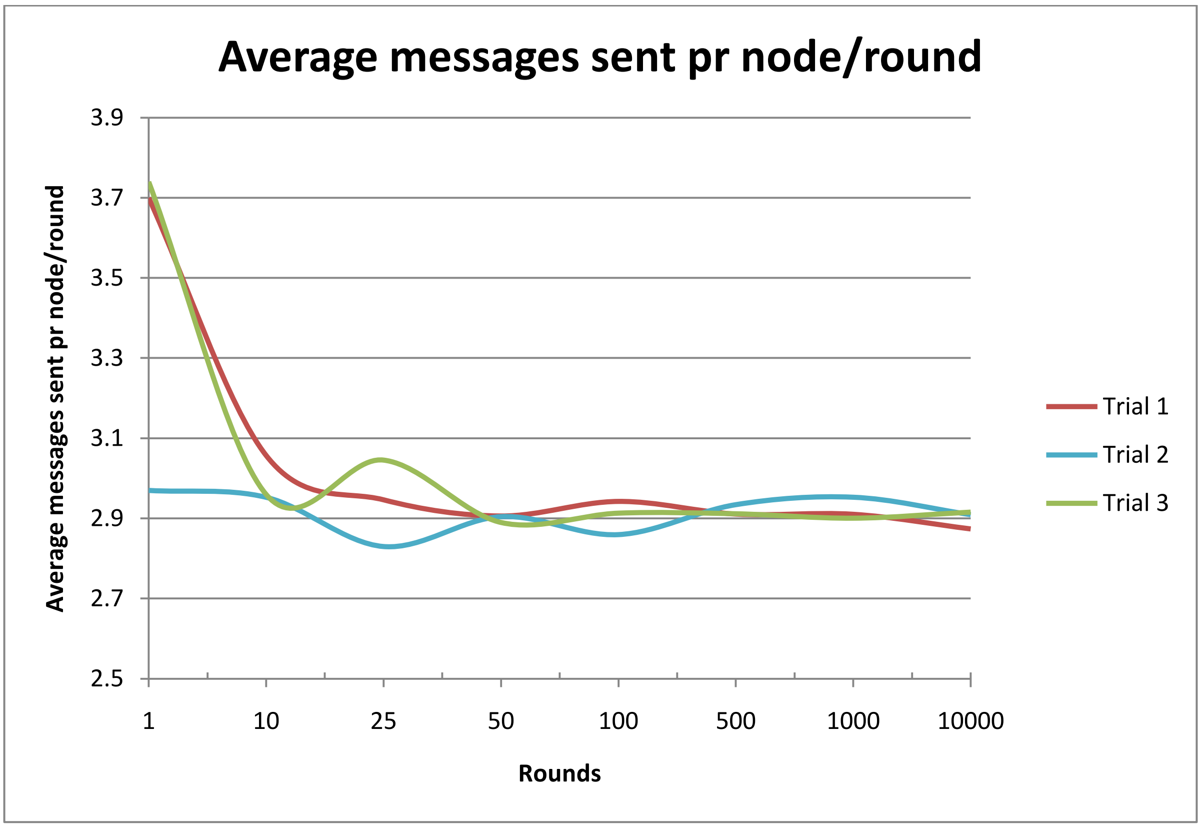
To determine the number of rounds necessary to get a satisfying accurate result the simulations were completed with varying number of rounds three times[29]. Each time the results were gathered. The simulator was set up with 50 vehicles, 10 initial messages, and a random delay time of less than 25 ms. Values were selected to represent working conditions that are both realistic and challenging. The results are presented in Table 6.

	Rounds	**1**	**10**	**25**	**50**	**100**	**500**	**1000**	**10000**
Trial 1	**Sent pr node/round**	3.700	3.057	2.946	2.906	2.942	2.911	2.910	2.874
	Time pr node/round	89.605	66.685	65.931	60.567	61.761	62.264	62.788	61.771
	Result pr node/round	9.600	9.986	9.899	9.980	9.963	9.937	9.893	9.946
Trial 2	**Sent pr node/round**	2.970	2.953	2.830	2.904	2.859	2.934	2.953	2.909
	Time pr node/round	62.517	65.156	61.901	64.661	61.827	63.160	63.933	62.551
	Result pr node/round	10.000	9.618	9.882	9.996	9.888	9.958	9.902	9.907
Trial 3	**Sent pr node/round**	3.740	2.960	3.046	2.890	2.913	2.912	2.901	2.916
	Time pr node/round	85.482	69.744	64.881	61.732	62.813	62.883	62.626	62.804
	Result pr node/round	10.000	9.880	9.802	9.930	9.895	9.915	9.866	9.905

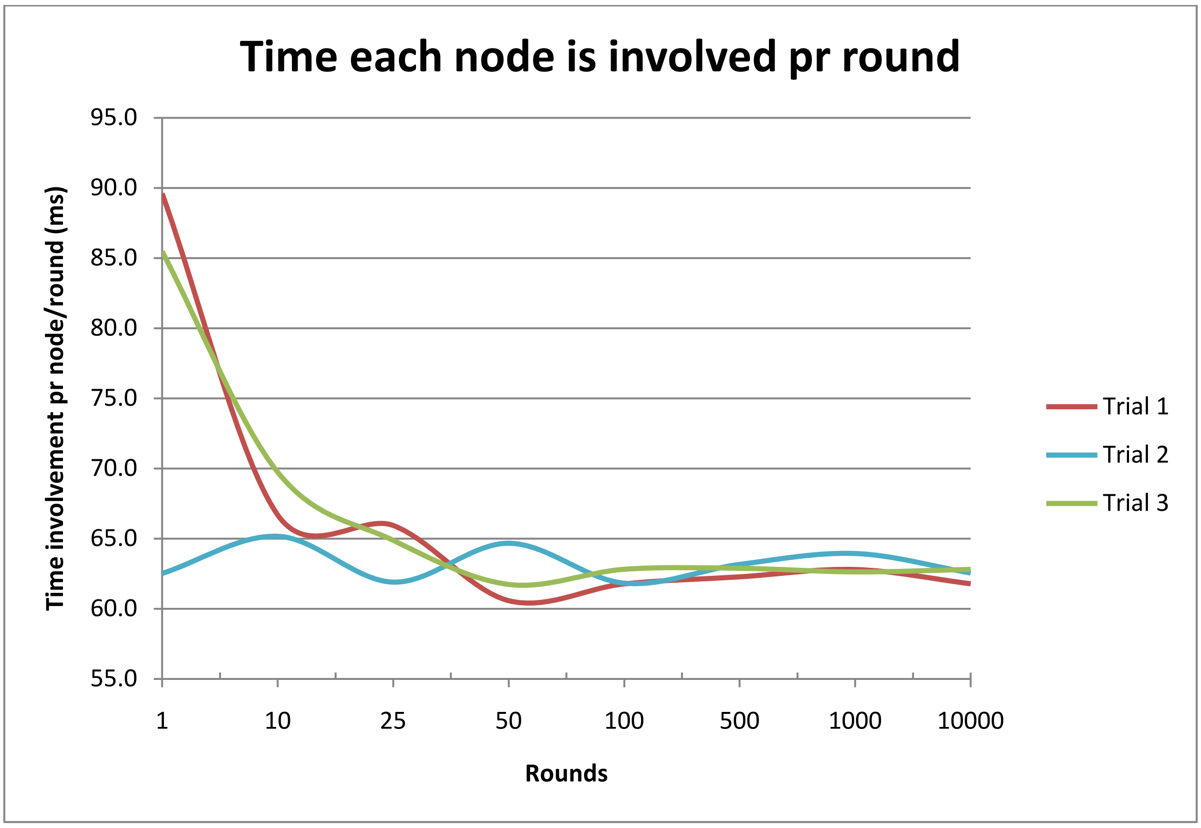
Table 6 Empirical results to determine number of ICE-rounds in the simulator

The result in Table 6 is represented in Graph 11 to Graph 13.

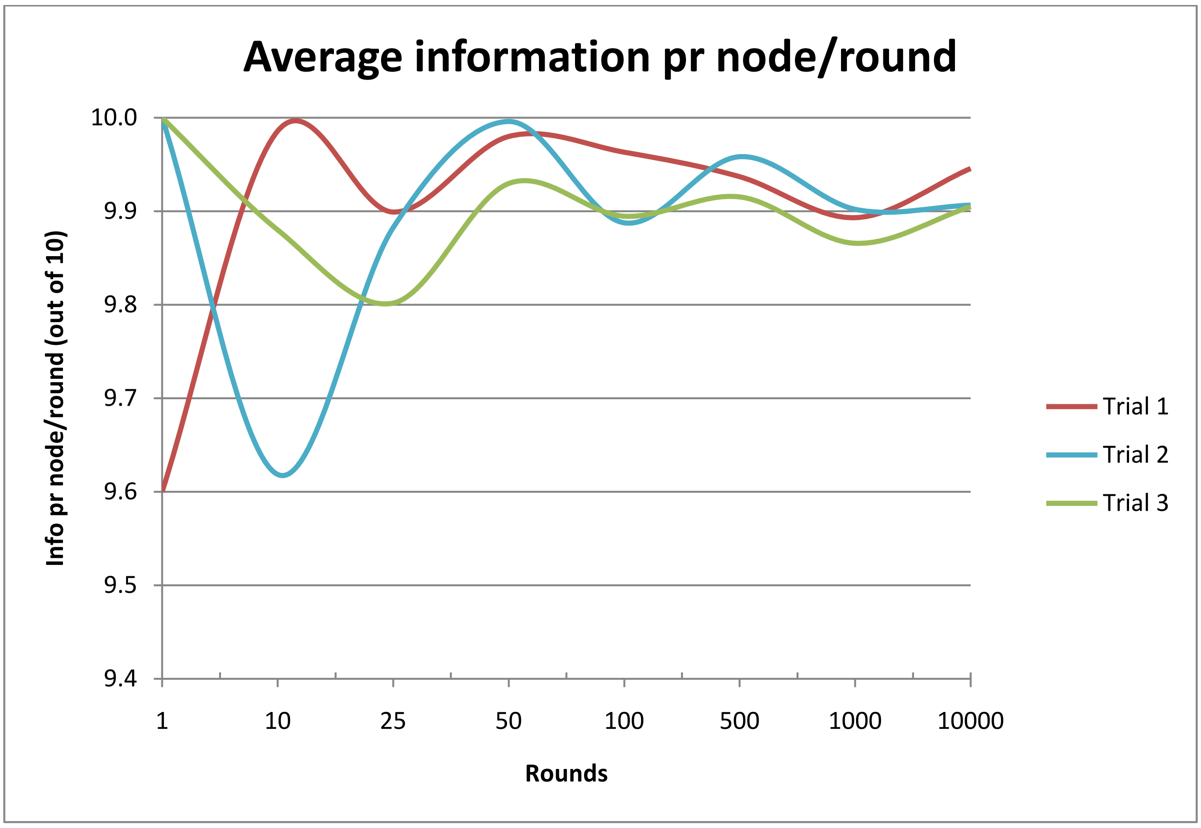
[29] Referred to as Trials in Table 6



Graph 11 Messages beeing transmitted with different number of rounds



Graph 12 Time involvment with different number of rounds



Graph 13 Average information gathered with different number of rounds

Graph 11 to Graph 13 show clear trends. The values seem to stabilize and produce a relatively flat graph when the number of rounds increases beyond 50. Between 1 and 50 rounds there are significant differences between the trials and between different numbers of rounds. The changes in the different trials between 100 rounds and 10.000 rounds are rather small. With 1 round the simulation is very dependent on “luck”, or whether the vehicles are positioned within reach of each other. With 10.000 rounds the vehicles have time to move across the simulation area several times, and by that normalize the probability of being within reach.

The “Monte Carlo method” [55] states that the average value will approach and stabilize at the expected value as the number of samples increase. Graph 11 to Graph 13 support this theory. It is necessary to have at least 50 rounds in the simulator, but the benefits of having much more than 100 rounds is overwhelmed by the amount of computational power needed to complete the simulations.

100 rounds is a sensible choice and should produce results with satisfactory accuracy and stability.

APPENDIX B Constructing cells using Java

In this appendix Code 1 gives the java code needed to illustrate the behaviour of the border finding algorithm as it is presented in Figure 12.

The coordinates of the Java code is a little different from a normal Cartesian coordinate system [67] as it has an inverted Y axis (Y=0 at the top and Y>0 at the bottom). This has an effect on the syntax of the mathematical expressions used to determine the relative position of each element. If element A is placed below element B, then Y_A would be bigger than Y_B.

An executable .jar file is available for testing the performance. The repaint method in Java causes the program to generate new scenarios each time the frame is refreshed. To avoid duplicated circles or lines, the lists are cleared each time the paint method is used. If the frame is resized before it is minimized it will utilize the whole frame when maximized again. New scenarios can be seen with repeated minimization and maximizations.

border_finder.java

```
package border_finder;

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.util.ArrayList;
import java.util.Random;

public class border_finder extends Frame {

        private ArrayList<Shape> circle_list = new ArrayList<Shape>();
        private ArrayList<Shape> border_list = new ArrayList<Shape>();
        private ArrayList<Shape> lowMin_list = new ArrayList<Shape>();
        private ArrayList<Shape> lowMax_list = new ArrayList<Shape>();
        private ArrayList<Shape> upMin_list = new ArrayList<Shape>();
        private ArrayList<Shape> upMax_list = new ArrayList<Shape>();
        private ArrayList<Shape> upper_Half = new ArrayList<Shape>();
        private ArrayList<Shape> lower_Half = new ArrayList<Shape>();

        private Random rand = new Random();
        private int nodes = 20;
        private Shape min0;
        private Shape max0;
        private Shape Min;
        private Shape Max;
        private boolean concave = true;

        private Color blue = new Color(0,0,255);
        private Color pink = new Color(255,20,147);
        private Color green = new Color(127,255,0);
        private Color red = new Color(255,0,0);

@Override
public void paint(Graphics g) {
        Graphics2D ga = (Graphics2D)g;

//  Initialize the nodes, give them a random position, and draw them as circles
        for (int z = 0; z < nodes; z++) {
              circle_list.add(new Ellipse2D.Float(50+rand.nextInt(800), 50+rand.nextInt(800), 10, 10));
              ga.draw(circle_list.get(z));
        }

//  Uses the maxMin method to draw the line between min0 and max0 in pink
        maxMin(ga, circle_list, pink);

//  Stores the result of maxMin as min0 and max0, and adds to the corresponding list
        min0=Min;
        max0=Max;
        upMin_list.add(min0);
        lowMax_list.add(max0);

//  Splits into upper/lower half, and removes unnecessary elements
        upper_Half = split(min0, max0, (ArrayList) circle_list.clone(), true);
        lower_Half = split(min0, max0, (ArrayList) circle_list.clone(), false);

//  Ensures that there the border is continuous in the upper half
        if (upper_Half.size()==0) {
              upper_Half.add(max0);
              upper_Half.add(min0);
        }

//  Finds max and min in the upper half, stores them in lists, split to remove
//  unnecessary elements, continue until there is no more elements in upper half
        while (upper_Half.size()>=1) {
              upper_Half = maxMin(ga, upper_Half, blue);
```

```
                upMin_list.add(Min);
                upMax_list.add(Max);
                upper_Half = split(Min, Max, upper_Half, true);
            }

//  Ensures that there the border is continuous in the lower half
            if (lower_Half.size()==0) {
                lower_Half.add(max0);
                lower_Half.add(min0);
            }

//  Finds max and min in the lower half, stores them in lists, split to remove
//  unnecessary elements, continue until there is no more elements in lower half
            while (lower_Half.size()>=1) {
                lower_Half = maxMin(ga, lower_Half, blue);
                lowMin_list.add(Min);
                lowMax_list.add(Max);
                lower_Half = split(Min, Max, lower_Half, false);
            }

//  Append the max/min elements from upper-half in the right order
    for (int z = 0; z < upMin_list.size(); z++) {
                upper_Half.add(upMin_list.get(z));
            }

            for (int z = upMax_list.size()-1; z >= 0; z--) {
                upper_Half.add(upMax_list.get(z));
            }

//  Append the max/min elements from lower-half in the right order
            for (int z = 0; z < lowMax_list.size(); z++) {
                lower_Half.add(lowMax_list.get(z));
            }

            for (int z = lowMin_list.size()-1; z >=0 ; z--) {
                lower_Half.add(lowMin_list.get(z));
            }

//  Merge and draw the lines between the border elements in red and with -2 offset
            border_list = mergeLists(lower_Half, upper_Half);
            drawBorderLines(ga, border_list, -2, red);

//  Remove concave elements from upper and lower half
            if (concave) {
                concaveRemover(min0, lower_Half, false);
                concaveRemover(max0, upper_Half, true);
            }

//  Merge and draw the lines between the border elements in green and with 0 offset
            border_list = mergeLists(lower_Half, upper_Half );
            drawBorderLines(ga, border_list, 0, green);
}

public void drawBorderLines (Graphics2D ga, ArrayList<Shape> listInput, int offset, Color color){
//  Draw lines between all the elements in a given list, connect last and first as well
//  Uses the offset to shift the lines to avoid lines on top of each other and uses a given colour based on border type
            ArrayList<Shape> lineList = new ArrayList<Shape>();
            ga.setPaint(color);
            for (int z = 0; z < listInput.size(); z++) {
            if (z == (listInput.size()-1)) {

                ga.draw(new Line2D.Double(listInput.get(z).getBounds2D().getCenterX()+offset,
listInput.get(z).getBounds2D().getCenterY()+offset, listInput.get(0).getBounds2D().getCenterX()+offset,
listInput.get(0).getBounds2D().getCenterY()+offset));

            } else {
                lineList.add(new Line2D.Double(listInput.get(z).getBounds2D().getCenterX()+offset,
listInput.get(z).getBounds2D().getCenterY()+offset, listInput.get(z+1).getBounds2D().getCenterX()+offset,
listInput.get(z+1).getBounds2D().getCenterY()+offset));

                ga.draw(new Line2D.Double(listInput.get(z).getBounds2D().getCenterX()+offset,
listInput.get(z).getBounds2D().getCenterY()+offset, listInput.get(z+1).getBounds2D().getCenterX()+offset,
listInput.get(z+1).getBounds2D().getCenterY()+offset));

        }
        }
}

public ArrayList<Shape> split (Shape point1, Shape point2, ArrayList<Shape> list, boolean up){
//  Separates the upper and lower half based upon the boolean value "up"
//  removes all unnecessary elements, including the previous max/min
//  Manages the case where there is just one element left in the upper/lower half
        list.remove(point1);
        list.remove(point2);

        double yTemp1 = point1.getBounds2D().getCenterY();
        double yTemp2 = point2.getBounds2D().getCenterY();
        double xTemp1 = point1.getBounds2D().getCenterX();
        double xTemp2 = point2.getBounds2D().getCenterX();

        for (int z = list.size()-1; z >= 0; z--) {
            double yTemp = list.get(z).getBounds2D().getCenterY();
            double xTemp = list.get(z).getBounds2D().getCenterX();
            double yTarget =yTemp1 + (((yTemp2-yTemp1)/(xTemp2-xTemp1)) * (xTemp-xTemp1));

            if (up){
                if ( (yTemp > yTarget)) {
                    list.remove(z);
                }
            }else{
                if ( (yTemp < yTarget)) {
                    list.remove(z);
                }
            }
        }
        if (up){
            if (list.size() == 1) {
                upMin_list.add(list.get(0));
            }
        }else{
            if (list.size() == 1) {
                lowMax_list.add(list.get(0));
            }
        }
return list;}

public ArrayList<Shape> maxMin(Graphics2D ga, ArrayList<Shape> circleTemp_list, Color color){
//  Finds the max and min in any given list of elements,
//  Draws the lines between these elements in a given colour, and with offset +2
        Min = circleTemp_list.get(0);
        Max = circleTemp_list.get(0);

        for (int z = circleTemp_list.size()-1; z >= 0 ; z--) {
            double xTemp = circleTemp_list.get(z).getBounds2D().getCenterX();
            double xTempMin = Min.getBounds2D().getCenterX();
            double xTempMax = Max.getBounds2D().getCenterX();
            if ( xTemp < xTempMin) {
                Min = circleTemp_list.get(z);
            }
            if ( xTemp > xTempMax) {
                Max = circleTemp_list.get(z);
            }
        }
        ga.setPaint(color);

        ga.draw(new Line2D.Double(Min.getBounds2D().getCenterX()+2, Min.getBounds2D().getCenterY()+2,
Max.getBounds2D().getCenterX()+2, Max.getBounds2D().getCenterY()+2));

return circleTemp_list;}

public ArrayList<Shape> mergeLists (ArrayList<Shape> lowHalf, ArrayList<Shape> upHalf){
//  Combines the two lists and returns this
        ArrayList<Shape> mergedList = new ArrayList<Shape>();
        mergedList.addAll(upHalf);
        mergedList.addAll(lowHalf);
return mergedList;}

public void concaveRemover (Shape value, ArrayList<Shape> node_Half, boolean up){
//  Removes concave elements form upper or lower half based upon boolean "up"
//  Uses the relative distance from a calculated point on the line between the elements neighbours.
        double xTemp1;
        double yTemp1;
        double yTarget;
        boolean ConcaveTest = true;

        while (ConcaveTest) {
            ConcaveTest = false;

            for (int z = node_Half.size()-1; z > 0; z--) {
                if (z == node_Half.size()-1) {
                    xTemp1 = value.getBounds2D().getCenterX();
                    yTemp1 = value.getBounds2D().getCenterY();
                } else{
                    xTemp1 = node_Half.get(z+1).getBounds2D().getCenterX();
                    yTemp1 = node_Half.get(z+1).getBounds2D().getCenterY();
                }
                double xTemp2 = node_Half.get(z).getBounds2D().getCenterX();
                double yTemp2 = node_Half.get(z).getBounds2D().getCenterY();
                double xTemp3 = node_Half.get(z-1).getBounds2D().getCenterX();
                double yTemp3 = node_Half.get(z-1).getBounds2D().getCenterY();

                yTarget = (yTemp1 + (((yTemp3-yTemp1)/(xTemp3-xTemp1)) * (xTemp2-xTemp1)));

                if (((yTemp2 <= yTarget) && !up) || ((yTemp2 >= yTarget) && up)) {
                    node_Half.remove(z);
                    ConcaveTest =true;
                }
            }
        }
}

public static void main(String args[]) {
        Frame frame = new border_finder();
        frame.setSize(900, 900);
        frame.setVisible(true);
        frame.addWindowListener(new WindowAdapter(){
            @Override
            public void windowClosing(WindowEvent we){
                System.exit(0);
            }
        });
}
}
```

Code 1 Java code for testing the border finding method

APPENDIX C Simulator code for ICE

The simulator presented in Code 2 is created to test the functionality of ICE as it is presented in Figure 16. The simulator does not rely on completed “Single ripple” or any other collaborative information gathering scheme. The simulator was used to produce the results for ICE as they are presented in Table 3. The code used to generate the results for “pure flooding” is presented in APPENDIX D.

This simulator can be executed without parameters. If this is the case the simulator will use default parameters embedded in the code. If parameters are defined by the user, they must be presented in the correct order.

P1. Number of nodes that shall be present in the system
P2. Number of unique messages in the system
P3. Number of rounds the simulator shall conduct
P4. Randomized delay time
P5. Binary value specifying choice to reuse movement files or create new

The execution command would be on the format: “*./ns ice_delay.tcl P1 P2 P3 P4 P5*”

At the end of the simulation the results are written to a result file. The following values are recorded as average values from all the independent rounds specified at the beginning of the simulation:

- Average number of sent messages from each vehicle
- Average time each vehicle has been involved
- Average result at each vehicle

ice_delay.tcl

```
# ==================================================================
# Author: Øyvind Risan
# ==================================================================
# Input to "ice_delay.tcl": #Nodes #Active #Rounds DefaultWait opt(random)
#
# Tests the performance of the ICE algorithm as described in Øyvind Risan's thesis of 2010
# Behaviour is to assign (random) a given number of nodes information to spread through the network.
# The input "DefaultWait" is used to delay each transmission a random value smaller than the given value
# Each node represents a vehicle and will gather all information that is presented.
# The nodes move independently through the whole simulation.
# A simulation consist of a number of rounds, this number should be above 50 to produce stable results.
# The gathered information, delays, and other metrics are gathered and written to a result file.

# ==================================================================
# Define Simulation options
# ==================================================================
set optNodes            50                          ;# defines the number of nodes
set optActive           10                          ;# number of active nodes
set optRounds           10                          ;# defines the number of simulator rounds
set opt(interval)       10                          ;# Time between each round (seconds)
set DefaultWait         0.01                        ;# Default delay before sending content
set opt(random)         1                           ;# Decide to make new random patterns
set optNam              0                           ;# Decide if NAM is to be used

# ==================================================================
# Define Environment options
# ==================================================================
set output_name         "ICE_trace"                 ;# defines the name of the output file
set resultFile          "results/ICE_Results.txt"   ;# defines the name of the result file
set opt(sc)             "scene"                     ;# name setting for the movement pattern file
set opt(speed)          10.0                        ;# defines the Max speed of the nodes (m/s)
set opt(pause)          0.10                        ;# defines the pause before the nodes move
set opt(X)              1000                        ;# defines the X-size of the area
set opt(Y)              1000                        ;# defines the Y-size of the area
set opt(n-size)         20                          ;# defines the size of the displayed nodes

# ==================================================================
# Define Operation options
# ==================================================================
set opt(cbr_size)       50000                       ;# constant bit rate
set opt(cbr_interval)   0.002                       ;# constant bit rate interval
set val(chan)           Channel/WirelessChannel     ;# channel type
set val(prop)           Propagation/TwoRayGround;# radio-propagation model
set val(netif)          Phy/WirelessPhy             ;# network interface type
set val(mac)            Mac/802_11                  ;# MAC type
set val(ifq)            Queue/DropTail/PriQueue     ;# interface queue type
set val(ll)             LL                          ;# link layer type
set val(ant)            Antenna/OmniAntenna         ;# antenna model
set val(ifqlen)         50                          ;# max packet in ifq
set val(rp)             DumbAgent                   ;# route protocol (DSDV, AODV, TORA or DSR)

# ==================================================================
# Define Command line input
# ==================================================================
if { $argc > 0 } {
        set optNodes            [lindex $argv 0]
        set optActive           [lindex $argv 1]
        set optRounds           [lindex $argv 2]
        set DefaultWait         [lindex $argv 3]
        set opt(random)         [lindex $argv 4]

# defines the name of the output file
        set output_name    "results/ICE_trace_V$optNodes+_A$optActive+_D$DefaultWait";
        set opt(sc) "sceneV$optNodes+_A$optActive"              ;# defines the name of the movement file
}
```

```
set MESSAGE_PORT 42
set BROADCAST_ADDR -1
set opt(duration) [expr {$optRounds*$opt(interval)}] ;# duration of the simulation

array set RX                    {}
array set TX                    {}
array set FIRST                 {}
array set LAST                  {}
array set INFO                  {}
array set RESULT                {}
array set TIME                  {}
array set node_result           {}

# Initialise ns_
set ns_ [new Simulator]
set tracefd [open $output_name.tr w]
$ns_ trace-all $tracefd

if { $optNam == 1 } {
        set nf [open $output_name.nam w]
        $ns_ namtrace-all-wireless $nf $opt(X) $opt(Y)
}

$ns_ use-newtrace

# ===================================================================
# prepare file for writing only used without support scripts
# ===================================================================
#set fhandle [open ICE_Results w]
#puts $fhandle "----------------------------------------------------------"
#puts $fhandle "Number of rounds:              $optRounds"
#puts $fhandle "Number of nodes:               $optNodes"
#puts $fhandle "Number of active cells:                $optActive"
#puts $fhandle "Default delay before send:     [expr $DefaultWait *1000]\n"
#close $fhandle

# ===================================================================
# Define colour index
# ===================================================================
$ns_ color 0 blue
$ns_ color 1 red
$ns_ color 2 chocolate
$ns_ color 3 red
$ns_ color 4 brown
$ns_ color 5 tan
$ns_ color 6 gold
$ns_ color 7 black

# set up topography object
set topo     [new Topography]
$topo load_flatgrid $opt(X) $opt(Y)

#changes the queue type when running DSR, prevents "Segmentation fault (core dumped)"
if { $val(rp) == "DSR" } {
  set val(ifq)        CMUPriQueue
} else {
  set val(ifq)        Queue/DropTail/PriQueue
}

#creates the God entity
set god_ "God"                          ;# Bypasses the problem with incorrect naming in the setdest file
create-god $optNodes

#generates random movement pattern
if { $opt(random) == "1"} {
     puts "*** NOTE: Random pattern generated."
```

```
        exec ./setdest.exe -n $optNodes -p $opt(pause) -M $opt(speed) -t $opt(duration) -x $opt(X) -y $opt(Y) > $opt(sc)
}

#configures the node template. Agent- and Mac-trace ON, Route- and Movement-trace OFF
$ns_ node-config -adhocRouting $val(rp)          -llType $val(ll) \
    -macType $val(mac)                           -ifqType $val(ifq) \
    -ifqLen $val(ifqlen)                         -antType $val(ant) \
    -propType $val(prop)                         -phyType $val(netif) \
    -channel  [new $val(chan)]                   -topoInstance $topo \
    -agentTrace OFF                              -routerTrace OFF\
    -macTrace ON                                 -movementTrace OFF

#create the nodes
for {set i 0} {$i < $optNodes } {incr i} {
                        set node_($i) [$ns_ node]
                        $node_($i) color "black"

                        set DISCARD($i) 0
                        set RX($i) 0
                        set TX($i) 0
                        set FIRST($i) 0
                        set LAST($i) 0
                        set INFO($i) ""
                        set RESULT($i) ""
                        set TIME($i) ""
                        set node_result($i) 0
}

# Set the node-positions and speed at random, and initialise the nodes
if { $opt(sc) == "" } {
                        puts "*** NOTE: no scenario file specified."
                        set opt(sc) "none"
            } else {
                        source $opt(sc)
                        puts "Random positioning"
            }

for {set i 0} {$i < $optNodes } {incr i} {
            $ns_ initial_node_pos $node_($i) $opt(n-size)
}

# =================================================================
# Subclass Agent/MessagePassing to make it do flooding
# =================================================================
Class Agent/MessagePassing/Flooding -superclass Agent/MessagePassing

# =================================================================
#Procedure for receiving messages
# =================================================================
Agent/MessagePassing/Flooding instproc recv {source sport size data} {
            $self instvar messages_seen node_ Trans_Time Delay

            global ns_ BROADCAST_ADDR RX TX LAST FIRST INFO DefaultWait optActive

            set Node_adr [$node_ node-addr]
            set now [$ns_ now]

            if {($FIRST($Node_adr) == 0)} {
                        set ::FIRST($Node_adr) $now
            }

            set ::LAST($Node_adr) $now
            incr ::RX($Node_adr)

            #
            # Extract information from the message
            #
            set Timestamp       [lindex [split $data ":"] 0]
            set Origin_ID       [lindex [split $data ":"] 1]
            set Information     [lindex [split $data ":"] 2]

            set j 1
            set i 0
            set new_info 0
            set counter 0

            # Detects new information and stores this
            while {$j} {
                        set condition [lindex [split $Information "+"] $i]

                        if {$condition != "!"} {
                                if {([lsearch $INFO($Node_adr) $condition] == -1)} {
                                        lappend ::INFO($Node_adr) $condition
                                        incr counter
                                        incr new_info
                                        #$ns_ trace-annotate "$Node_adr received new {$condition} from $source"
                                }
                        } else {
                                set j 0
                        }
                        incr i
            }

            # Makes the node green when the information is complete (visual aid only)
            if {([llength $INFO($Node_adr)] == $optActive )} {
                                $ns_ at $now "$node_ color green"
            }

            # Schedules the transfer of the complete information table
            if {($new_info > 0) & ($Trans_Time < $now) } {
                        set Trans_Time [expr ($now + {double(rand()*$DefaultWait)})]
                        $ns_ at $Trans_Time "$self send_message 10 $Trans_Time $Node_adr $sport"
            }
}

# ===============================================================
#Procedure for sending messages
# ===============================================================
Agent/MessagePassing/Flooding instproc send_message {size Timestamp Origin_ID port} {
            $self instvar messages_seen node_
            global ns_ MESSAGE_PORT BROADCAST_ADDR TX INFO

            set Node_adr [$node_ node-addr]
            set SendList ""

            # Generating the message that is going to be sent, with the right format
            for {set i 0} {$i < [llength $INFO($Node_adr)]} {incr i} {
                        append SendList [lindex $INFO($Node_adr) $i]+
            }

            append SendList !
            $self sendto $size "$Timestamp:$Origin_ID:$SendList" $BROADCAST_ADDR $port
            incr ::TX($Node_adr)
}
# attach a new Agent/MessagePassing/Flooding to each node on port $MESSAGE_PORT
for {set i 0} {$i < $optNodes} {incr i} {
            set a($i) [new Agent/MessagePassing/Flooding]
            $node_($i) attach  $a($i) $MESSAGE_PORT
             $a($i) set Trans_Time 0
            $a($i) set Delay $DefaultWait
}

#set up the events
set now [$ns_ now]
```

```
for {set j 0} {$j < $optRounds} {incr j} {
            for {set i 0} {$i < $optActive} {incr i} {
                        set start_time [expr {(double(rand()*0.01)+($j*$opt(interval)))}]
                        set random_node [expr {int(rand()*($optNodes-1))}]

                        $ns_ at [expr {$j*$opt(interval)}] "lappend INFO($random_node) [expr $i+($j*100)]"
                        #$ns_ at $start_time "$node_($random_node) color gold"
                        $ns_ at $start_time "$a($random_node) send_message 10 $start_time $random_node
$MESSAGE_PORT"
            }

            $ns_ at [expr (double($j*$opt(interval)+($opt(interval) / 2)))] "clear"
}

# Tell nodes when the simulation ends

for {set i 1} {$i < $optNodes } {incr i} {
            $ns_ at [expr $opt(duration) +10.0] "$node_($i) reset";
}

$ns_ at [expr $opt(duration) +10.0] "print"
$ns_ at [expr $opt(duration) +10.0] "finish"
$ns_ at [expr $opt(duration) +10.01] "puts \"ns_ Exiting...\"; $ns_ halt"

# ======================================================================
# Procedure to clear the information in each node
# ======================================================================
proc clear {} {
            global optNodes node_ INFO RESULT LAST FIRST TIME node_result

            for {set i 0} {$i < $optNodes} {incr i} {
                                    lappend RESULT($i) [llength $INFO($i)]
                                    set ::node_result($i) [expr ($node_result($i) + [llength $INFO($i)])]
                                    lset ::INFO($i) ""

                                    lset ::TIME($i) [expr ($TIME($i) + ($LAST($i) - $FIRST($i))*1000)]
                                    lset ::FIRST($i) 0
                                    lset ::LAST($i) 0

                                    $node_($i) color blue
            }
}

# ======================================================================
# Procedure to print relevant information
# ======================================================================
proc print {} {
            global RX TX RESULT TIME optNodes optRounds node_result DefaultWait optActive resultFile

            set avr_sent 0.0
            set avr_time 0.0
            set avr_result 0.0

            for {set i 0} {$i < $optNodes} {incr i} {
                        set avr_sent [expr ($avr_sent + $TX($i))]
                        set avr_time [expr ($avr_time + $TIME($i))]
                        set avr_result [expr ($avr_result + $node_result($i))]
            }

            puts " \n Number of ICE rounds:          $optRounds"
            puts " Number of nodes:                $optNodes"
            puts " Number of active cells:                     $optActive"
            puts " Default delay before send:      [expr $DefaultWait *1000] ms"
            puts " Average sent pr node/round:     [expr $avr_sent / $optNodes / $optRounds]"
            puts " Average time pr node/round:     [expr $avr_time / $optNodes / $optRounds] ms"
            puts " Average result pr node/round:   [expr $avr_result/$optNodes/$optRounds] out of $optActive\n"
```

```
        set fhandle [open $resultFile a]
                puts $fhandle "Average sent pr node/round:      [expr $avr_sent / $optNodes / $optRounds]"
                puts $fhandle "Average time pr node/round:      [expr $avr_time / $optNodes / $optRounds]"
                puts $fhandle "Average result pr node/round:    [expr $avr_result/$optNodes/$optRounds]\n\n"
        close $fhandle
}

# =======================================================================
# Procedure to clean up and end the simulation
# =======================================================================
proc finish {} {
        global ns_ tracefd nf optNam output_name

        $ns_ flush-trace
        close $tracefd

        if { $optNam == 1 } {
                close $nf
                exec nam $output_name &
        }
exit 0
}
puts "Starting Simulation..."
$ns_ run
```

Code 2 Simulator code for the ICE algorithm

APPENDIX D Simulator code for “pure flooding”

The following code was used to produce the results for “pure flooding” as presented in Table 3. This code is similar to the code presented in APPENDIX C, but as the necessary alterations are scattered throughout the code, the code is presented in its entirety.

The simulator records the same values to a result file, and is executed using the same parameters and formats. The parameter defining the delay before transmission is of course not used simulating this scenario, as this contradicts the purpose of “pure flooding”, but it is included in the syntax to make the execution and the support scripts more generic

ice_flood.tcl

```
# ==================================================================
# Author: Øyvind Risan
# ==================================================================
# Input to " ice_flood.tcl": #Nodes #Active #Rounds DefaultWait opt(random)
#
# Tests the performance of the a pure flooding algorithm as described in Øyvind Risan's thesis of 2010
# Behaviour is to assign (random) a given number of nodes information to spread through the network.
# Each node represents a vehicle and will gather all information that is presented.
# The nodes move independently through the whole simulation.
# A simulation consist of a number of rounds, this number should be above 50 to produce stable results.
# The gathered information, delays, and other metrics are gathered and written to a result file.
# The input “DefaultWait” is not used, it is just kept to comply with the support script

# ======================================================================
# Define Simulation options
# ======================================================================
set optNodes            50                              ;# defines the number of nodes
set optActive           10                              ;# number of active nodes
set optRounds           10                              ;# defines the number of simulator rounds
set opt(interval)       10                              ;# Time between each round (seconds)
set DefaultWait         0.01                            ;# Default delay before sending content
set opt(random)         1                               ;# Decide to make new random patterns
set optNam              0                               ;# Decide if NAM is to be used

# ======================================================================
# Define Environment options
# ======================================================================
set output_name         "output"                        ;# defines the name of the output file
set resultFile          "results/flood_Results.txt"     ;# defines the name of the result file
set opt(sc)             "scene"                         ;# name setting for the movement pattern file
set opt(speed)          10.0                            ;# defines the Max speed of the nodes (m/s)
set opt(pause)          0.10                            ;# defines the pause before the nodes move
set opt(X)              1000                            ;# defines the X-size of the area
set opt(Y)              1000                            ;# defines the Y-size of the area
set opt(n-size)         20                              ;# defines the size of the displayed nodes

# ======================================================================
# Define Operation options
# ======================================================================
set opt(cbr_size)       50000                                   ;# constant bit rate
set opt(cbr_interval)   0.002                                   ;# constant bit rate interval
set val(chan)           Channel/WirelessChannel                 ;# channel type
set val(prop)           Propagation/TwoRayGround                ;# radio-propagation model
set val(netif)          Phy/WirelessPhy                         ;# network interface type
set val(mac)            Mac/802_11                              ;# MAC type
set val(ifq)                Queue/DropTail/PriQueue          ;# interface queue type
set val(ll)                 LL                               ;# link layer type
set val(ant)                Antenna/OmniAntenna              ;# antenna model
set val(ifqlen)             50                               ;# max packet in ifq
set val(rp)                 DumbAgent                        ;# routing protocol (DSDV, AODV, TORA or DSR)

# ======================================================================
# Define Command line input
# ======================================================================
if { $argc > 0 } {
        set optNodes            [lindex $argv 0]
        set optActive           [lindex $argv 1]
        set optRounds           [lindex $argv 2]
        set DefaultWait         [lindex $argv 3]
        set opt(random)         [lindex $argv 4]

        # defines the name of the output file
        set output_name     "results/flood_trace_V$optNodes+_A$optActive+_D$DefaultWait";
        set opt(sc) "sceneV$optNodes+_A$optActive"          ;# defines the name of the movement file
}

set MESSAGE_PORT 42
set BROADCAST_ADDR -1
set opt(duration) [expr {$optRounds*$opt(interval)}] ;# duration of the simulation

array set RX                {}
array set TX                {}
array set FIRST             {}
array set LAST              {}
array set INFO              {}
array set MESSAGE_SEEN      {}
array set RESULT            {}
array set TIME              {}
array set node_result       {}

# Initialise ns_
set ns_ [new Simulator]
set tracefd [open $output_name.tr w]
$ns_ trace-all $tracefd

if { $optNam == 1 } {
        set nf [open $output_name.nam w]
        $ns_ namtrace-all-wireless $nf $opt(X) $opt(Y)
}
$ns_ use-newtrace

# set up topography object
set topo     [new Topography]
$topo load_flatgrid $opt(X) $opt(Y)

#changes the queue type when running DSR, prevents "Segmentation fault (core dumped)"
if { $val(rp) == "DSR" } {
  set val(ifq)         CMUPriQueue
} else {
  set val(ifq)         Queue/DropTail/PriQueue
}

#creates the God entity
set god_ "God"                              ;# Bypasses the problem with incorrect naming in the setdest file
create-god $optNodes

#generates random movement pattern
if { $opt(random) == "1"} {
        puts "*** NOTE: Random pattern generated."
    exec ./setdest.exe -n $optNodes -p $opt(pause) -M $opt(speed) -t $opt(duration) -x $opt(X) -y $opt(Y) > $opt(sc)
}
```

```
#configures the node template. Agent- and Mac-trace ON, Route- and Movement-trace OFF
$ns_ node-config -adhocRouting $val(rp)          -llType $val(ll) \
    -macType $val(mac)                            -ifqType $val(ifq) \
    -ifqLen $val(ifqlen)                          -antType $val(ant) \
    -propType $val(prop)                          -phyType $val(netif) \
    -channel  [new $val(chan)]                    -topoInstance $topo \
    -agentTrace OFF                               -routerTrace OFF\
    -macTrace ON                                  -movementTrace OFF

#create the nodes
for {set i 0} {$i < $optNodes } {incr i} {
                        set node_($i) [$ns_ node]
                        set DISCARD($i) 0
                        set RX($i) 0
                        set TX($i) 0
                        set FIRST($i) 0
                        set LAST($i) 0
                        set INFO($i) ""
                        set MESSAGE_SEEN($i) ""
                        set RESULT($i) ""
                        set TIME($i) ""
                        set node_result($i) 0
}

# Set the node-positions and speed at random, and initialise the nodes
if { $opt(sc) == "" } {
                        puts "*** NOTE: no scenario file specified."
                        set opt(sc) "none"
            } else {
                        source $opt(sc)
                        puts "Random positioning"
            }

for {set i 0} {$i < $optNodes } {incr i} {
            $ns_ initial_node_pos $node_($i) $opt(n-size)
}

# ====================================================================
# Subclass Agent/MessagePassing to make it do flooding
# ====================================================================
Class Agent/MessagePassing/Flooding -superclass Agent/MessagePassing

# ====================================================================
#Procedure for receiving messages
# ====================================================================
Class Agent/MessagePassing/Flooding -superclass Agent/MessagePassing

#Receiving the messages
Agent/MessagePassing/Flooding instproc recv {source sport size data} {
            $self instvar messages_seen node_
            $self instvar Status node_
            global ns_ BROADCAST_ADDR RX TX DISCARD LAST FIRST INFO MESSAGE_SEEN optActive
            set Node_adr [$node_ node-addr]
            set now [$ns_ now]
            if {($FIRST($Node_adr) == 0)} {
                        set ::FIRST($Node_adr) $now
            }

            set ::LAST($Node_adr) $now
            set ::RX($Node_adr) [expr $RX($Node_adr) + 1]

            # extracts information from the message
            set Timestamp       [lindex [split $data ":"] 0]
            set Origin_ID       [lindex [split $data ":"] 1]
            set Msg             [lindex [split $data ":"] 2]

            if {([lsearch $MESSAGE_SEEN($Node_adr) $Timestamp+$Origin_ID+$Msg] == -1)} {
```

```
                        lappend ::MESSAGE_SEEN($Node_adr) $Timestamp+$Origin_ID+$Msg

            # Detects new information and stores this
            if {([lsearch $INFO($Node_adr) $Msg] == -1)} {
                        lappend ::INFO($Node_adr) $Msg
            }
                        $self sendto $size $data $BROADCAST_ADDR $sport
                        set ::TX($Node_adr) [expr $TX($Node_adr) + 1]

            } else {
                        set ::DISCARD($Node_adr) [expr $DISCARD($Node_adr) + 1]
            }
}

# ============================================================================
#Procedure for sending messages
# ============================================================================
Agent/MessagePassing/Flooding instproc send_message {size Timestamp Origin_ID Msg port} {
            $self instvar messages_seen node_
            global ns_ MESSAGE_PORT BROADCAST_ADDR TX MESSAGE_SEEN
            set Node_adr [$node_ node-addr]
            $self sendto $size "$Timestamp:$Origin_ID:$Msg" $BROADCAST_ADDR $port
            incr ::TX($Node_adr)
            lappend ::MESSAGE_SEEN($Node_adr) $Timestamp+$Origin_ID+$Msg
}

# attach a new Agent/MessagePassing/Flooding to each node on port $MESSAGE_PORT
for {set i 0} {$i < $optNodes} {incr i} {
             set a($i) [new Agent/MessagePassing/Flooding]
            $node_($i) attach  $a($i) $MESSAGE_PORT
            $a($i) set messages_seen {}
            $a($i) set Trans_Time 0
            $a($i) set Delay $DefaultWait
}

#set up the events
set now [$ns_ now]

for {set j 0} {$j < $optRounds} {incr j} {
            for {set i 0} {$i < $optActive} {incr i} {
                        set start_time [expr {(double(rand()*0.01)+($j*$opt(interval)))}]
                        set random_node [expr {int(rand()*($optNodes-1))}]
                        $ns_ at [expr {$j*$opt(interval)}] "lappend INFO($random_node) [expr $i+($j*100)]"
                        $ns_ at $start_time "$a($random_node) send_message 10 $start_time $random_node [expr $i+($j*100)]
$MESSAGE_PORT"
            }
            $ns_ at [expr (double($j*$opt(interval)+($opt(interval) / 2)))] "clear"
}

# Tell nodes when the simulation ends
for {set i 1} {$i < $optNodes } {incr i} {
            $ns_ at [expr $opt(duration) +10.0] "$node_($i) reset";
}

$ns_ at [expr $opt(duration) +10.0] "print"
$ns_ at [expr $opt(duration) +10.0] "finish"
$ns_ at [expr $opt(duration) +10.01] "puts \"ns_ Exiting...\"; $ns_ halt"

# ============================================================================
# Procedure to clear the information in each node
# ============================================================================
proc clear {} {
            global optNodes node_ INFO RESULT LAST FIRST TIME node_result
            for {set i 0} {$i < $optNodes} {incr i} {
                                    lappend RESULT($i) [llength $INFO($i)]
                                    set ::node_result($i) [expr ($node_result($i) + [llength $INFO($i)])]
                                    lset ::INFO($i) ""
```

```
                              lset ::MESSAGE_SEEN($i) ""
                              lset ::TIME($i) [expr ($TIME($i) + ($LAST($i) - $FIRST($i))*1000)]
                              lset ::FIRST($i) 0
                              lset ::LAST($i) 0
          }
}

# ==========================================================================
# Procedure to print relevant information
# ==========================================================================
proc print {} {
          global RX TX RESULT TIME optNodes optRounds node_result DefaultWait optActive resultFile
          set avr_sent 0.0
          set avr_time 0.0
          set avr_result 0.0

          for {set i 0} {$i < $optNodes} {incr i} {
                    set avr_sent [expr ($avr_sent + $TX($i))]
                    set avr_time [expr ($avr_time + $TIME($i))]
                    set avr_result [expr ($avr_result + $node_result($i))]
          }

          puts " \n Number of Flood rounds:                    $optRounds"
          puts " Number of nodes:                            $optNodes"
          puts " Number of active cells:                               $optActive"
          puts " Average sent (=result) pr node/round:       [expr $avr_sent / $optNodes / $optRounds]"
          puts " Average time pr node/round:                 [expr $avr_time / $optNodes / $optRounds] ms"

          set fhandle [open $resultFile a]
                    puts $fhandle "Average sent pr node/round:       [expr $avr_sent / $optNodes / $optRounds]"
                    puts $fhandle "Average time pr node/round:       [expr $avr_time / $optNodes / $optRounds]"
                    puts $fhandle "Average result pr node/round:     [expr $avr_result/$optNodes/$optRounds]\n\n"
          close $fhandle
}

# ==========================================================================
# Procedure to clean up and end the simulation
# ==========================================================================
proc finish {} {
          global ns_ tracefd nf optNam output_name
          $ns_ flush-trace
          close $tracefd
          if { $optNam == 1 } {
                    close $nf
                    exec nam $output_name &
          }
exit 0
}
puts "Starting Simulation..."
$ns_ run
```

Code 3 Simulator code for “pure flooding”

APPENDIX E Support scripts using Python

This appendix contains two scripts that were used to support the simulator.

The first script called "Sim_execute.py", presented in Code 4, executes the simulator with given parameters such as:

- Nodes_list = [25, 50, 100, 200]
- Active_list = [10, 20]
- Delay_list = [0.00, 0.005, 0.01, 0.025, 0.05, 0.1]
- Rounds = 100

Any permutation of these parameters is used with both ICE and "pure flooding", and the parameters are written to separate files as each simulation are started. A little vigilance is needed to avoid losing the results if this script is executed repeatedly as the result files are overwritten each time.

The simulator appends the result of each simulation to the same files so that the result will be linked to the corresponding parameters.

Sim_execute.py

```
#!/bin/python
import os
import datetime
filenameICE = "results/ICE_Results.txt"
filenameFlood = "results/Flood_Results.txt"

#Create and initialize the result files
file = open(filenameICE, 'w')
file.write("******ICE Results******\n")
file.write("Simulation started at " + str(datetime.datetime.now()) + "\n")
file.close()

file = open(filenameFlood, 'w')
file.write("******Flood Results******\n")
file.write("Simulation started at " + str(datetime.datetime.now()) + "\n")
file.close()

#Set up the simulation parameters
Nodes_list = [25, 50, 100, 200]
Active_list = [10, 20]
Delay_list = [0.00, 0.005, 0.01, 0.025, 0.05, 0.1]
Rounds = 100

#Iterate through the parameters, and execute the simulators with the corresponding values
for i in range(0, len(Nodes_list)):
        Nodes = Nodes_list[i]


        for k in range(0, len(Active_list)):
```

```
                    Active = Active_list[k]

                    #Determines the production of a new movement file for each active combo
                    Random = 1

                    #Write parameters to the result files
                    file = open(filenameICE, 'a')
                    file.write("--------------------------\n")
                    file.write("Rounds "+ str(Rounds) +"\n")
                    file.write("Nodes "+ str(Nodes) +"\n")
                    file.write("Active "+ str(Active) +"\n")
                    file.write("Delays "+ str(Delay_list) +"\n\n")
                    file.close()

                    file = open(filenameFlood, 'a')
                    file.write("--------------------------\n")
                    file.write("Rounds "+ str(Rounds) +"\n")
                    file.write("Nodes "+ str(Nodes) +"\n")
                    file.write("Active "+ str(Active) +"\n")
                    file.write("Delays "+ str(Delay_list) +"\n\n")
                    file.close()

                    for l in range(0, len(Delay_list)):
                              Delay = Delay_list[l]

                              #prepares and executes the ICE simulator
                              content = "./ns ice_delay.tcl " +str(Nodes)+ " "+str(Active)+ " "+str(Rounds)+"
"+str(Delay)+" "+str(Random)
                              os.system(content)

                              #prevents generating of new movement file for flood
                              Random = 0

                              #prepares and executes the flood simulator
                              content = "ns ice_flood.tcl " +str(Nodes)+ " "+str(Active)+ " "+str(Rounds)+" "+str(Delay)+"
"+str(Random)
                              os.system(content)

#Ends and closes the result files
file = open(filenameICE, 'a')
file.write("Simulation ended at " + str(datetime.datetime.now()) + "\n")
file.write("******ICE Results******")
file.close()

file = open(filenameFlood, 'a')
file.write("Simulation ended at " + str(datetime.datetime.now()) + "\n")
file.write("******Flood Results******")
file.close()
```

Code 4 Automated execution of the simulations

The second script called "Tracefile_iterator.py", presented in Code 5, is useful for extracting the results from the trace files. As the trace files increase exponentially in size as more vehicles are involved, the size and number of lines grow too big to handle with conventional software.

The script iterates through the trace file and counts the number of sent (s), received (r) and dropped (d) messages one line at a time. Each line in the trace file represents one action on one message. The details regarding each action are discarded as it is of no interest in this simulation.

"Pure flooding" with 200 nodes ≈ 25 millions messages transactions during a 100 round simulation, and this gives a trace file in excess of 3.4 GB.

Tracefile_iterator.py

```
#!/bin/python
import os
import datetime
import fileinput
filenameResult = "drop_result.txt"

#Create and initialize the result file
file = open(filenameResult, 'w')
file.write("******Tracefile iterator ******\n")
file.write("Started at " + str(datetime.datetime.now()) + "\n\n")
file.close()

#Determine the tracefiles to iterate through
Tracefiles= ["file1.tr", "file2.tr", "file3.tr"]

#Initialize counters
dr = 0
rx = 0
tx = 0

#Iterate through the tracefiles
for l in range(0, len(Tracefiles)):
        trace = Tracefiles[l]

        for line in fileinput.input(["results/"+trace]):
                char = line[0]          # read first character in line

                #Check what type of information the line represents
                if char == "d":
                        dr = dr +1
                if char == "r":
                        rx = rx +1
                if char == "s":
                        tx = tx +1

        #Write results to the result file
file = open(filenameResult, 'a')
        file.write("\n")
        file.write(str(dr) +"           is dropped by:      "+str(trace)+"\n")
        file.write(str(rx) +"           is received by:     "+str(trace)+"\n")
        file.write(str(tx) +"           is sent by:         "+str(trace)+"\n")
        file.close()

#Print result on screen
        print str(dr) +"                is dropped by:      "+str(trace)
        print str(rx) +"                is received by:     "+str(trace)
        print str(tx) +"                is sent by:         "+str(trace)

#reset counter for the next tracefile
        dr = 0
        rx = 0
        tx = 0

##Ends and closes the result files
file = open(filenameResult, 'a')
file.write("\n Ended at " + str(datetime.datetime.now()) + "\n")
file.write("******Tracefile iterator******")
file.close()
```

Code 5 Extraction of data for sent, received and dropped messages