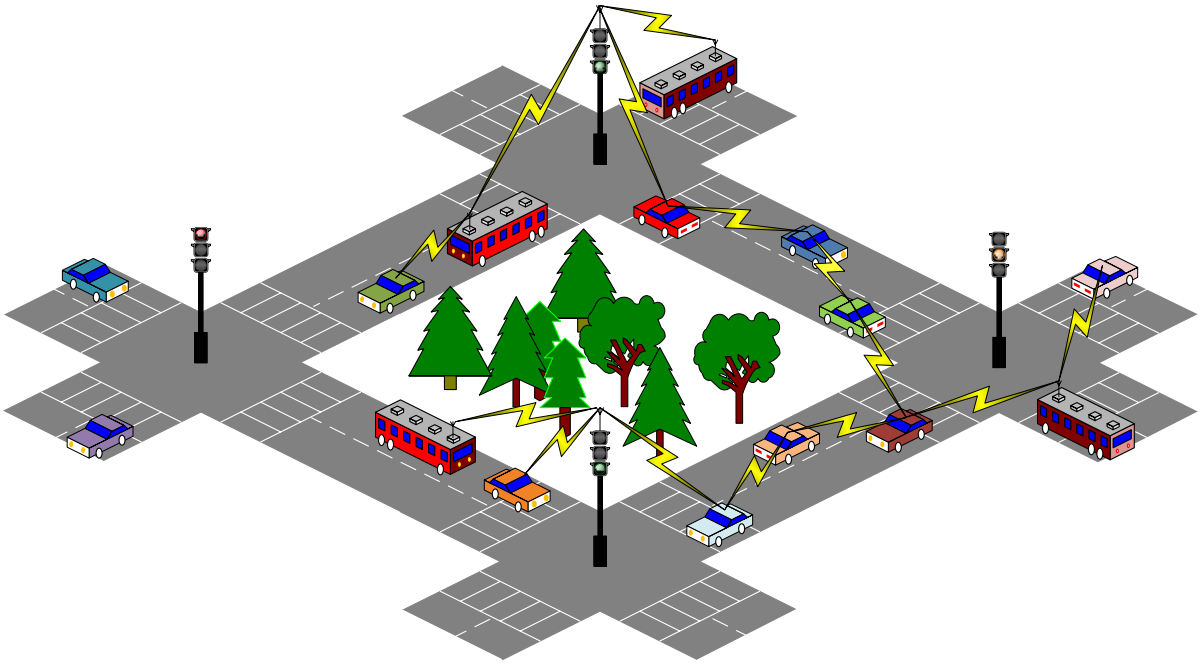
NTNU
Norwegian University of Science and Technology

NOTTINGHAM TRENT UNIVERSITY

Status sharing in a vehicle to vehicle context

Øyvind Risan

Autumn 2009



Dr. Evtim Peytchev

Professor Dr. Steinar Andresen

PhD Yueyue Li

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY, MATHEMATICS AND ELECTRICAL ENGINEERING

PROJECT ASSIGNMENT

Student's name: Øyvind Risan

Course: TTM4511

Title: Status sharing in a vehicle to vehicle context

Description:

Vehicles mounted with communication equipment can be configured to communicate their internal status to other vehicles. This status might be information about ABS, engine parameters, GPS location, airbag state, and the status of other relevant functions.

A simulator can be used to simulate different scenarios, such as different distance between the vehicles, different amounts of data traffic, and the need for a permanent infrastructure.

Establish a context that makes it possible to standardize messages, cut down on the overall data load, and speed up the transmissions.

This project aims to:

- Study the effect of status sharing between different vehicles in a moving environment.
- Study the migration of data through the connected nodes.
- Constructing a useful context for the messages that might be necessary.
- Study how to gather information from the network.

Deadline: 18/12-2009
Submission date: 17/12-2009
Department: Department of Telematics
Supervisor: Professor Steinar Andresen & Dr. Evtim Peytchev

Trondheim, 28/9-2009

Steinar Andresen
Professor

FOREWORD

This project is a part of the “final year assignments” for my master degree within communications technology.

The project has been conducted during the autumn of 2009, from September to December. 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, has been a valuable contribution to my work. The weekly meetings with Peytchev and his PhD’s has helped me keep progression and kept me motivated right from the start.

I’m very grateful for all the help and support I have gotten from Professor Steinar Andresen at “Norwegian University of Science and Technology”. Without his help I would never have gotten such an interesting assignment and opportunities.

A special thanks to Yueyue Li for her help and assistance when I found myself too deep in simulator problems.

I need to thank my fiancée for supporting me, even though I spent most of the time in front of the computer, completely oblivious to my surroundings.

All the material used to write this report is available electronically. This includes the document, simulator codes, simulator trace files, figures and tables.

CONTENT

PROJECT ASSIGNMENT ..... **I**

FOREWORD ..... **I**

CONTENT ..... **II**

FIGURE INDEX ..... **IV**

TABLE INDEX ..... **IV**

CODE INDEX ..... **V**

GRAPH INDEX ..... **V**

ACRONYMS ..... **VI**

SUMMARY ..... **VII**

1. INTRODUCTION **1**

1.1. BACKGROUND 1
Communication *1*
Status maintenance *2*
Infrastructure *2*
Information exchange *2*
Area discovery *2*
1.2. PROBLEM DESCRIPTION 2
1.3. LIMITATIONS 3
1.4. STRUCTURE 4

2. THEORY **5**

2.1. GENERAL THEORY ABOUT THE SYSTEM 5
2.2. STATUS MAINTENANCE 6
2.2.1. Keeping information alive *6*
2.2.2. Keeping the information consistent *7*
2.3. INFRASTRUCTURE 8
2.4. INFORMATION EXCHANGE 9
2.5. AREA DISCOVERY 11

3. IMPLEMENTATION **14**

3.1. STATUS MAINTENANCE 14
3.1.1. Keeping information alive *14*
3.1.2. Keeping the information consistent *15*
3.2. INFRASTRUCTURE 18
3.3. INFORMATION EXCHANGE 20
Push-message *20*
Pull-message *21*
3.4. AREA DISCOVERY 23

4. SIMULATION 29

4.1. INSTALLATION AND PROGRAMMING 29
4.2. THE EXPERIMENT 29
4.3. THE CODE 31
4.4. RESULTS 33

5. DISCUSSION 36

6. CONCLUSION 43

REFERENCES 45

APPENDIX A: DUAL AREA DISCOVERY 47

APPENDIX B: SIMULATOR CODE FOR 16 VEHICLES 52

APPENDIX C: REPLACEMENT CODE 59

FIGURE INDEX

FIGURE 1 FLOODING TO NEARBY NODES, N=10 6
FIGURE 2 VEHICLES AND INFRASTRUCTURE WORKING TOGETHER 9
FIGURE 3 STRUCTURE OF A PUSH-MESSAGE 10
FIGURE 4 STRUCTURE OF A PULL-MESSAGE 11
FIGURE 5 VEHICLES DETERMINING THE SIZE OF A RAINCLOUD 13
FIGURE 6 FLOWCHART FOR THE UPDATE PROCEDURE 15
FIGURE 7 DISCOVERY AND MAYORITY VOTE FOR "NEW" VEHICLES 17
FIGURE 8 BUS AS MOVABLE INFORMATION STORAGE AND SOURCE 19
FIGURE 9 TRANSMISSION OF A PUSH-MESSAGE 20
FIGURE 10 BASIC ILLUSTRATION OF A PULL-MESSAGE TRANSMISSION 22
FIGURE 11 ALGORITHM FOR DISCOVERING AREAS 24
FIGURE 12 EXAMPLE OF TWO CONDITION SCENARIO 25
FIGURE 13 EXAMPLE OF A PULL-MESSAGE FLOW 26
FIGURE 14 FIELDS USED TO IDENTIFY MESSAGES UNIQUELY 28
FIGURE 15 SCENARIO FOR THE SIMULATOR 30
FIGURE 16 SCREENSHOT FROM THE SIMULATOR WITH 4 VEHICLES 32
FIGURE 17 TWO AREA DISCOVERY, LARGE EXAMPLE 47

TABLE INDEX

TABLE 1 UPDATE INTERVALLS FOR DIFFRENT PUSH-MESSAGES 16
TABLE 2 BINARY AND HEX REPRESENTATION OF VEHICLE CONDITIONS 21
TABLE 3 DIRECTION ENCODING 22
TABLE 4 RESULTS FROM THE SIMULATOR WITH 16, 12, 8 AND 4 VEHICLES 34
TABLE 5 TOTAL NUMBER OF SENT, RECEIVED AND DROPPED PACKETS 35
TABLE 6 ROUTING TABLE FOR TWO AREA DISCOVERY EXAMPLE 47
TABLE 7 MESSAGES SENT AND RECEIVED BY VEHICLE 0 49
TABLE 8 MESSAGES SENT AND RECEIVED BY VEHICLE 1 49
TABLE 9 MESSAGES SENT AND RECEIVED BY VEHICLE 2 50
TABLE 10 MESSAGES SENT AND RECEIVED BY VEHICLE 3 50
TABLE 11 MESSAGES SENT AND RECEIVED BY VEHICLE 4 50
TABLE 12 MESSAGES SENT AND RECEIVED BY VEHICLE 5 51
TABLE 13 MESSAGES SENT AND RECEIVED BY VEHICLE 6 51
TABLE 14 MESSAGES SENT AND RECEIVED BY VEHICLE 7 51

CODE INDEX

Code 1 Pseudo-code for discovering an area 27
Code 2 Simulator code for "intelligent flooding" with 16 vehicles 58
Code 3 Replacement code for "pure flooding" 60

GRAPH INDEX

Graph 1 Complete presentation of sent, received and dropped packets 36
Graph 2 Received packets using both algorithms 37
Graph 3 Sent packets using both algorithms 38
Graph 4 Average time a vehicle is involved using both algorithms 39
Graph 5 Dropped packets using both algorithms 40
Graph 6 Dropped packets out of the total number of packets 41

ACRONYMS

ABS	Anti-lock Braking System
Ad-Hoc	Latin; "for this purpose", decentralized wireless network.
CALM	Continuous Air interface for Long and Medium distance
CVIS	Cooperative Vehicle- Infrastructure Systems
GPS	Global Positioning System
HSPA	High Speed Packet Access
Intelligent flooding	Algorithm for discovering areas by as few messages as possible.
MAC	Media Access Control address uniquely identifies most network equipment.
NS2	Network Simulator, based on C++, through Otcl.
NTNU	Norwegian University of Science and Technology
NTU	Nottingham Trent University
Otcl	Object-oriented Tool Command Language
POI	Point of Interest, A specific point represented by coordinates or other unique identifier.
Pure flooding	Algorithm that sends any new message to any receiver within reach.
TCP	Transmission Control Protocol, counterpart of UDP, includes retransmission and flow control.
UDP	User Datagram Protocol, message from a sender that doesn't expect a reply.
UMTS	Universal Mobile Telecommunications System, 3G mobile telecommunications.
VANET	Vehicular Ad-Hoc Network, provide communications among nearby vehicles.
VIN	Vehicle Identification Number uniquely identifies any vehicle.
WiMAX	WiMAX, meaning Worldwide Interoperability for Microwave Access
WLAN	Wireless local area network

SUMMARY

Vehicle to vehicle communication makes it possible to share status information by using wireless technology. The nature of such a network demands self configuration and autonomous behaviour.

A vehicle to vehicle network that is reliant on infrastructure will not be as flexible and agile as a network that uses whatever recourses available. Such a network would be able to take advantage of any present infrastructure, but would not depend upon it. Infrastructure might be used to solve problems related to coverage, and it can be used to supply information and services. I have also discussed how buses, or trams, might be used as movable infrastructure and data stores. The benefit from this is the predictability of route and the large area they cover.

Housekeeping is best done by letting messages time out after a specified number of seconds or minutes, or by limiting how far the information travels by imposing limitations on how far from the source the message is of interest. At distances greater than this limit the message is discarded, and thus not forwarded further.

Updating the status within the network is an important task, this is necessary to get new vehicles up to date and to cover holes left by lost or damaged messages. As this network doesn't have a master the best method of update would be to inform your neighbour vehicles about your interpretation about the status in the network. A majority vote on any received update messages would ensure that the system converges towards a common status that is as close to reality as possible.

I have identified the need for two types of messages in the vehicle to vehicle communication. One message will inform, or warn, other vehicles about special points of interests on the road. Examples of this might be Ice, or a broken down car on a steep curve in the road. Another type of message will be used to extract information from other vehicles in the network.

This information can be used in several different services. This report focuses on one such service: Area discovery.

By gathering information from vehicles it is possible to determine the area in which a condition applies. My “intelligent flooding” algorithm will perform this task involving only vehicles with the desired conditions, and their neighbours. This ensures a minimal load on the network.

When “intelligent flooding” is tested against a standard “pure flooding” algorithm there is clear benefits when it comes to the number of sent messages, and the number of unnecessary messages. The amount of sent messages can be reduced with up to 89% compared to “pure flooding”. The load on each vehicle is also reduced as the number of received messages is reduced by 91%.

“Intelligent flooding” is more susceptible to interference, and loss of packets, than “pure flooding”. “Pure flooding” is more robust, but also more resource consuming

The main conclusion to be derived from this report is:

- If there are no limitations on the recourses in the network the most robust and flexible solution is to use “pure flooding”.
- If there are limitations regarding number of messages, delay, computational power and network load, then “intelligent flooding” is the best solution.

There are several ongoing projects researching vehicle to vehicle communication, and their results will provide new services and platforms, useful for future work. “Car 2 Car Consortium” and CVIS are examples of such projects.

1. INTRODUCTION

1.1. Background

The number of objects that contains some sort of a processor is increasing. Some of this is due to reductions in: costs, size, weight, and power consumption. Another reason is that the consumers enjoy being surrounded by intelligent objects.

The amount of information that can be gathered from these objects is staggering, and the amount will keep on increasing in the future. The trend of today is to equip objects with the capability to communicate, and interact, with their surroundings. Some of the information that earlier was limited to use within one vehicle is suddenly available for use in other vehicles. By gathering the available information it is possible to provide new services and functionality.

A modern car has several hundred sensors that are used to keep the car working. By equipping the car with communications technology it will no longer be an isolated node but becomes a part of a bigger system. Exchange of information between cars might prevent accidents, and increase safety & efficiency.

Some countries have introduced a so called “Zero-Road-Fatality-Vision” or similar policy. The aim of these policies is to reduce the number of road related fatalities to a minimum. The implementation of technology for vehicle to vehicle communication can provide a valuable contribution in reaching this vision.

If vehicles are going to be able to exchange information there are some aspects that need to be addressed:

Communication

How are the vehicles going to communicate? Within the field of wireless communication there are numerous standards and solutions (WLAN IEEE 802.11a/b/g/p, UMTS, WiMAX IEEE 802.16, HSPA, etc.), each with its own benefits and drawbacks. The choice of technology will impact the design, and function, of the network. Coverage and throughput will be important factors.

Status maintenance

Is it possible to maintain a fairly updated common status within the moving vehicles? The vehicles must be able to maintain the information, and keep it up to date. Information that is no longer valid must be discarded, and new information must be spread to all the relevant vehicles.

Infrastructure

What are the demands for infrastructure? To find a sustainable solution flexibility and robustness must be considered, along with the capability of providing service to the users.

Information exchange

How can we make the communication between the vehicles fast and reliable, without sacrificing the information quality? The vehicles need to communicate by using some kind of standard or language. Message size and content must be considered as part of this.

Area discovery

Is it possible to discover the size of an area of interest? This involves gathering information from the network in the most effective way possible, without wasting recourses. Determination of an area based on some condition(s), or criteria, might be valuable to the system.

This report will try to answer some of these aspects of vehicle to vehicle interaction.

1.2. Problem description

Consider a flock of pigeons. Their path in the sky is directed by the amount, and positions, of food and predators. To be able to move as a flock they have work together, and exchange information between the individual pigeons. The first pigeon to spot a hawk will alert the others and the flock will change direction. The same happens when some of them discover food.

A collection of vehicles can behave in similar ways. Some of the vehicles might possess information that is of interest to the others.

Two example of this is the use of ABS, and rain. If the vehicle in front of you uses the ABS, the probability that you will also need to use ABS increases significantly. If it is raining, how big is the cloud? This is information you don't have, but it is information that can be gathered by collecting available data.

There is a significant difference between these two examples. One of them is based on messages being "pushed" (before you need the information), and the latter is a "pull" (someone has requested the information).

This all comes down to some questions that need to be answered.

- Can a vehicle to vehicle network, that is both robust and autonomous, be created?
- Is it possible to maintain a fairly updated common status within moving vehicles?
- Is it possible to do this without a centralized server?
- How should the necessary messages be constructed?
- What impact will infrastructure have on the network?
- How can one use the available data to discover an area?
- Which algorithms might be used to control the number of messages?
- Is it possible to make a simulator that can prove the concept behind area discovery?

1.3. Limitations

The aspect "Communication" from chapter 1.1 will not be considered in this report. I will assume that the vehicles are equipped with all the necessary communication equipment, and that they are able to communicate with each other, or similar equipment.

I also assume that the vehicle's sensors are connected to the communications equipment. The interfaces and standards that is necessary for this are beyond the scope of this project.

By considering the communication as a black box I detach the communication from the information exchange, and from the vehicle itself. This will make it possible to apply the results of this report to other systems, without being bound by one special technology, or architecture such as CVIS, CALM and VANET.

I will not lay any restrictions on what format is used for positioning. I have used (x/y)-coordinates, as this is the most natural in the simulator. In real-life vehicles one of the GPS-formats may be used instead.

I will not spend any time on the security aspects that are linked to such a network. All the communication will be considered “friendly” and all the messages are genuine.

The aspects “Status maintenance”, “Infrastructure” and “Information exchange” will be covered, but not to their full depths. This is to limit the scope of this report. The part of this report that is the most detailed is the work on “Area discovery”.

1.4. Structure

This report has 3 main parts. The first part is regarding theory, where there will be a theoretical analysis of the aspects presented in chapter 1.1. The second part is devoted to issues regarding implementation. Here I will set up some possible models regarding the different aspects, with special emphasis on “Area discovery”. The last part is the Simulator-part where I will test the solutions that have been presented in the earlier chapters. This part also presents the results that have been gathered through the simulator.

After the last main part there is the Discussion, where I will analyse the results before I, in the Conclusion, try to answer the questions presented in this chapter.

All simulation code, and special material, will be placed in the appendix, but simpler parts will be presented in the main text.

2. THEORY

2.1. General theory about the system

Research in vehicle to vehicle networking is a very large field, and there are a lot of projects underway in Europe. The Car 2 Car Consortium [1] is working on creating a common European standard for this communication. Of special interest is the CVIS project [2]. The aim of CVIS is to enable communication across multiple technologies and standards. The number of test sites for CVIS is growing. In September 2009 Test Site Norway became an associated part of the CVIS project. This test site has 14 street stations and 4 rooftop stations. The backbone of CVIS is CALM [3], which is an architecture that allows the simultaneous use of multiple channels. VANET [4] or Intelligent VANET[1] [5] is similar to CVIS, but the contributions within these fields are modest compared to the European CVIS projects.

Common for all these projects is that vehicles communicate in a mobile ad-hoc network, often with multiple hops. Such a network can be used to improve road safety by supplying early warnings. It is also possible to get information about weather conditions and other up-to-date traffic information. The goal is to make a ubiquitous network.

The fact that the sender and receiver are placed in a vehicle that can reach speeds far beyond any pedestrian presents some challenges. At low speeds there are few location changes per second, but as speed increases the number of location changes per second rises dramatically [6]. This fact commands that the system must be robust and self configuring.

A system in this report is a changing thing. A system might consist of just two vehicles, and cover a very small geographical area. The system might also consist of hundreds of vehicles and cover a huge area. The density of vehicles and the location of each vehicle will give the size of the covered area. Due to limitations in transition power, and signal propagation, there will be a limit as to how far two adjacent vehicles can be apart before the system must be considered as two autonomous systems, or sub-systems.

[1] Referred to as InVANET

The system will only exist as long as there are vehicles that have information to exchange, and as long as there are vehicles available to keep the information alive. If there are no vehicles in the system, there is no information of interest to be exchanged, and the need for the system disappears [7].

2.2. Status maintenance

2.2.1. Keeping information alive

Information must migrate and be kept alive. This is achieved by retransmission. A message that is received and retransmitted gains two things. It survives a period in time, and it might migrate to new receivers. A retransmission can be done in two ways, either retransmit to specific addresses, or retransmit to everyone. The latter method of communication will later in the report be referred to as “pure flooding”.

Transmission to specific addresses is most effective if the environment and the receivers are static, or motionless. Once the discovering, and determination, of neighbours is finished each message needs only be sent once, which saves capacity and time. This solution might demand special hardware such as directive antennas and controllers.

Flooding is like opening a door with a sledgehammer instead of a key. It is big, noisy and brutal, but it gets the job done. The biggest problem with flooding is the sheer number of messages that are being received, when each node transmits all new messages to any receiver within reach [8].

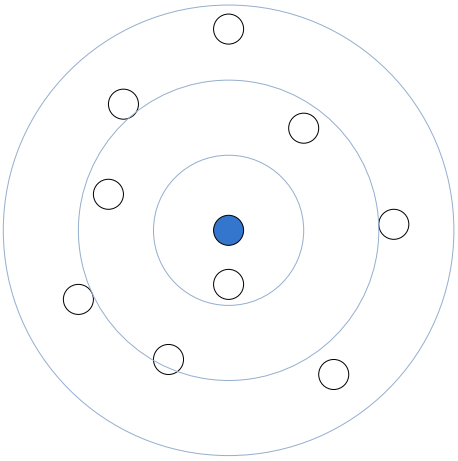


Figure 1 Flooding to nearby nodes, n=10

If there are n receivers within reach[2] of each other, the message will be received n*(n-1) times. As seen in Figure 1 this will cause the sender to receive 9 messages that are of no value. If all the other receivers can reach 9 nodes there are 90 messages that are wasted. This will unfortunately introduce interference, and possible loss of messages.

There is a small benefit to the messages being sent so many times. If the message type is UDP, the presence of several identical messages increases the probability that every receiver actually gets the message.

An update-scheme is also necessary. Special rules can be applied to the different types of information, and thereby give a satisfactory update procedure. The problem that presents itself is the fact that we do not have a master in the network. Then who is going to be responsible for updating the information in the system? With no master the responsibility lies within each vehicle. An algorithm for this will be presented in chapter 3.1.1

2.2.2. Keeping the information consistent

The mechanisms used to keep the messages alive might cause some problems with keeping the information consistent and updated. Retransmissions and replication might produce duplicate messages. This introduces the need for telling the messages apart.

The information in the system shall depict the real environment as truthful as possible. As time passes the conditions once observed in the environment might have changed, moved or disappeared. These challenges will be addressed in chapter 3.1.2.

A major issue is that there might be N^3 different interpretations about what the “truth” is. Each vehicle, or node, in the system might have a slightly different perception of what is the overall status in the network.

A bigger problem is the arrival of new vehicles. How can they get an impression about what the status is?

[2] Including the sender itself
[3] Finite integer lager than 0, representing the total number of vehicles in the system

The vehicles that is most likely to have the most complete picture of what the status is in the network is probably the vehicles that are closest to the centre of the network. These vehicles have the highest probability of having received all messages that have migrated trough the network.

A new vehicle will have difficulties discovering these vehicles, so an easier solution might be to rely on the nearest vehicle instead. Due to the many interpretations about the status, it is not enough to ask just one neighbour. A majority vote between any vehicles within reach would provide a fairly good estimate about what the status is at this part of the network. This is definitely a “best-effort” task. An algorithm for this is given in chapter 3.1.2.

2.3. Infrastructure

The goal is to produce a robust, self configuring and autonomous network. If the network is reliant of infrastructure the network might be limited to operations in areas where the infrastructure is in place.

It is better for the network to be independent of existing infrastructure, but use whatever resources that are available. A lamppost equipped with the same communications equipment as the vehicles might be regarded as a very slow moving vehicle (as seen from the network). This permanent equipment can be used to supply the network with important information from “the outside” (i.e. Traffic information or congestion warnings) and might gather information for some centralized service (statistical information about troublesome areas).

Permanent infrastructure might be helpful to cover “black holes” or difficult positions (i.e. can cover both sides of a difficult corner). The directivity and mounting height of the antenna plays a major part in the connectivity [9]. On a vehicle the most natural choice is an Omni-directional antenna. The mounting height of this is limited by the vehicle and aesthetic concerns. On a permanent structure it is possible to use very different equipment depending on the location, and needs, of the area. Figure 2 illustrates what this might look like in an urban environment. The light-signals communicate with buses and cars, and the vehicles communicate with each other.

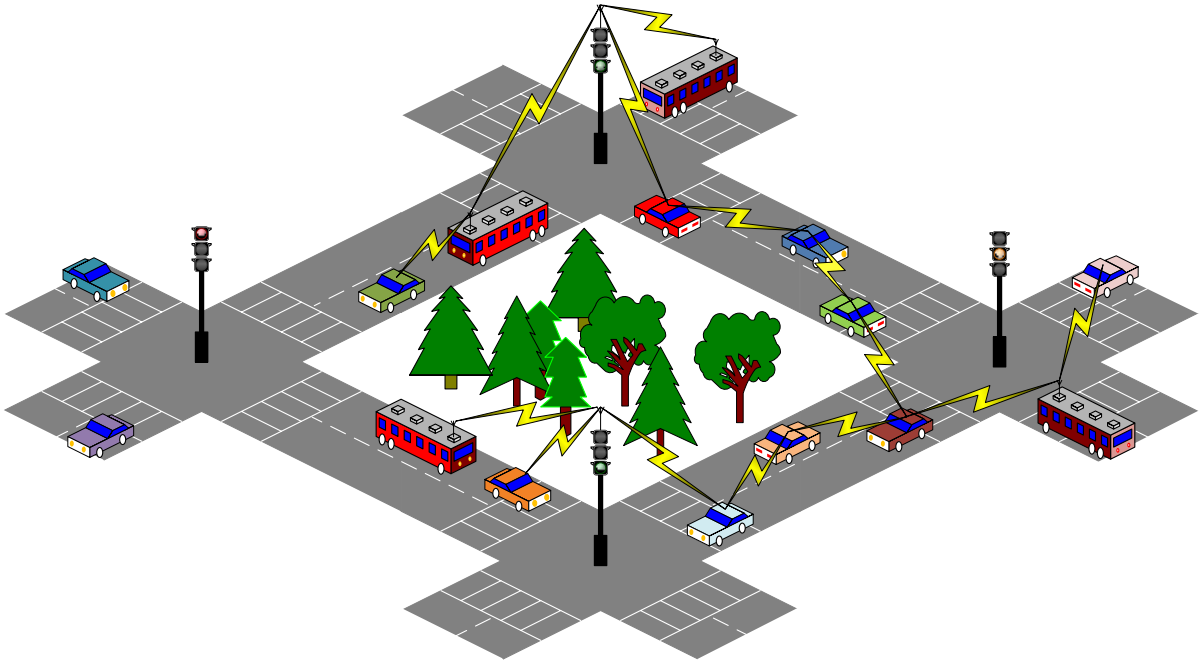


Figure 2 Vehicles and infrastructure working together

There is another type of infrastructure that might be useful. Buses and trams use a predictable route at predictable times. The bus or tram might work as a buffer, they get information from a source, and they can release this information when they get to new areas. In gathering statistical information a moving infrastructure comes in handy, as they cover a large area at regular intervals.

2.4. Information exchange

Each message need to be on a format that is small in size and rich on content.
I have identified two cases of communication that might happen in a vehicle to vehicle system. In case of an accident, or special driving conditions there is a need for a warning. A vehicle that discovers some condition will try and warn other vehicles about this. Information is pushed out to anyone that is able to receive. I have labelled this as a push-message[4].

[4] Pushing information into the system

The information that needs to be transmitted in a push-message:

- Timestamp (identification of the message)
- Message source (who originated the sequence)
- Message condition (what type of information is needed)
- Position (coordinates of interest)
- Message direction (**outwards:** discovery, **return**: back to origin and **push**: warnings)

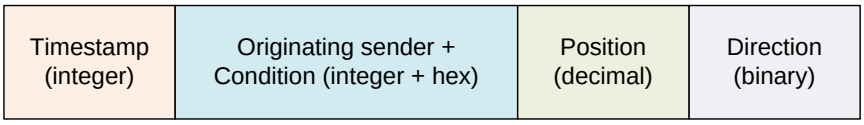


Figure 3 Structure of a push-message

The other case is related to requesting information from the other vehicles. This is gathering information about what vehicles that have some specific conditions. It can be something like "How big is the raincloud", "Are there any queues in front of me". This message aims at collecting information from the system. I have labelled it a pull-message[5].

The "Timestamp" field should be as unique as possible (collisions might appear) so that it is possible to separate the messages from duplicates, and retransmissions. Integers could do this job.

The "Originating sender + Condition" field specifies the ID of the message originator, and what condition the message is regarding. This field is never changed, as this identifies the originator and the condition in question. The ID must be some unique way of determining the sender, an integer should suffice. The use of a MAC-address or a VIN would provide the necessary uniqueness. The "Condition" will contain the information about what kind of information this message is regarding. To be able to send much information in little space one can use hex or binary coding. Binary is more intuitive than hex, but hex contains more information. This is described further in chapter 3.3.

[5] Pulling information from the system

The “Position” field contains the POI. In a push-message this would be the position of the originator. In an outwards pull-message it would contain the originators position, and in the return pull-message it would contain the position of the replier.

The “Direction” field specify the direction of the message, in a push-message the value would inform the receiving vehicles that they should forward this message unaltered, but they don’t need to send a reply to the originator. Other uses of the direction field will be described in chapter 2.5

The pull-message would need some additions to the information in a push-message. One would need to include information about:

- Reply-sender (who replied to the message)
- Reply-sender-status (what are the status of the replier)

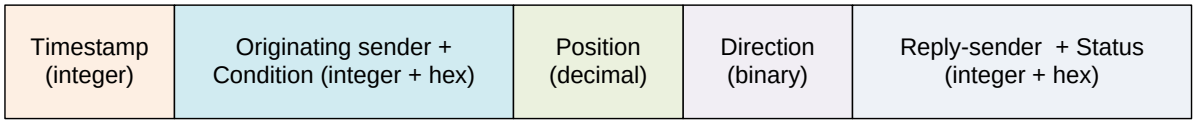


Figure 4 Structure of a pull-message

The various fields will be in different formats, and contain different information. There will also be different rules for who are allowed to change each field, and under which conditions. The specifics concerning the pull-message will be discussed in chapter 2.5

2.5. Area discovery

The discovery of how many vehicles have a specific condition, or how large area the condition covers might provide useful information.

To determine the size of an area data has to be collected, and to do this a pull-message will be appropriate. This message can be constructed as illustrated in Figure 4.

The field "Direction" indicates which direction[6] the message is being transmitted. It is sufficient to identify if the message is a "push" (away from the originator, without reply), "outwards" (away from the originator, expecting reply), or if the message is a "return" (heading back to the originator as a reply). This field should only be changed by an originator, or a replier.

The content of the "Reply-sender + Status" field is dependent on which direction the message is travelling. In an outwards message the originator uses its own address and conditions, while in a return message the replier places its address and status in this field. By doing this the intermediate vehicles can update their information tables with as much information as possible.

There are two ways to determine the area of a condition. One can find all the vehicles that have the condition, and determine the area by comparing their positions. This will give a very accurate description about where the conditions apply, except for the fact that it fails to discover the borders. The result is an underestimate of the size, and would be on the form: "The area is at least this big". This information is interesting, but not as useful as a result on the form "The area is smaller than this".

By constructing the discovering algorithm carefully we can reduce the amount of uncertainty regarding the result. The information we want is the position of vehicles that does not have the condition, but has a neighbour that does. The border of the condition then has to be between these two vehicles.

The example of a raincloud was presented in chapter 1.2. I have used a raincloud as an example since this will illustrate the area determination quite clearly. The function of area determination is of course applicable to many other fields and conditions. In this report I only focus on getting information from the vehicles, I do not explore how this information could be utilized.

[6] Not in any geographical sense, but outwards from, or return to originator

Figure 5 illustrates 5 vehicles[7] inside a raincloud. These vehicles communicate with each other, and with the vehicles[8] just outside the cloud. The vehicles on the outside of the cloud respond by returning their positions, and information that they don't have rain, to the vehicles inside the cloud. The vehicles inside the cloud update their information tables, and pass this information to the other vehicles inside the cloud. Based on the data in the information tables the vehicles can calculate the area that the cloud covers.

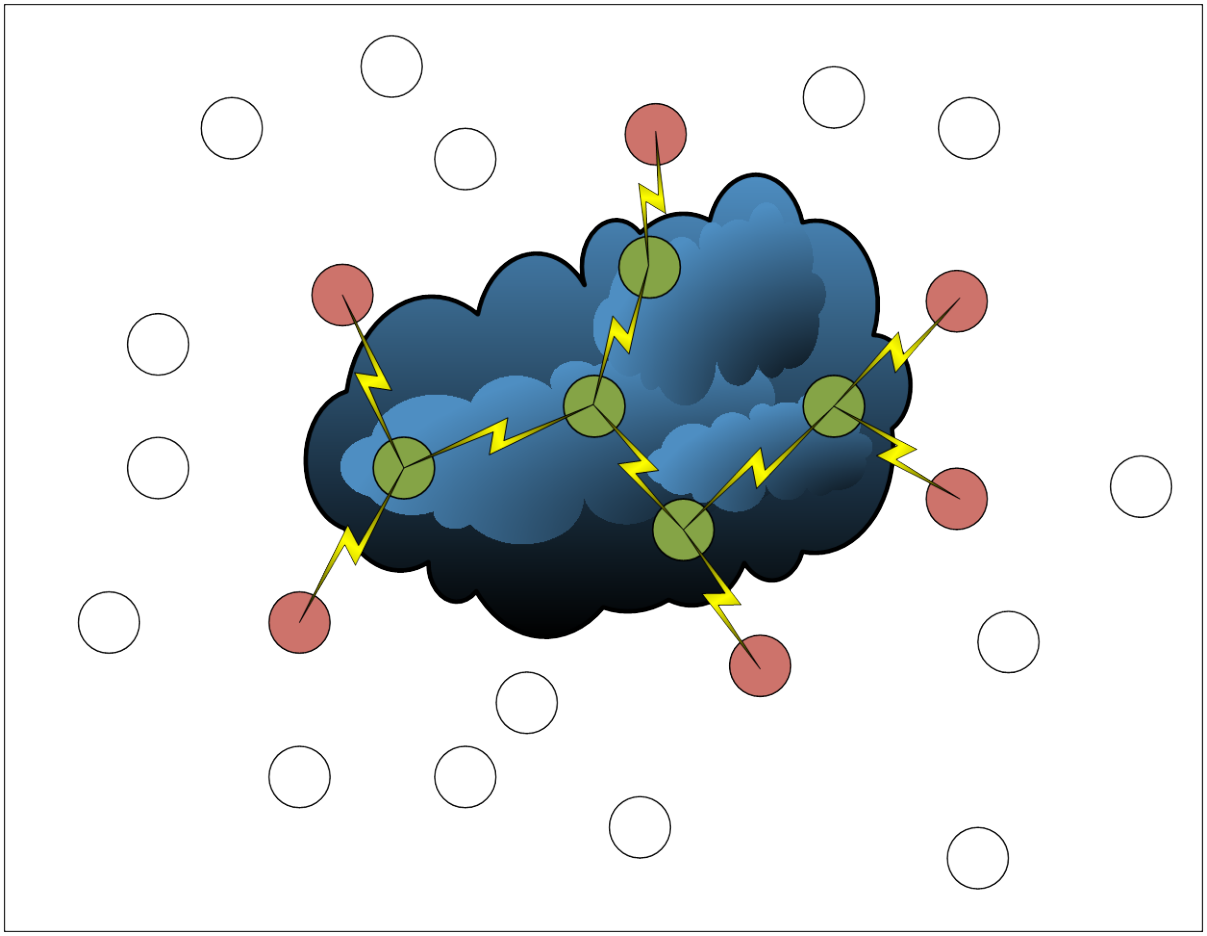


Figure 5 Vehicles determining the size of a raincloud

Flowchart and an algorithm for the area discovery is presented in chapter 3.4

[7] Green circles
[8] Red circles

3. IMPLEMENTATION

3.1. Status maintenance

3.1.1. Keeping information alive

As proposed in chapter 2.2.1 the information needs to be retransmitted. The information that needs to be kept alive is the information that was sent out by the “push-messages”.

The two advantages with an update scheme are: you get a common interpretation about the status in the system, and you get new vehicles up to date. You also get the ability to fill holes left by lost, or discarded, messages.

If a master was present in the system the easy choice would have been to let the master start a chain of updates by either requesting updated information from vehicles, or by sending out its own interpretation of the status in the system. The problem is how to do this in a system without a master.

A solution to this is to make each vehicle responsible for the information it has stored, and send this out to the other vehicles. If every vehicle sent out messages to every other vehicle the result would be complete flooding, redundancy and interference. A more refined method is to let each vehicle take responsibility of updating their neighbours. If every vehicle at regular intervals transmits their status to their neighbours, a common status will evolve within the network. The received updates must be merged with the information within each vehicle. It is necessary to use a majority vote on this merge; this will ensure that the most truthful interpretation will survive. This is illustrated in Figure 6. In case of the majority vote being a tie, one would always choose the “worst-case-scenario”.

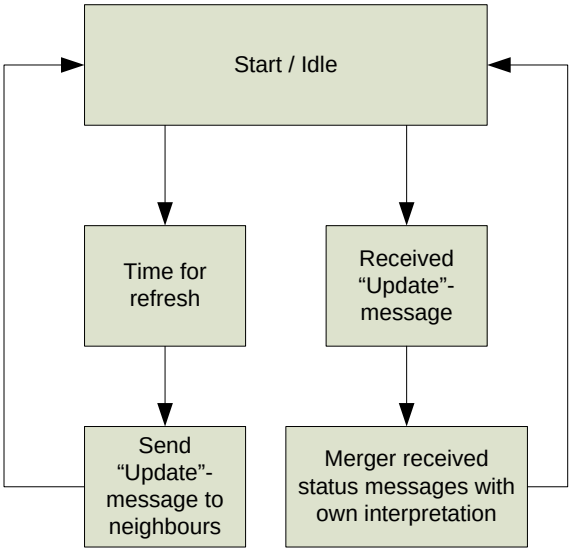


Figure 6 Flowchart for the update procedure

Information might update itself by being present for other vehicles to discover. The presence of an icy spot on the road will cause several vehicles to warn other about the position, and by that the information is kept updated and it will not time out. The paradox in this is that when vehicles actually start to take precautions about the icy spot the information is in danger of timing out and disappearing. This will ultimately lead to some vehicle to re-discover the icy spot, and by that revive the information. It would be desirable to avoid unprepared rediscovery, but the information cannot be updated indefinitely without confirmation about the validity.

Keeping the information alive is one thing; deciding which information to remove is another matter. The information-housekeeping will be discussed in chapter 3.1.2

3.1.2. Keeping the information consistent

A simple update algorithm might keep the information alive, but that is not enough to keep it consistent. There have to be rules about how the information should behave. In the case of the pull-messages it is just a need to discover the area again when you need to get fresh information about the area. It is the push-messages that pose the problem.

The information given by push-messages needs to behave a little different from the pull-messages. As they are warnings there is a great need for them to be correct and not misleading, or false. I propose to use different rules for the different push-messages. This is easy to implement, and will give a very flexible solution.

There has to be some limitations on how far the information will travel. The information about an icy-spot might travel through the network for miles, but this information is not of interest to the vehicles that are not in the area. In this regard a boundary, based on proximity, will be suitable. By saying that the information regarding ice in the road only is valid/interesting if the receiver is within a given distance from the POI we can limit the number of superfluous messages. The use of a distance limit will also contribute to the housekeeping in the vehicles information table. When the vehicle no longer is within the area, the information can be safely removed from the tables.

Information also has a "shelf-life" since the presence of a situation might change. There is no need to warn about ice that has actually melted. In this regard there has to be put a time limit on the information. After some specific time the message will time out, and be removed. If the information is confirmed by other sources it is of course valid, and the table is updated. If the vehicle doesn't receive any new confirmation about the condition within the timeout interval, it is removed from the table. This contributes to the paradox mentioned at the end of chapter 3.1.1.

Information	Timeout interval	Distance from POI
Ice	6000s	2000m
Humidity	3000s	1000m
Temperatur drop	4000s	500m
⋮	⋮	⋮

Table 1 Update intervalls for diffrent push-messages

In Table 1 I have given some examples about what an update scheme might look like. I have given each of the conditions some parameters. This sets a limit to how long the system will keep the information before removal if no new[9] confirmation is received. The last column

[9] Might be the same vehicle as originated the original message, but the condition has been rediscovered.

depicts how far away from the POI the information will travel. This table will typically be stored in each vehicle.

There is another aspect related to the consistency of information. Until now I have focused on consistency between the information available and the real observable conditions. Consistency can be viewed in another way. The information should be consistent in the system as a whole, or at least in significant sub-systems.

As mentioned in chapter 2.2.2 there is a need to include, and inform, new vehicles that have entered the system. My proposal is that vehicles that have been out of contact with other vehicles for some specific time starts to send out a discovery message. The idea is that this message will make any adjacent neighbours respond with their interpretations about the status of the system. A majority vote between these results should provide the new vehicle with the necessary information. This is illustrated in Figure 7.

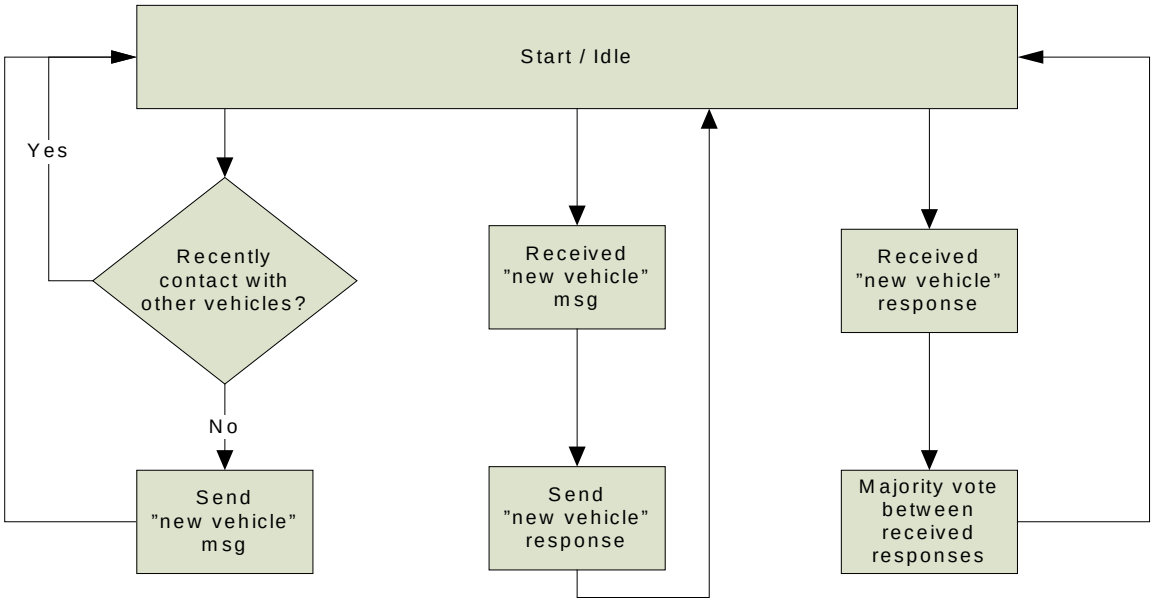


Figure 7 Discovery and mayority vote for "new" vehicles

As the flowchart illustrates this is an action that is "best effort". The longer a vehicle stays in the system, and the more central[10] the vehicle is, the more complete the information table will be.

[10] In relation to other vehicles.

3.2. Infrastructure

Infrastructure can provide extra functionality to the system. The system might view the infrastructure just as another vehicle. The infrastructure is a source of information, and it is a requester of information, just like all the other vehicles in the system. The only differences are that the infrastructure doesn't move (just like a parked car), and the infrastructure might have better coverage due to antenna size/positions. These differences are not something that the system needs to take note of. The system must be constructed in such a way that it can function regardless of the presence of an infrastructure.

In an urban environment the speed of the vehicle doesn't play a very important role in vehicle to vehicle, or vehicle to infrastructure communication. This is because the speed of the transmission is so much higher than the speed of the vehicle [10]. At higher speed there are some limitations on the transmission speed, but this is highly technology dependent, and beyond the scope of this report.

In chapter 2.3 buses was introduced as part infrastructure, and part normal vehicle. This presents some interesting benefits.

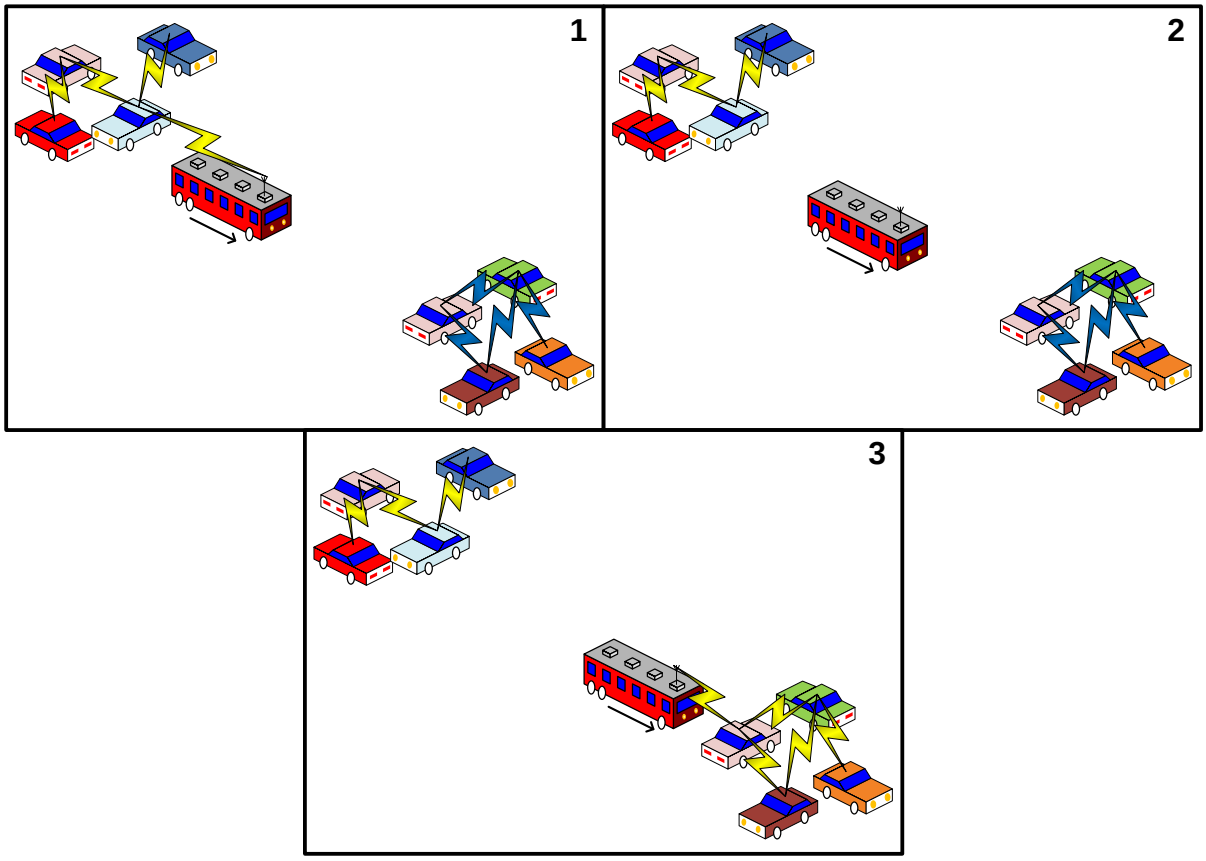


Figure 8 Bus as movable information storage and source

Figure 8 shows how a bus[11] can function as a movable storage of information between two clusters of vehicles. By equipping this type of vehicle with communication equipment the information can be spread from fixed infrastructure to areas that don't have the benefits of a permanent infrastructure. To do this the bus must be able to store any information that other vehicles might find useful, and transmit the information to each new cluster of vehicles. The bus will still function as a normal vehicle, and communicate with the others as usual.

A danger with moving infrastructure is that it might create duplicates and redundant messages. This places great responsibility on the bus, or tram, not to waste resources on information the cluster already possesses.

Part of this research-field is beyond the scope of this report, but more information can be found in the text by Yueyue Li [11].

[11] Or tram

3.3. Information exchange

To make the exchange of information possible it is necessary that the messages follow a specific format. These formats were presented in Figure 3 and Figure 4.

Push-message

The path of a push-message is rather straight forward. A vehicle will forward any message it hasn't seen before, as long as it complies with the rules of the message. (I.e. do not forward if the distance to the POI is too great). No fields of the message should be changed by anyone but the originator. The messages can uniquely be identified with the "Timestamp" and the "Originator_ID" An example regarding the sending of a push-message is presented in Figure 9.

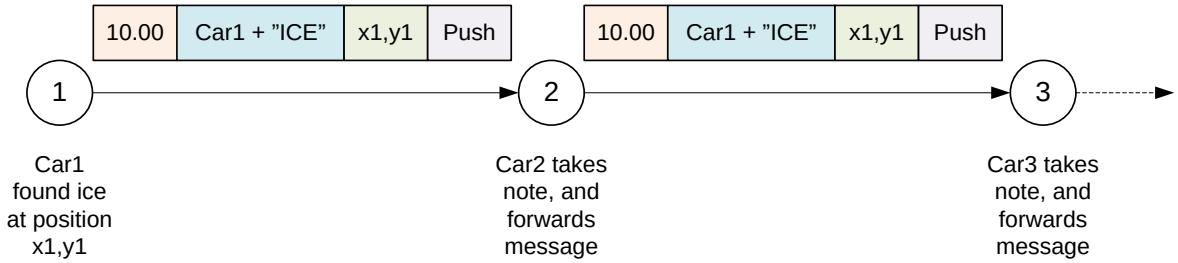


Figure 9 Transmission of a Push-message

To make the transmission more compact and smaller there is a need to compress, and encode the information in such a way that the size is reduced without losing information.

I propose to use binary or hex to do this encoding. The use of hex is more compact, but not as intuitive as the use of binary. This can be seen from Table 2. Each condition is identified by a single bit in the binary condition representation. This method is quick and easy to work with in the implementation, but it is not as effective as the use of a hex representation when it comes to message size. By using hex we can leave out combinations of conditions that aren't necessary to transmit at a given time. There is seldom a need to include information about the "ABS" working if there is no "Motion". By removing unnecessary information the message might be compressed further. There is of course some need to include room for future expansions.

Binary					Hex		Condition
0	0	0	0	0	0	0	No conditions
0	0	0	0	1	0	1	Motion sensor
0	0	0	1	0	0	2	Rain sensor
0	0	0	1	1	0	3	Rain + Motion
0	0	1	0	0	0	4	ABS sensor
0	0	1	0	1	0	5	ABS + Motion
0	0	1	1	0	0	6	ABS + Rain
0	0	1	1	1	0	7	ABS + Rain + Motion
0	1	0	0	0	0	8	⋮
0	1	0	0	1	0	9	⋮
0	1	0	1	0	0	A	⋮
0	1	0	1	1	0	B	⋮
0	1	1	0	0	0	C	⋮
0	1	1	0	1	0	D	⋮
0	1	1	1	0	0	E	⋮
0	1	1	1	1	0	F	⋮
1	0	0	0	0	1	A	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	1	1	F	Update
-	-	ABS	Rain	Motion			

Table 2 Binary and hex representation of vehicle conditions

Pull-message

The path of a pull message is far more complex than the push-message, as it has to take into consideration both messages going out, and replies coming back. Each message should stay unaltered and retransmitted by any intermediate vehicles, and only be altered when there is a reason to do so. This is illustrated in Figure 10. The details about how the messages are handled is described in chapter 3.4

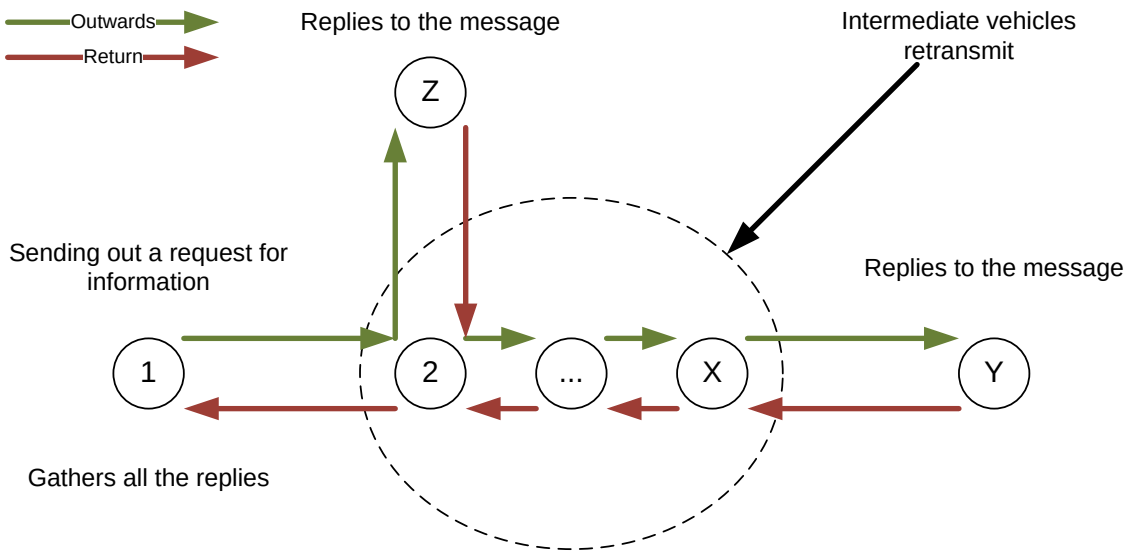


Figure 10 Basic illustration of a Pull-message transmission

By introducing a Pull-message there is a need to distinguish which direction a message is sent. This imposes the addition of the direction field in the message format. The easiest way to separate the directions is by using a simple binary encoding that represents the various directions a message might be sent. As seen in Table 3, it is enough with 2 bits to represent the 3 directions that the system needs. Future need might be satisfied by including an extra bit if necessary.

Value		Direction
0	0	Push
0	1	Outward
1	0	Return
1	1	⋮

Table 3 Direction encoding

In the implementation there is some gain to be found in standardizing as much as possible. As there are very little differences between the Push- and the Pull-message it is possible to use the Pull-format on the Push-message. One might just leave the “Reply-sender + Status” blank. The biggest drawback with this is the fact that one introduces redundancy to a system that has limited recourses. With that in mind it is far better to have two different formats, and let the extra work be done locally in the vehicles.

3.4. Area discovery

Determining the size of an area might give valuable information. As proposed in Chapter 2.5 the most accurate method of determining how large an area is, is to find the outside borders of the condition. To do this we rely on the information from vehicles that are just outside the area. Figure 11 illustrates an algorithm for the message handling when discovering an area.

Each vehicle starts off in the “start/idle” state before it migrates through the various states and actions, until it ends up back in “start/idle” again.

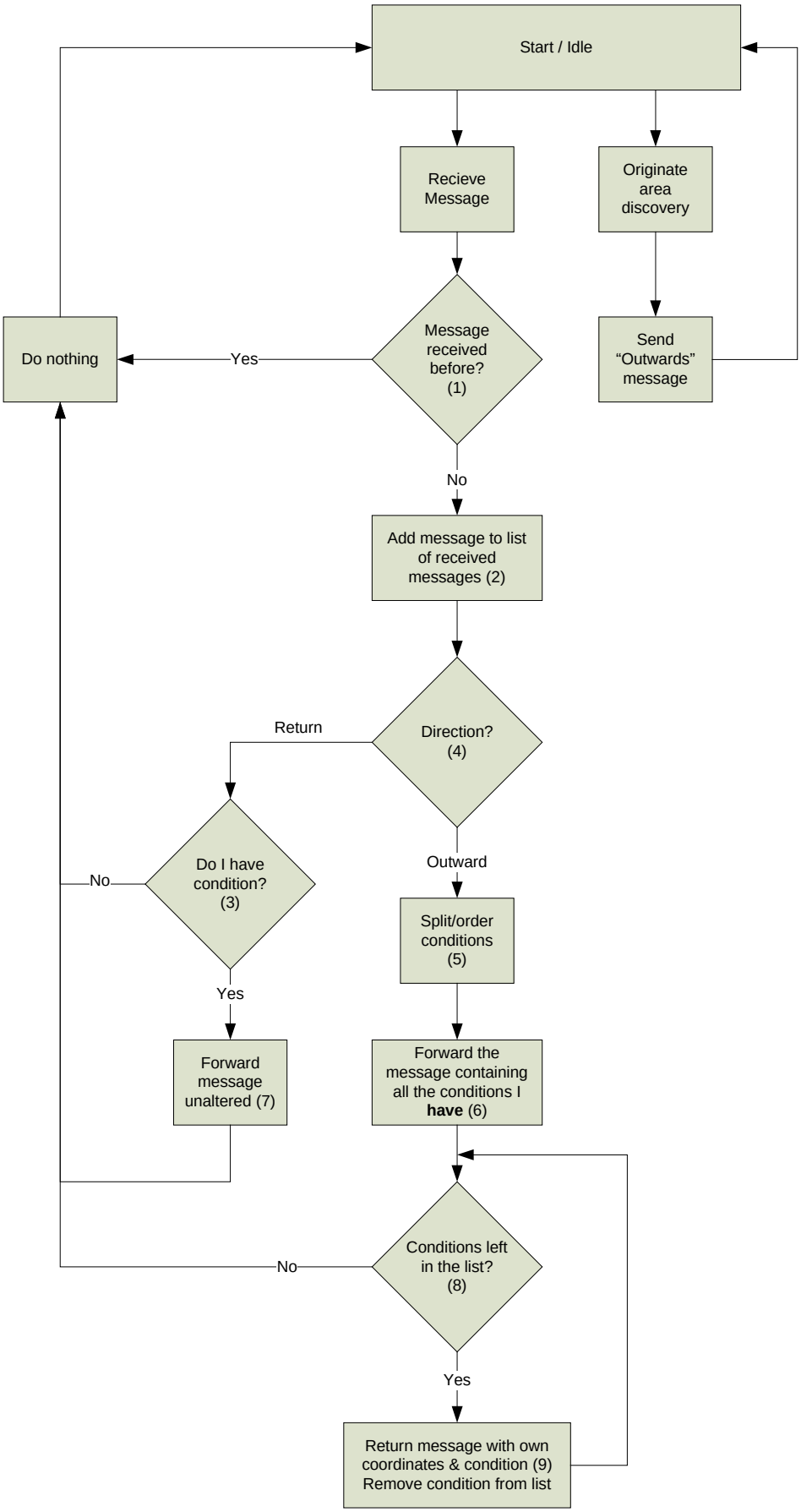


Figure 11 Algorithm for discovering areas

The algorithm presented in Figure 11 illustrates the originator deciding to initiate an area discovery, and how every other vehicle will react when they receive such a message. This algorithm allows the originator to inquire about multiple conditions at the same time. This will make it possible to discover the size of several areas with one message.

A scenario that can be used as an example is the one presented in Figure 12. This scenario consists of four vehicles and two conditions (rain and ice). In this scenario the connections are as follow: 0->(1), 1->(0,2), 2->(1,3), 3->(2).

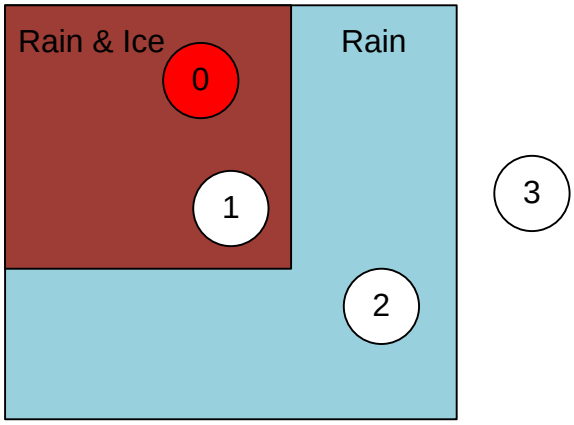


Figure 12 Example of two condition scenario

The area discovery is initiated by Vehicle 0. The aim is to get information about the respective areas. The flow of messages is presented in Figure 13. In this example no duplicate messages are shown, neither are messages that are redundant.

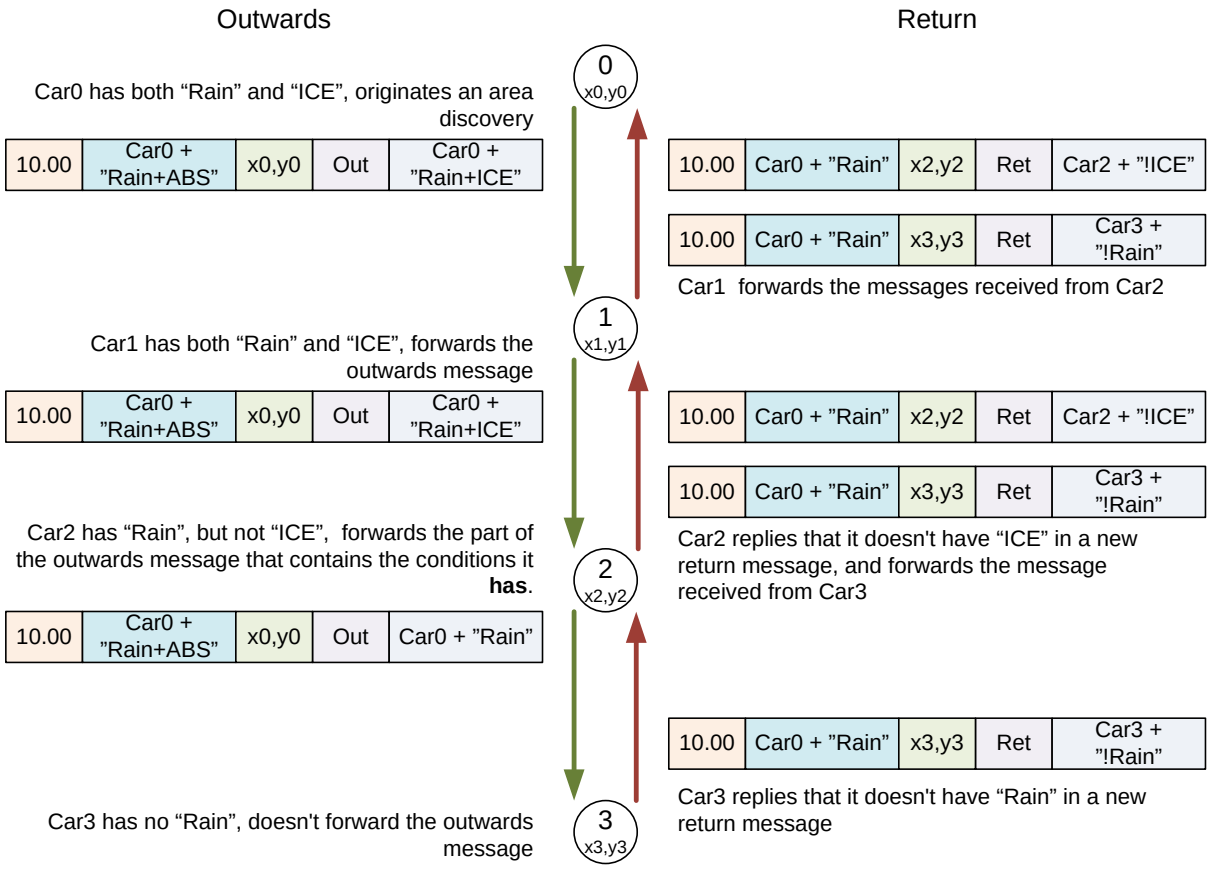


Figure 13 Example of a Pull-message flow

As can be seen from Figure 13 the “outwards” message is only altered when it crosses the “border” of a condition. In this example these borders are between Car1 & Car2, and between Car2 & Car3. The “return” message is never altered by any vehicle. This message also carries the header from the original message sent from the originator (in this case Car0).

A more extensive example regarding discovery of two areas can be found in APPENDIX A: Dual area discovery.

Based on the algorithm in Figure 11 I made the pseudo-code shown in Code 1. I have included numbers in the code to relate to Figure 11. These numbers indicate which paths / blocks that are involved.

```
Message_received {data} {
#Message format: (Timestamp|OriginID:Cond|Direction|X:Y|ReplyID:ReplyCond)

        # Relates to 1, 2, 4, 5, 6, 7, 8, and 9
        IF {(Direction = "Outward") & (Timestamp & OriginID & ReplyID & ReplyCond) = unseen)} {

                # Relates to 1, 2
                ADD (Timestamp & OriginID & ReplyID & ReplyCond) to list of seen messages

                # Relates to 5, 6, 8 and 9
                FOR EVERY Condition in List {
                        IF {condition = my_condition} {
                                ADD Condition to Cond_I_Got
                        } ELSE {
                                # Relates to 9
                                Send (Timestamp|OriginID:Cond|"Return"|My_X:My_Y|My_adr:Condition)
                                ADD (Timestamp & OriginID & My_adr & Condition) to list of seen msg
                        }
                        REMOVE condition from List
                }
                IF {Cond_I_Got != 0}{

                        # Relates to 6
                        Send (timestamp |OriginID:Condition|"Outward"|X:Y|ReplyID:Cond_I_Got)
                        ADD (Timestamp & OriginID & ReplyID & Cond_I_Got) to list of seen messages
                        CLEAR Cond_I_Got
                }
        }
        # Relates to 1, 2, 4, 3, and 7
        IF {(Direction = "Return") & (Timestamp & OriginID & ReplyID & ReplyCond) = unseen) {

                # Relates to 1, 2
                ADD (Timestamp & OriginID & ReplyID & ReplyCond) to list of seen messages

                # Relates to 3, 7
                IF {msg_condition = my_condition} {
                        # Relates to 7
                        Send (data)
                }
        }
        #Do nothing
}
```

Code 1 Pseudo-code for discovering an area

One of the things about the code that is worth taking note of is the criteria to check if the message is seen before: “Timestamp”, “Originator_ID”, “Replier_ID” and “Replier_condition”. By using these fields it is possible to uniquely identify each message from any other. The field “Replier_condition” is included to make sure that a vehicle can respond correctly to a request for multiple areas. Figure 14 illustrates which field in the message that is used for identification.

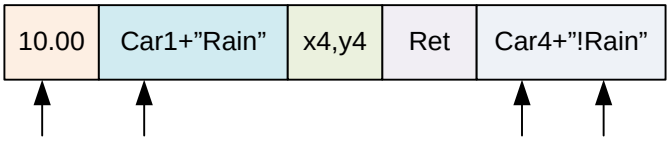


Figure 14 Fields used to identify messages uniquely

A random number of adequate size could replace the need for using so many fields for identification. I have chosen not to use this as that would increase the total amount of data necessary to send. As message size is an important matter in this system the load is now placed on the receiver’s computational power instead of the transmission capacity.

4. SIMULATION

The hypothesis is:

> *The "intelligent flooding" algorithm will perform faster, and with less load on the system than would "pure flooding".*

To prove this I need to make a simulator that can model wireless equipment in vehicles, and their interactions.

4.1. Installation and programming

For the experiments in this report I have used a standard NS2 [12] installation, version 2.29-Allinone. I have run it on Windows XP through Cygwin [13]. Installation instructions can be found on the internet [14] [15]. I used NS2 version 2.29, but there are newer relapses available on-line. I chose 2.29 because it is an installation with which I might get help from personnel at NTU. There should not be any limitations on the experiment due to the choice of installation revision.

The code was made according to previous perfected standards [16] [17] [18]. The main source of information in solving simulator related problems have been found by searching the web.

4.2. The experiment

I would like to test how the algorithm in Figure 11 would perform against a standard "pure flooding" algorithm [8]. I recorded the performance of each algorithm using different numbers of vehicles in order to compare how this affected each algorithm.

Relevant measurements:

- How many packages where sent.
- How many packages were received.
- Time between first and last package received.
- How many packets were discarded because they were duplicates.
- How many packets were dropped due to interference.

To find these values I had to design a scenario that would allow me to manipulate the number of vehicles, and how they behaved.

My scenario is as can be seen in Figure 15. It is similar to the examples presented earlier in this report, but it contains 16 vehicles. The vehicles are given conditions based on their location. Vehicle 0 and 1 have two conditions, vehicle 2 and 5 have one condition (but not the same), all other vehicles do not have any conditions.

The placement, and numbering, of the vehicles allow me to test with 4, 8, 12 and 16 vehicles without having to change their positions.

- To test with 4 vehicles I simply removed vehicles 4 – 15.
- To test with 8 vehicles I simply removed vehicles 8 – 15.
- To test with 12 vehicles I simply removed vehicles 12 – 15.
- To test with 16 there is no need to remove any vehicles.

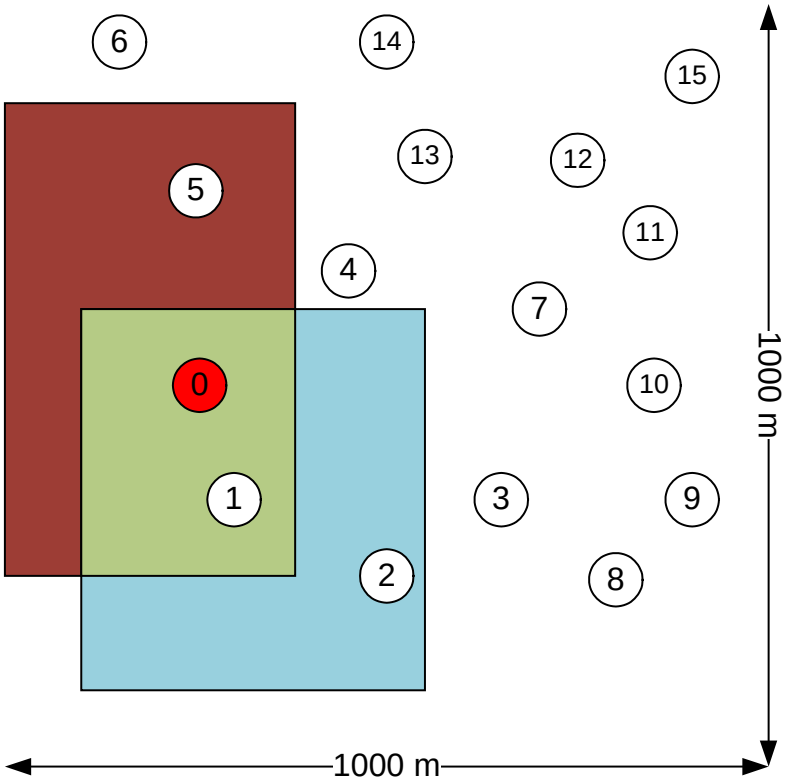


Figure 15 Scenario for the simulator

The distances between the vehicles are rather large[12], this is done to make the simulator work under as difficult conditions as possible.

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. [19]

4.3. The code

The code used is presented in APPENDIX B: Simulator code for 16 vehicles. I have tried to keep the code as simple and clean as possible, but the main focus has been on correct behaviour according to the algorithm in Figure 11. The changes needed to make a "pure flooding" algorithm can be found in APPENDIX C: Replacement code.

After each run the simulator prints out (see Figure 16):

- The number of sent packets.
- The number of received packages.
- The time between first and last packets.
- The number of duplicated packets.

This is done for each of the vehicles in the simulation. There is no randomness built into the code, so that the results will be constant as long as the system is unaltered.

[12] 100m < Distance between vehicles > 250m

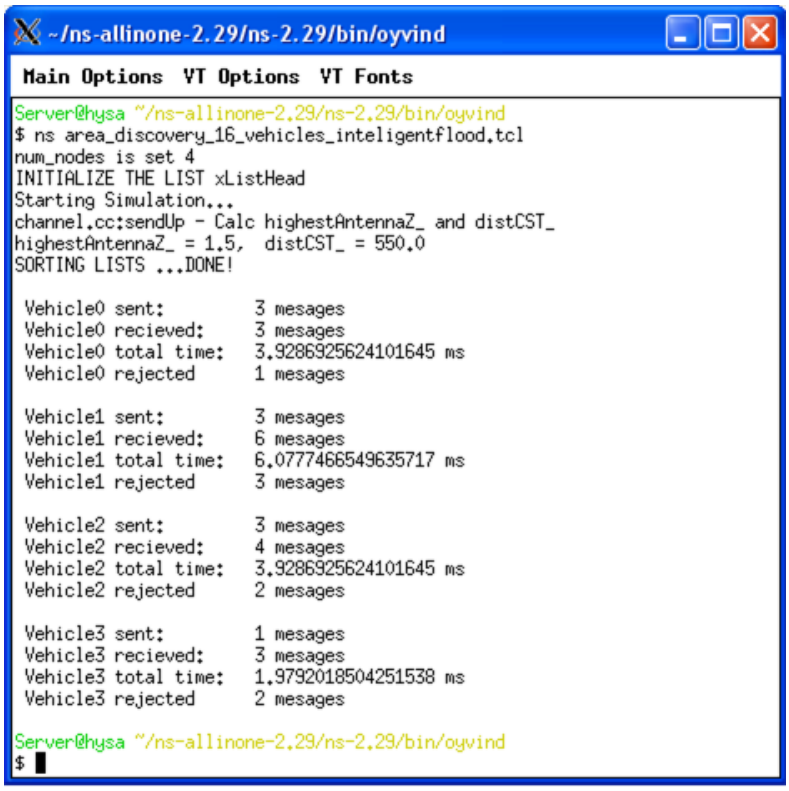


Figure 16 Screenshot from the simulator with 4 vehicles

The vehicles are kept stationary through the whole experiment. This is a simplification that does not influence the result, as the speed of any vehicle will not significantly change the relative position of the sender and receiver within the time it takes to transmit a message. The mathematical background for this will be given in chapter 5.

4.4. Results

The results gathered from the simulator are displayed in Table 4. The table has four main parts, one for each group of vehicles (16, 12, 8 and 4). Each of these groups is presented with results for two scenarios: my "intelligent flooding" algorithm and "pure flooding".

Within these sections:

- "TX" refers to the number of messages sent by each vehicle.
- "RX" refers to the number of received messages.
- "Time"[13] refers to the time between the first and last message at each vehicle.
- "Dis." refers to the number of messages that was discarded as duplicates already received.

I have included a row for the total number in each column, and a row for averaged values. The average time is based on vehicles that are actually involved, leaving out any vehicles that have received 1 packet or less.

[13] In 10^{-3} seconds (ms)

	Intelligent flooding				Pure flooding				Intelligent flooding				Pure flooding				Intelligent flooding				Pure flooding				Intelligent flooding				Pure flooding			
	TX	RX	Time	Dis.	TX	RX	Time	Dis.	TX	RX	Time	Dis.	TX	RX	Time	Dis.	TX	RX	Time	Dis.	TX	RX	Time	Dis.	TX	RX	Time	Dis.	TX	RX	Time	Dis.
Total	16	26	28.04	14	143	282	706.07	154	18	37	43.84	23	89	157	374.83	79	20	41	54.84	25	44	80	138.32	43	10	16	15.88	8	16	24	31.36	12
Avg	1.0	1.6	4.0	0.9	8.9	17.6	44.1	9.6	1.5	3.1	5.5	1.9	7.4	13.1	31.2	6.6	2.5	5.1	6.9	3.1	5.5	10.0	17.3	5.4	2.5	4.0	4.0	2.0	4.0	6.0	7.8	3.0
	16 Vehicles								12 Vehicles								8 Vehicles								4 Vehicles							
Veh.	TX	RX	Time	Dis.	TX	RX	Time	Dis.	TX	RX	Time	Dis.	TX	RX	Time	Dis.	TX	RX	Time	Dis.	TX	RX	Time	Dis.	TX	RX	Time	Dis.	TX	RX	Time	Dis.
0	3	3	2.32	1	8	14	48.01	7	3	4	3.46	2	8	14	32.19	7	4	6	5.47	3	5	8	12.96	4	3	3	3.92	1	4	4	8.24	1
1	2	3	4.74	1	9	17	45.26	9	3	4	7.37	1	8	15	32.99	8	4	6	8.69	2	6	8	21.04	3	3	6	6.07	3	4	8	10.07	5
2	2	2	3.18	1	11	14	44.07	4	3	4	6.08	2	8	11	31.79	4	3	5	8.69	3	6	11	19.77	6	3	4	3.92	2	4	8	8.24	5
3	1	2	0.86	1	9	22	30.07	13	1	3	2.34	2	6	9	24.44	3	1	3	1.89	2	5	9	14.27	4	1	3	1.97	2	4	4	4.81	1
4	2	5	6.09	4	8	18	50.38	11	2	7	7.96	6	6	16	33.58	11	2	7	8.69	6	5	14	21.04	10								
5	5	4	4.74	0	8	17	49.79	10	5	5	7.37	1	9	19	35.63	11	5	6	8.69	2	6	15	21.04	10								
6	1	5	6.11	4	9	8	47.48	0	1	5	7.37	4	10	9	32.99	0	1	5	6.66	4	6	6	13.22	1								
7	0	1	0	1	8	19	49.79	12	0	2	1.89	2	7	15	33.57	9	0	3	6.06	3	5	9	14.98	5								
8	0	0	0	0	9	14	38.07	6	0	1	0	1	7	8	28.81	2																
9	0	0	0	0	10	23	39.95	14	0	1	0	1	6	12	29.41	7																
10	0	0	0	0	10	19	42.27	10	0	1	0	1	7	18	30.01	12																
11	0	0	0	0	9	24	44.15	16	0	0	0	0	7	11	29.42	5																
12	0	0	0	0	8	25	49.19	18																								
13	0	1	0	1	9	26	49.79	18																								
14	0	0	0	0	9	8	48.02	0																								
15	0	0	0	0	9	14	29.78	6																								

Table 4 Results from the simulator with 16, 12, 8 and 4 vehicles

	16 Vehicles		12 Vehicles		8 Vehicles		4 Vehicles	
	Intelligent	Pure	Intelligent	Pure	Intelligent	Pure	Intelligent	Pure
TX	16	143	18	89	20	44	10	16
RX	26	282	37	157	41	80	16	24
Dropped	16	143	9	74	7	16	0	0
Total	42	425	46	231	48	96	16	24
D/Total	38.10 %	33.65 %	19.57 %	32.03 %	14.58 %	16.67 %	0.00 %	0.00 %
D/Vehicle	1.0000	8.9375	0.7500	6.1667	0.8750	2.0000	0.0000	0.0000
D/TX	1.0000	1.0000	0.5000	0.8315	0.3500	0.3636	0.0000	0.0000
D/RX	0.6154	0.5071	0.2432	0.4713	0.1707	0.2000	0.0000	0.0000

Table 5 Total number of sent, received and dropped packets

Table 5 shows the total number of dropped packet in relations to the number of sent and received packets.

“D/Total” is the number of dropped packets in relation to the total number of packets received, and dropped, in the system. This is a measure of the interference between the communicating vehicles.

“D/Vehicle” is the number of dropped packets pr vehicle. This gives an average of how many packets each vehicle has been unable to receive.

“D/TX” is the number of dropped packets pr sent packet. In a wireless system each transmission has potentially several receivers, and it is the receiver that drops the packet.

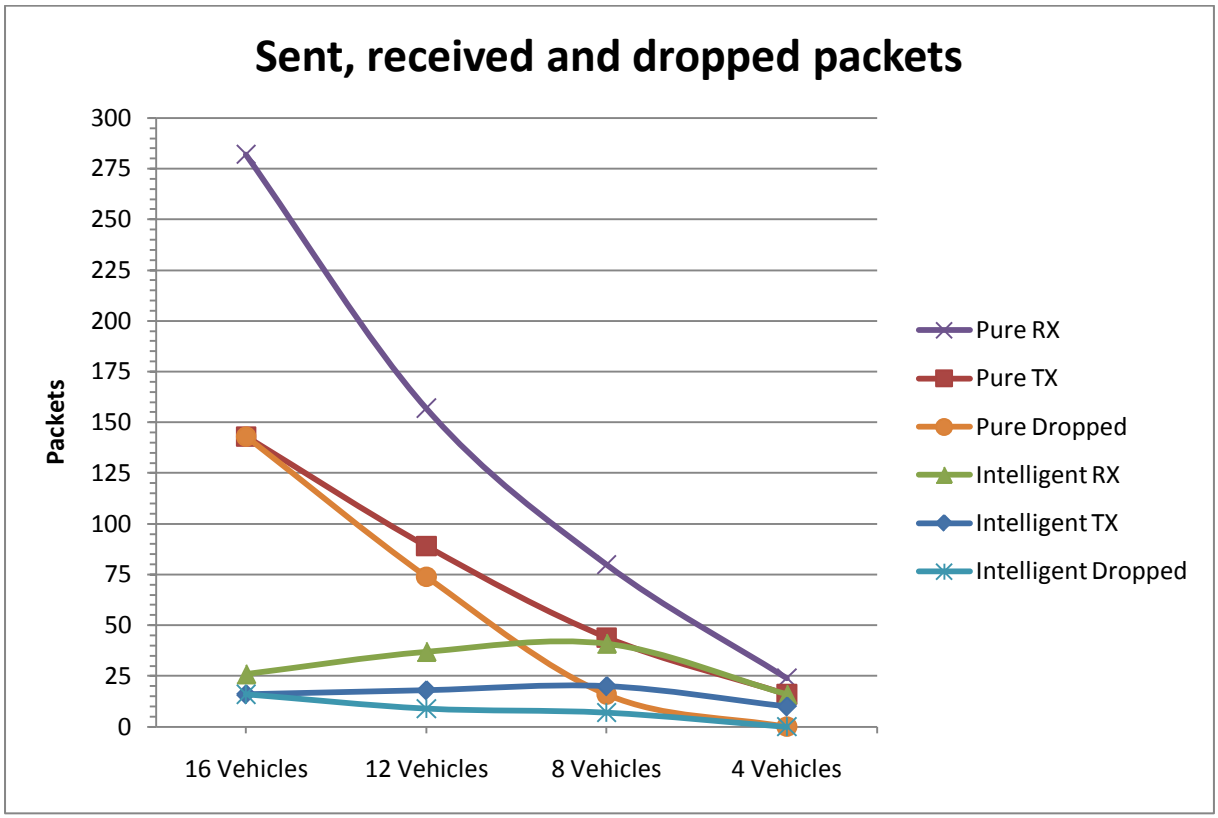
“D/RX” is the number of dropped packets pr successfully received packet. This gives an average of packet loss in the system.

5. DISCUSSION

The simulation was performed with immovable vehicles. My main concern about simulating with static vehicles is that this ignores the difference in interference patterns you would expect with any change in inter vehicular positions. However I assume that the distance each vehicle moves, during the time it is actually involved in the communication, is so short that its movement will not significantly influence the pattern of interference.

From Table 4 we can see that the highest average time any one vehicle was involved is 44.1 ms, at “pure flooding” and “16 vehicles”. The average speed in some major cities in the UK is just 17.8 mph [10] (= 28.2 km/h). An average speed of 30 km/h means that a vehicle will travel 8.3 meters each second. In 44.1 ms the vehicle would then have moved 36.6 cm. This movement is so small that it can be neglected in this simulation. At a speed of 90 km/h the vehicles will move 1.1 meter in 44.1 ms.

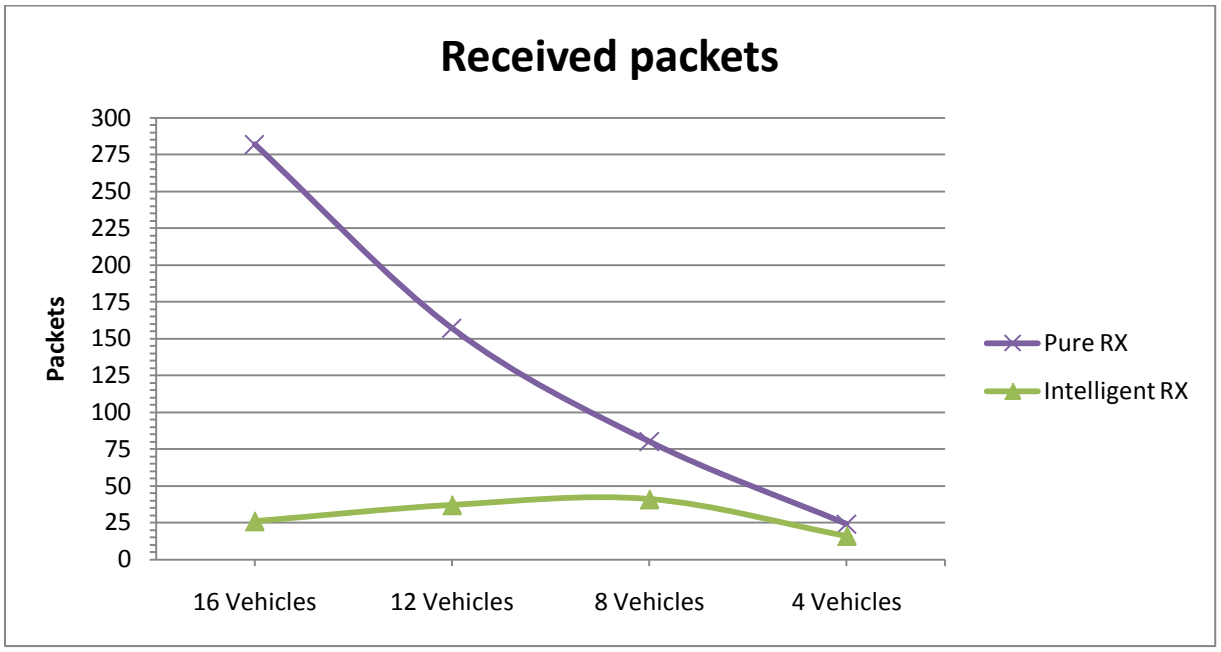
The results in Table 4 and Table 5 can be displayed, and compared, using graphs.



Graph 1 Complete presentation of sent, received and dropped packets

Graph 1 depicts how the “intelligent flooding” algorithm and the “pure flooding” algorithm performed in terms of how many packets that where sent, received and dropped. The shape of the curves belonging to “pure flooding” is as expected, while “sent” and “received” from “intelligent flooding” has a little unexpected shape. This will be discussed shortly.

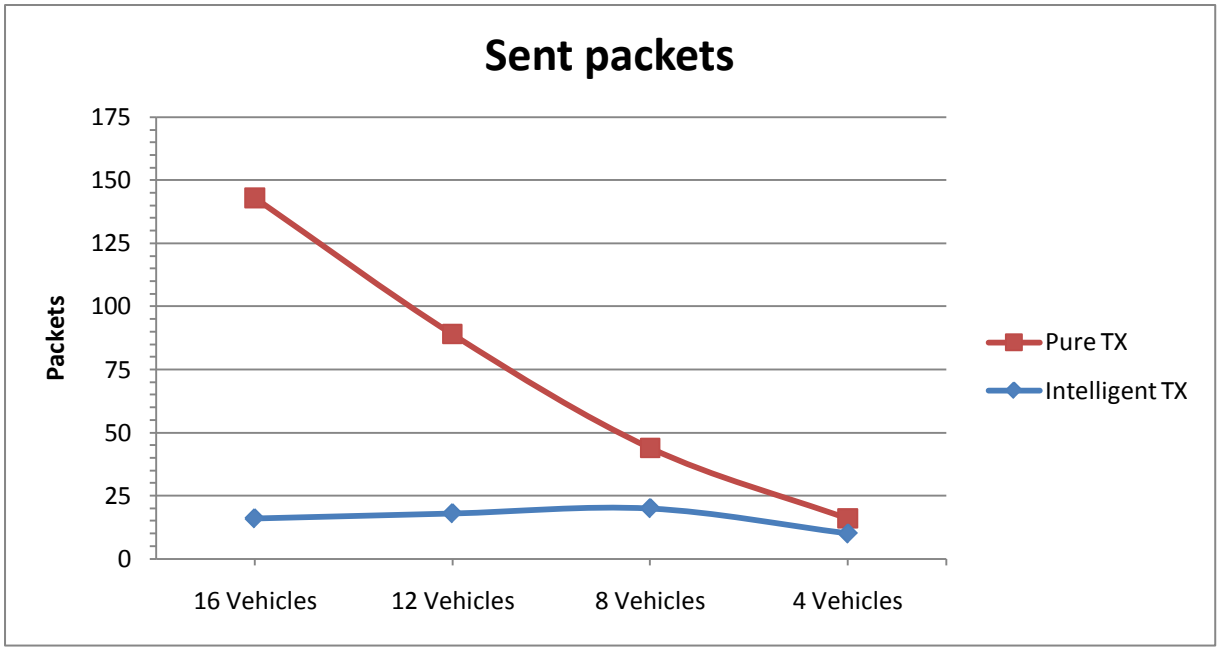
If we break Graph 1 down into smaller graphs it is possible to se details more clearly.



Graph 2 Received packets using both algorithms

From Graph 2 it is evident that the number of received messages peaks at “8 Vehicles” when using the “intelligent flooding” algorithm. The fact that the number of received messages drops with increasing number of vehicles is due to increased interference in the system. This limits the throughput in the system. In “pure flooding” the increased interference is masked by the sheer number of packets.

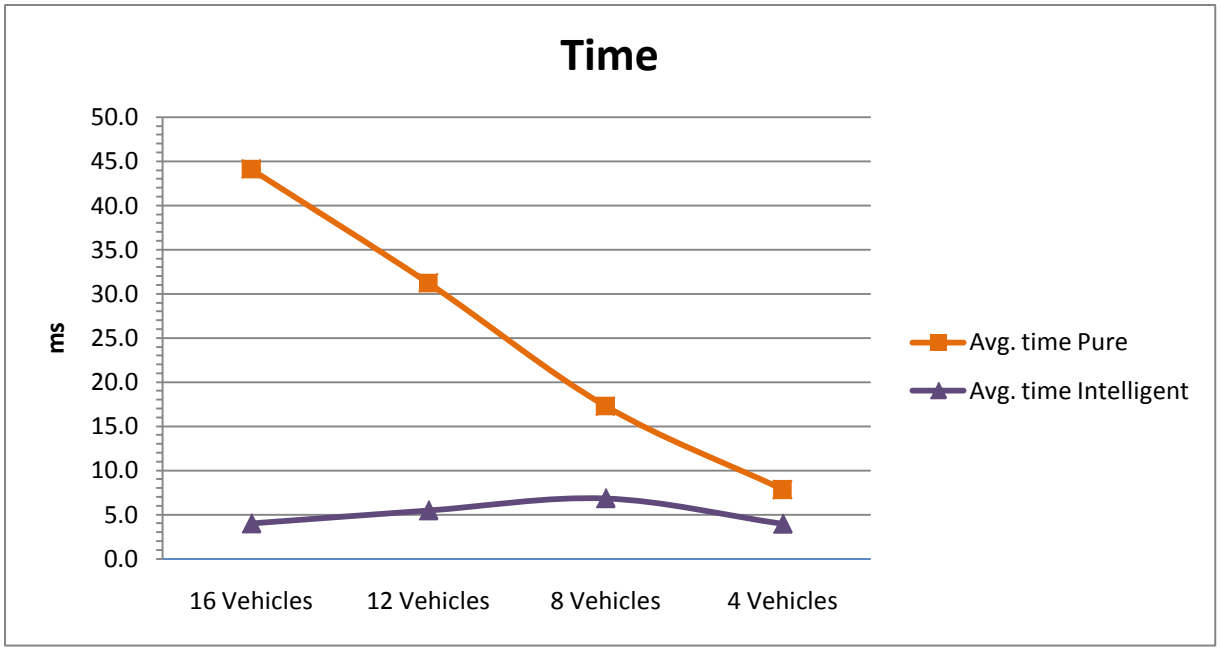
As seen by Graph 2 “intelligent flooding” reduced the number of messages being received by 91% compared to “pure flooding” (16 vehicles). This means that the load on the scarce resources in the system will have increased as well. “Pure flooding” involves more vehicles than is strictly necessary to get the requested information, but offers alternative routes of communication, making up for some of the lost packets.



Graph 3 Sent packets using both algorithms

From Graph 3 one can see that there is a drop in the number of sent messages when the number of vehicles exceeds 8. This is highly linked to the similar drop we could observe in Graph 2. A drop in received messages must give a drop in sent messages, because there are no messages to forward/retransmit.

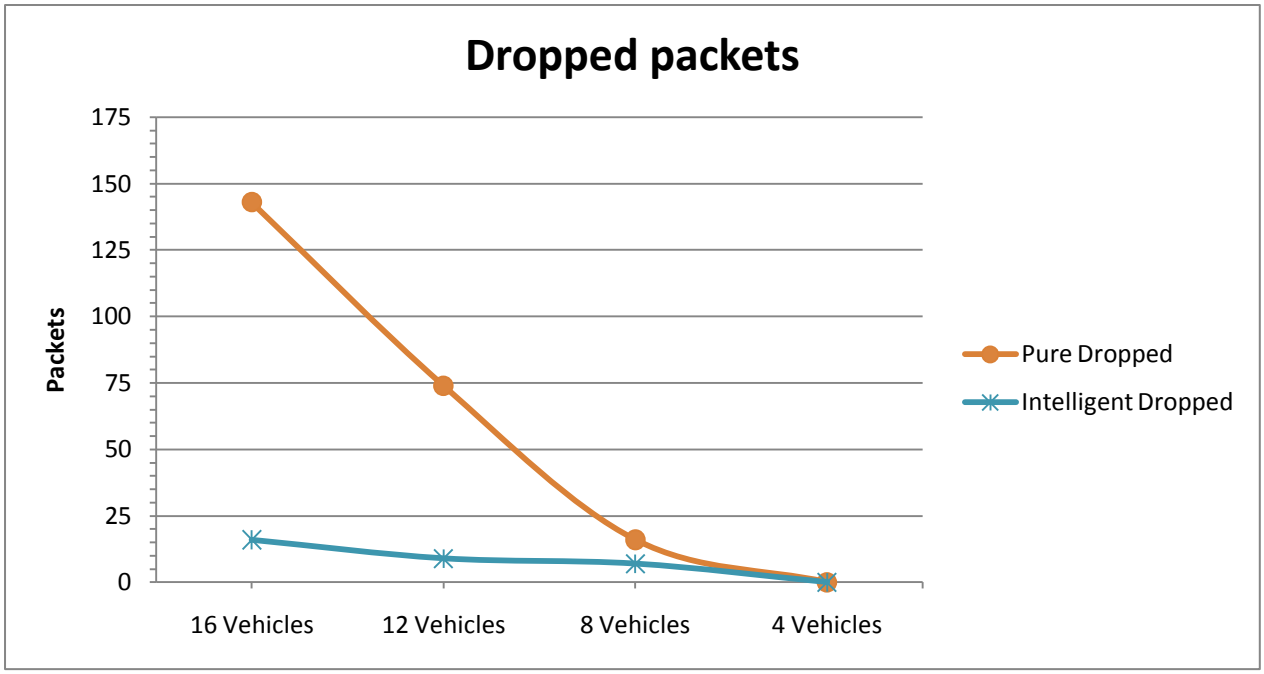
The total number of messages sent in the system is a measure of how big the load on the system is. It is evident that “pure flooding” introduces much more load on the system. “Intelligent flooding” sends 89% less packets than “pure flooding” with 16 vehicles. “Intelligent flooding” uses the recourses in a small number of vehicles, and no resources in the others; “pure flooding” uses the same amount of recourses in all the vehicles.



Graph 4 Average time a vehicle is involved using both algorithms

Graph 4 shows how long, on average, a vehicle is included in the communication. The time a vehicle is considered involved in the communication is based on the time between the first and the last packet received at that vehicle. As expected the increase is rather constant between 4 and 16 vehicles in “pure flooding”. In “intelligent flooding” the shape corresponds well with the curves of Graph 2 and Graph 3. This is not surprising as there is a correlation between the number of received packets and how long the vehicle is involved. More packets received means that the vehicle is involved over a longer period of time.

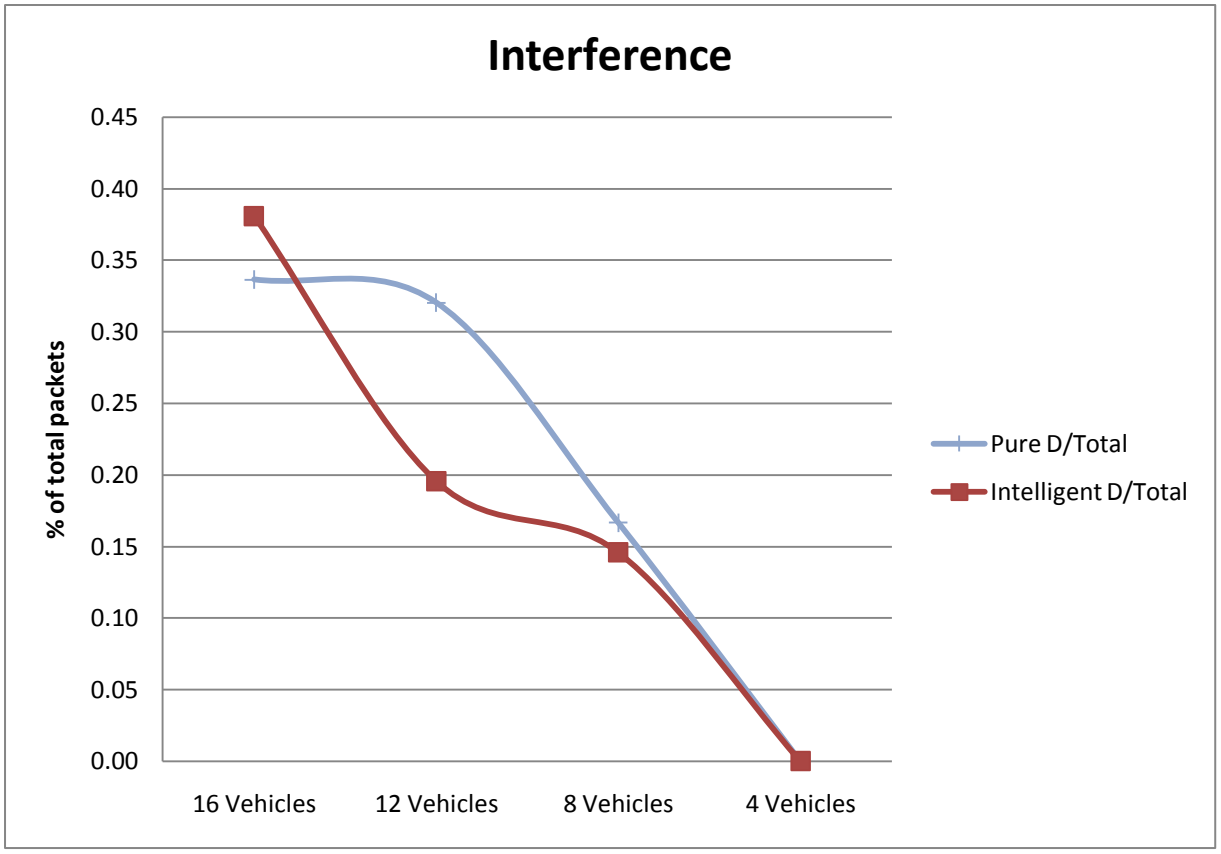
Vehicles involved in “intelligent flooding” spends far less time processing messages, the time is reduced by 91% compared to “pure flooding” (16 vehicles).



Graph 5 Dropped packets using both algorithms

From Graph 5 it is evident that the number of dropped messages increases as the number of vehicles increases. The rise in “intelligent flooding” is rather modest compared with “pure flooding”. The reason for this is the fact that “intelligent flooding” sends fewer messages, and involves fewer vehicles. Each sent message is of greater value in the “intelligent flooding” than in the “pure flooding”. This is related to the possibilities of messages being received through different routes when using “pure flooding”.

This feature makes the “intelligent flooding” more vulnerable to interference than “pure flooding”.



Graph 6 Dropped packets out of the total number of packets

As can be seen in Graph 6 the “intelligent flooding” reduces the number of interference at 12 vehicles, but increases again at 16 vehicles. The “pure flooding” has an unexpected slow rise from 12 to 16 vehicles, but this can be contributed to the fact that most of the new vehicles from 12 to 16 are placed at the edge of the simulation area. The position of the new vehicles means that they contribute to the total number of messages, without increasing the interference as much, and thus “pure flooding” seems less prone to interference than “intelligent flooding” when involving more vehicles.

From Table 5 “D/Vehicle” with “Vehicle 16”, we find that the average of “dropped packets pr vehicle” is much higher in “pure flooding” than in “intelligent flooding”. You could argue that the average loss of packets pr vehicle should be higher in “intelligent flooding”. For the 16 vehicle “intelligent flooding” value in Table 5 it might be more correct to divide the number of lost packets on 9 vehicles rather than 16, as this algorithm excludes non-relevant vehicles from the communication.

As can be seen in Table 4 the vehicles 8–12 and 14-15 is undisturbed by the “intelligent flooding”. This means that there are 9 vehicles that are involved in the communication. That would give an average drop rate (16 dropped messages divided by 9 vehicles) of 1.77, which would be more accurate than 1.0. This is still less than 20% of the “pure flooding” drop rate.

Each sent and received message in “intelligent flooding” contributes more to the result than does each message in “pure flooding” since the information is specific to the conditions we want to explore. This fact also makes each dropped packet more valuable. The communication protocol used in the system is UDP, this was chosen because TCP introduces extra load. However with UDP there is no retransmission if a packet is lost or damaged. Using “intelligent flooding” this is a problem as the algorithm aims at involving as few vehicles as possible. The possibilities of receiving the information via alternate routes are slim. In “pure flooding” there is more redundancy, and therefore it is more likely that the information will be received through several different routes. The presence of several identical messages increases the potential for the receivers to actually receive the message.

There are at least two solutions to the problem of lost messages in “intelligent flooding”:

1. Let the vehicles update each other, as proposed in Figure 6. This would, over time, cover the holes left by lost messages.
2. Use the fact that the vehicles move. After the first run let some time pass before the same discovery is initiated again. After 1 second, with an average of 30 km/h, the vehicles will have moved 8 meters, and after 10 seconds they will have moved 80 meters. This might be enough to change the pattern of packet loss, and thereby gain new information.

I have not implemented vehicular movement in my simulator code, so these solutions have not been investigated in this report.

6. CONCLUSION

At the beginning of this report I raised several questions that needed to be answered.

The first question was regarding the possibilities of constructing a robust and autonomous network for vehicle to vehicle communication. To make such a network it is necessary to equip the vehicles with sufficient communication equipment, and have enough vehicles within reach of each other to make a network. By using ad-hoc protocols the network will be self configuring. The network will also shrink/grow at the rate of the available vehicles. We have all the necessary technology to make such networks. The biggest limiting factor is the density of vehicles. As the speed of the vehicles increases so will the distance between them. This will stretch the network, and possibly, split it into smaller sub-networks.

To keep a fairly updated status among the vehicles I devised an update scheme in chapter 3.1.1 that focuses on letting each vehicle update their nearest neighbours. By choosing this solution the status in the network will converge to a common status without having to involve a centralised server.

In chapter 2.4 I described a message format that I felt would be suited for the exchange of information. I know that there is other work being done [20] within this field, and hopefully the result of this work will give an even better solution. I have used the format I proposed in all my simulations, and it has worked according to the intentions.

The presence of infrastructure only adds advantages to the network. The network will work as intended without infrastructure, but infrastructure might provide several new sources of information and services as described in chapter 2.3. The nature of the network will allow the infrastructure to be regarded as just another vehicle. Infrastructure might also be beneficial when it comes to solving geographical challenges such as corners and narrow streets. In chapter 3.2 I have also shown that buses might be used as moving infrastructure or moving data stores.

There are two different approaches when trying to determine the area in which a certain condition is relevant. It is possible to find all vehicles that have the condition and use this to determine the area covered. The other approach is to find every vehicle that doesn’t have the

condition, and base the determination on them. The latter solution will give an area that is larger than the real situation. This means that the result will include the real situation, and by that give more information than the solution that at best can give a minimum of the size.

The process of determining every vehicle that doesn't have a certain condition will generate a lot of redundant messages. By using the algorithm in Figure 11 I'm able to reduce the number of involved vehicles to a minimum. This algorithm will only include the vehicles that have a neighbour with the condition. In my example I have used "Rain" and "Ice" to illustrate different conditions, but any real life conditions might be far more complex.

The "intelligent flooding" algorithm was tested using an NS2 simulator, and these experiments showed that "intelligent flooding " was faster and less resource consuming than "pure flooding", thereby confirming my hypothesis. The results of the simulator can be seen in chapter 4.4 and they are discussed in chapter 5. The results were as expected, and the simulation proved that "intelligent flooding" produced far less messages, and is much quicker than "pure flooding". It also showed that "intelligent flooding" is vulnerable to interference, and because of that important packets could be lost. Resolving these issues would require some dedicated attention in a later assignment.

The simulator that I made does not include movement of the vehicles; this should be a part of future work. An interesting feature would be to include driving patterns in the simulator in such a manner that the vehicles would adjust their speed according to the distance to the vehicle in front. The simulator from TraNS [21] might prove useful for this as it incorporates some driving patterns and also maps/streets.

There is much work to be done in calculating the specific details based on the information gathered from "intelligent flooding". The migration of the common status is also an interesting aspect that would benefit from further study. A study of behaviour with different technologies could provide useful information [22].

The main conclusion to be derived from this report is: To discover an area, when resources are limited, the best choice would be "intelligent flooding". Without limits on recourses "pure flooding" is easier, and more robust.

REFERENCES

1. **CAR 2 CAR Communication Consortium.** CAR 2 CAR Communication Consortium. [Online] 15 November 2009. [Cited: 15 November 2009.] http://www.car-to-car.org.

2. **Cvis Project.** Cvis Project. [Online] 10 Desember 2009. [Cited: 10 Desember 2009.] http://www.cvisproject.org.

3. **HA EU Watch Project Team.** CALM standards and CVIS. [Online] 01 November 2006. [Cited: 10 Desember 2009.] http://www.haeuwatchits.info/press/press_detail.asp?pid=135&aid=433.

4. **Wikipedia UK.** VANET - Wikipedia. [Online] 21 April 2009. [Cited: 31 September 2009.] http://en.wikipedia.org/w/index.php?title=Vehicular_ad-hoc_network&oldid=285173308.

5. —. Inteligent VANET - Wikipedia. [Online] 31 August 2009. [Cited: 31 September 2009.] http://en.wikipedia.org/w/index.php?title=Intelligent_Vehicular_ad-hoc_Network&oldid=311131215.

6. **Eichler, Stephan, Schroth, Christoph and Eberspächer, Jörg.** Research platform Alexandria. *Car-to-Car Communication.* [Online] 23 October 2006. [Cited: 4 November 2009.] http://www.alexandria.unisg.ch/Publikationen/30950.

7. **Université De Genève.** *From Objects to Mobile Systems and Services.* [ed.] Dimitri Konstantas. Genève : Advanced Systems Group, 2006. pp. 161-165.

8. **Wikipedia UK.** Flooding - Wikipedia. [Online] 31 Oktober 2009. [Cited: 31 Oktober 2009.] http://en.wikipedia.org/w/index.php?title=Flooding_algorithm&oldid=248733394.

9. —. Antenna Theory - Wikipedia. [Online] 7 November 2009. [Cited: 7 November 2009.] http://en.wikipedia.org/w/index.php?title=Antenna_%28radio%29&oldid=324389814.

10. **Massey, Ray.** Daily Mail UK Newspaper. [Online] 26 July 2007. [Cited: 30 October 2009.] http://www.dailymail.co.uk/news/article-471022/Increased-congestion-means-average-speed-towns-17-8mph.html.

11. **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.

12. **VINT project.** Usedr Information -NsNam. [Online] 25 Aug 2009. [Cited: 1 Oct 2009.] http://nsnam.isi.edu/nsnam/index.php?title=User_Information&oldid=4573.

13. **Red Hat Cygwin Product.** Cygwin information and installation. [Online] 14 Jun 2008. [Cited: 1 Oct 2009.] http://cygwin.com/.

14. **Chih-Heng, Ke.** Windows + Cygwin + myNS2. [Online] 26 Oct 2006. [Cited: 1 Oct 2009.] http://hpds.ee.ncku.edu.tw/~smallko/ns2/mysetup_en.htm.

15. **HNS.** Cygwin and ns2 installation. [Online] 01 Sept 2009. [Cited: 05 Oct 2009.] http://ns2-hns.blogspot.com/2009_05_01_archive.html.

16. **The VINT Project.** The ns Manual. [Online] 6 Jan 2009. [Cited: 3 Oct 2009.] http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf.

17. **Greis, Marc.** Marc Greis' Tutorial for the Network Simulator ns. [Online] - - -. [Cited: 10 Oct 2009.] http://www.isi.edu/nsnam/ns/tutorial/index.html.

18. **Yoon, Jisun and Soo, Kim Il.** How to use NS for Wlan. [Online] - - -. [Cited: 15 Nov 2009.] http://nislab.bu.edu/sc546/sc546Fall2002/NS80211/course.html.

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

20. **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.

21. **TraNS.** TraNS (Traffic and Network Simulation Environment). [Online] TraNS. [Cited: 4 November 2009.] http://trans.epfl.ch/.

22. **CAR 2 CAR Communication Consortium.** Overview of the C2C-CC System. *Manifesto.* Public : CAR 2 CAR Communication Consortium, 2007. Vol. 2007, 1.1.

APPENDIX A: Dual area discovery

This appendix describes discovery of two semi-overlapping conditions. Figure 17 illustrates what this might look like. The areas are shown as red (represents Rain) and blue (represents Ice). Green represents the area where both Rain and Ice is present. The number of vehicles (0 to 7) is kept at a bare minimum to keep the amount of messages manageable. The vehicles are positioned so as to illustrate as many different combinations of behaviour as possible.

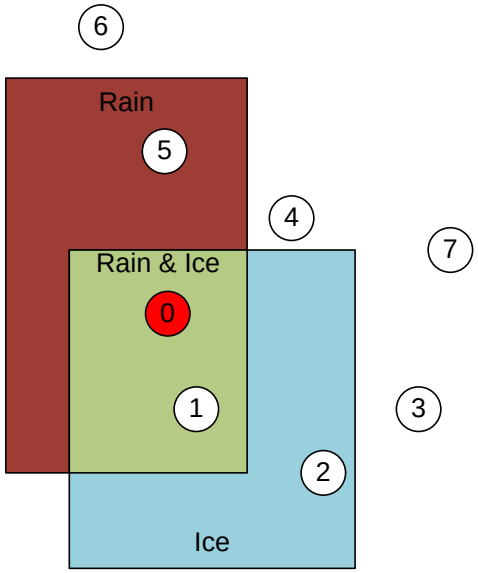


Figure 17 Two area discovery, large example

The different vehicles are within reach of different neighbours. Table 6 displays which vehicle is within reach of each other.

From:	0	1	2	3	4	5	6	7
To:	1,4,5	0,2	1,3	2,7	0,5,7	0,4,6	5	4,3

Table 6 Routing table for two area discovery example

This example will not deal with the collected data, such as to calculate the area borders. The number of vehicles is too small to illustrate border definition satisfactorily. This example intends to show which messages are received at each vehicle, and what response this causes.

The red vehicle (Vehicle 0) starts off the discovery by sending out a query[14] about Rain- and Ice-area. In Table 7 to Table 14 all the sent and received messages are shown. The tables are separated in two sections: messages sent and messages received. The received section describes who it received the message from, and if the message should be forwarded, replied to or discarded. A vehicle should discard any message that is either seen before or without relevance to that vehicle. There are two reasons for a message to be of no relevance to the vehicle: The vehicle has already seen another version of the same message, or the vehicle is "outside" the area of interest, and will correctly discard any "return" messages.

The arrows in the table indicate that a received message has triggered the sending of one or more messages from the vehicle. This is in accordance with the algorithm in Figure 11. The messages in Table 7 to Table 14 are not ordered chronologically, but rather grouped together by source. This is to improve readability of the tables.

As in accordance with the specifications given in chapter 3.4 the "Time" and "Source+cond." is unaltered at all times. These fields identify messages related to the query originated by "Vehicle 0". The notation (!Ice) means ("not Ice"), and represents the fact that the vehicle is not within the Ice area.

[14] Red message in Table 7

Sent from Vehicle 0						Received at Vehicle 0						
Time	Source+cond	Pos	Dir	Reply+ cond.		Time	Source+cond	Pos	Dir	Reply+ cond.	From	Action
1.00	0+(Rain,Ice)	x0,y0	Out	0+(Rain, Ice)								
						1.00	0+(Rain,Ice)	x0,y0	Out	0+(Rain, Ice)	1	Discard
						1.00	0+(Rain,Ice)	x0,y0	Out	0+(Rain)	5	No use
1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)	<--	1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)	1	Forward
						1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)	5	Discard
1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)	<--	1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)	5	Forward
						1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)	1	Discard
1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	<--	1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	4	Forward
1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)	<--	1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)	4	Forward
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	5	Discard
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	1	Discard
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)	1	Discard
1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	<--	1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	1	Forward
						1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	5	Discard
1.00	0+(Rain,Ice)	x6,y6	Ret	6+(!Rain)	<--	1.00	0+(Rain,Ice)	x6,y6	Ret	6+(!Rain)	5	Forward
						1.00	0+(Rain,Ice)	x6,y6	Ret	6+(!Rain)	1	Discard

Table 7 Messages sent and received by Vehicle 0

Sent from Vehicle 1						Received at Vehicle 1						
Time	Source+cond	Pos	Dir	Reply+ cond.		Time	Source+cond	Pos	Dir	Reply+ cond.	From	Action
1.00	0+(Rain,Ice)	x0,y0	Out	0+(Rain, Ice)	<--	1.00	0+(Rain,Ice)	x0,y0	Out	0+(Rain, Ice)	0	Forward
						1.00	0+(Rain,Ice)	x0,y0	Out	0+(Ice)	2	No use
1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)	<--	1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)	2	Forward
						1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)	0	Discard
1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)	<--	1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)	0	Forward
						1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)	2	Discard
1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	<--	1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	0	Forward
1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)	<--	1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)	0	Forward
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)	2	Discard
1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	<--	1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	2	Forward
						1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	0	Discard
1.00	0+(Rain,Ice)	x6,y6	Ret	6+(!Rain)	<--	1.00	0+(Rain,Ice)	x6,y6	Ret	6+(!Rain)	0	Forward

Table 8 Messages sent and received by Vehicle 1

Sent from Vehicle 2						Received at Vehicle 2						
Time	Source+cond	Pos	Dir	Reply+ cond.		Time	Source+cond	Pos	Dir	Reply+ cond.	From	Action
1.00	0+(Rain,Ice)	x0,y0	Out	0+(Ice)	<--	1.00	0+(Rain,Ice)	x0,y0	Out	0+(Rain, Ice)	1	Forward
1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)								Reply
						1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)	1	Discard
1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)	<--	1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)	1	Forward
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	1	No use
1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)	<--	1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)	1	Forward
1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	<--	1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	3	Forward
						1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	1	Discard
						1.00	0+(Rain,Ice)	x6,y6	Ret	6+(!Rain)	1	No use

Table 9 Messages sent and received by Vehicle 2

Sent from Vehicle 3						Received at Vehicle 3						
Time	Source+cond	Pos	Dir	Reply+ cond.		Time	Source+cond	Pos	Dir	Reply+ cond.	From	Action
1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	<--	1.00	0+(Rain,Ice)	x0,y0	Out	0+(Ice)	2	Reply
						1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	2	Discard
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)	2	No use
						1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)	2	No use
						1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)	2	No use

Table 10 Messages sent and received by Vehicle 3

Sent from Vehicle 4						Received at Vehicle 4						
Time	Source+cond	Pos	Dir	Reply+ cond.		Time	Source+cond	Pos	Dir	Reply+ cond.	From	Action
1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	<--	1.00	0+(Rain,Ice)	x0,y0	Out	0+(Rain, Ice)	0	Reply
1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)								Reply
						1.00	0+(Rain,Ice)	x0,y0	Out	0+(Rain)	5	No use
						1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)	0	No use
						1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)	5	Discard
						1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)	5	No use
						1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)	0	Discard
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	5	No use
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	0	Discard
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)	0	No use
						1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	0	No use
						1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	5	Discard
						1.00	0+(Rain,Ice)	x6,y6	Ret	6+(!Rain)	5	No use
						1.00	0+(Rain,Ice)	x6,y6	Ret	6+(!Rain)	0	Discard

Table 11 Messages sent and received by Vehicle 4

Sent from Vehicle 5						Received at Vehicle 5						
Time	Source+cond	Pos	Dir	Reply+ cond.		Time	Source+cond	Pos	Dir	Reply+ cond.	From	Action
1.00	0+(Rain,Ice)	x0,y0	Out	0+(Rain)	<--	1.00	0+(Rain,Ice)	x0,y0	Out	0+(Rain, Ice)	0	Forward
1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)								Reply
1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)	<--	1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)	0	Forward
						1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)	0	Discard
1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	<--	1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	4	Forward
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)	4	Discard
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	0	No use
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)	0	No use
1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	<--	1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	0	Forward
1.00	0+(Rain,Ice)	x6,y6	Ret	6+(!Rain)	<--	1.00	0+(Rain,Ice)	x6,y6	Ret	6+(!Rain)	6	Forward
						1.00	0+(Rain,Ice)	x6,y6	Ret	6+(!Rain)	0	Discard

Table 12 Messages sent and received by Vehicle 5

Sent from Vehicle 6						Received at Vehicle 6						
Time	Source+cond	Pos	Dir	Reply+ cond.		Time	Source+cond	Pos	Dir	Reply+ cond.	From	Action
1.00	0+(Rain,Ice)	x6,y6	Ret	6+(!Rain)	<--	1.00	0+(Rain,Ice)	x0,y0	Out	0+(Rain)	5	Reply
						1.00	0+(Rain,Ice)	x2,y2	Ret	2+(!Rain)	5	No use
						1.00	0+(Rain,Ice)	x5,y5	Ret	5+(!Ice)	5	No use
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	5	No use
						1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	5	No use
						1.00	0+(Rain,Ice)	x6,y6	Ret	6+(!Rain)	5	Discard

Table 13 Messages sent and received by Vehicle 6

Sent from Vehicle 7						Received at Vehicle 7						
Time	Source+cond	Pos	Dir	Reply+ cond.		Time	Source+cond	Pos	Dir	Reply+ cond.	From	Action
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Rain)	4	No use
						1.00	0+(Rain,Ice)	x4,y4	Ret	4+(!Ice)	4	No use
						1.00	0+(Rain,Ice)	x3,y3	Ret	3+(!Ice)	3	No use

Table 14 Messages sent and received by Vehicle 7

APPENDIX B: Simulator code for 16 vehicles

The following code was used to produce the results in Table 4 regarding “intelligent flooding”. The alterations needed to use it for 12, 8 and 4 vehicles are few. The places that need alterations are highlighted in red in the code.

To produce results simulating “pure flooding” the code had to be altered to some extent. The alteration is highlighted in blue, and the replacement code can be found in APPENDIX C: Replacement code.

```
# ======================================================================
#Author: Øyvind Risan
# ======================================================================
#  This simulator will simulate 0-15 stationary vehicles (depending on settings),
#  and let them communicate using UDP.
#
#  The flooding algorithm will keep the number of messages at a minimum.
#  To use pure flooding instead of intelligent flooding some of the code needs to be swapped out.
#
#  The simulator is set up to let vehicle 0 discover 2 areas.
#  All vehicles that have both conditions will forward the message unaltered.
#  All vehicles that have one of the conditions will forward the message containing the condition they have.
#  All vehicles that don’t have any of the conditions will not forward the message,
#  but reply using an return  message.
#  All vehicles that don’t have the condition will discard the return message.
#  All vehicles that have the condition will forward a return message unaltered.
#
# Routing Table
#  0->(1,5,4), 1->(0,2), 2->(1,3), 3->(2,7,8,9,10), 4->(0,5,7,13), 5->(0,4,6), 6->(5), 7->(3,4,10,11,12,13)
#  8->(3,9), 9->(3,8,10), 10->(3,7,9), 11->(7,10,12,15), 12->(7,11,13,15), 13->(7,4,12,14), 14->(13),
# 15->(11,12)
#
# Simulator saves the tracefile: tr-output.tr, and starts NAM using the nam fil: nam-output.nam
#
# After the simulation is done it prints out:
# VehicleX sent:          YY messages"
# VehicleX received:      YY messages"
# VehicleX total time:    YY ms"
# VehicleX rejected       YY messages \n"
#
# Important: the first message must be sent on other than 0.0 due to time calculations
#

# ======================================================================
# Define Environment options
# ======================================================================
set optNodes            16                  ;# defines the number of nodes
set opt(duration)       100                 ;# defines the duration of the simulation
set opt(out-name)       "output"            ;# defines the name of the output file
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)         50                  ;# defines the size of the displayed nodes
```

```
# ======================================================================
# Define Operation options
# ======================================================================
set opt(cbr_size)       500
set opt(cbr_interval)   0.002
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)

set MESSAGE_PORT 42
set BROADCAST_ADDR -1

array set DISCARD {}
array set RX {}
array set TX {}
array set FIRST {}
array set LAST {}

# ======================================================================
# Initialize ns_
# ======================================================================
set ns_ [new Simulator]
set tracefd [open tr-$opt(out-name).tr w]
$ns_ trace-all $tracefd

set nf [open nam-$opt(out-name).nam w]
$ns_ namtrace-all-wireless $nf $opt(X) $opt(Y)
$ns_ use-newtrace

# ======================================================================
# Define color 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 queuetype when DSR, prevents "Ccore dumped"
# ======================================================================
if { $val(rp) == "DSR" } {
  set val(ifq)          CMUPriQueue
} else {
```

```
  set val(ifq)            Queue/DropTail/PriQueue
}

# ===================================================================
#creates the God entity
# ===================================================================
create-god $optNodes

# ===================================================================
#configures the node template
# ===================================================================
$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 ON -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
}

# ===================================================================
# Provide initial (X,Y) co-ordinates for the nodes
# ===================================================================

$node_(0) set X_ 250.0
$node_(0) set Y_ 450.0

$node_(1) set X_ 300.0
$node_(1) set Y_ 300.0

$node_(2) set X_ 500.0
$node_(2) set Y_ 200.0

$node_(3) set X_ 650.0
$node_(3) set Y_ 300.0

$node_(4) set X_ 450.0
$node_(4) set Y_ 600.0

$node_(5) set X_ 250.0
$node_(5) set Y_ 700.0

$node_(6) set X_ 150.0
$node_(6) set Y_ 900.0

$node_(7) set X_ 650.0
```

```
$node_(7) set Y_ 550.0

$node_(8) set X_ 800.0
$node_(8) set Y_ 200.0

$node_(9) set X_ 900.0
$node_(9) set Y_ 300.0

$node_(10) set X_ 850.0
$node_(10) set Y_ 450.0

$node_(11) set X_ 850.0
$node_(11) set Y_ 650.0

$node_(12) set X_ 750.0
$node_(12) set Y_ 750.0

$node_(13) set X_ 550.0
$node_(13) set Y_ 750.0

$node_(14) set X_ 500.0
$node_(14) set Y_ 900.0

$node_(15) set X_ 900.0
$node_(15) set Y_ 850.0

# ======================================================================
#Initialise the nodes
# ======================================================================
for {set i 0} {$i < $optNodes } {incr i} {
        $ns_ initial_node_pos $node_($i) $opt(n-size)
}

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

# ======================================================================
#Reciveing 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

        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]

        # ============================================================
        # extract Information from the message
        # ============================================================
```

```
        set Timestamp    [lindex [split $data ":"] 0]
        set Origin_ID    [lindex [split $data ":"] 1]
        set Cond         [lindex [split $data ":"] 2]
        set Direction    [lindex [split $data ":"] 3]
        set x_pos_rec    [lindex [split $data ":"] 4]
        set y_pos_rec    [lindex [split $data ":"] 5]
        set ReplyID      [lindex [split $data ":"] 6]
        set ReplyCond    [lindex [split $data ":"] 7]

        if {(($Direction == "OUT") & ([lsearch $messages_seen $Timestamp+$Origin_ID] == -1)) +
           (($Direction == "RET") & ([lsearch $messages_seen
$Timestamp+$Origin_ID+$ReplyID+$ReplyCond] == -1))} {

                $ns_ trace-annotate "$Node_adr received {$data} from $source"
                puts "$Node_adr received $data from $source"

                if {($Direction == "OUT")} {
                        lappend messages_seen $Timestamp+$Origin_ID

                        #$ns_ trace-annotate "$Node_adr received {$data} from $source"
                        #puts "$Node_adr received $data from $source"

                        set j 1
                        set i 0
                        set CondList ""
                        set counter 0

                        while {$j} {
                                set condition [lindex [split $ReplyCond "+"] $i]
                                if {$condition != "!"} {
                                        if {([lsearch $Status $condition] >= 0)} {
                                                append CondList $condition+
                                                set counter [expr $counter + 1]
                                        } else {
                                                set x_pos [$node_ set X_]
                                                set y_pos [$node_ set Y_]
                                                $self sendto $size
"$Timestamp:$Origin_ID:$Cond:RET:$x_pos:$y_pos:$Node_adr:$condition" $BROADCAST_ADDR $sport
                                                lappend messages_seen
$Timestamp+$Origin_ID+$Node_adr+$condition
                                                set ::TX($Node_adr) [expr $TX($Node_adr) + 1]
                                        }
                                } else {
                                        set j 0
                                        append CondList !
                                }
                                set i [expr $i + 1]
                        }

                        if {($counter > 0)} {
                                $self sendto $size
"$Timestamp:$Origin_ID:$Cond:OUT:$x_pos_rec:$y_pos_rec:$Origin_ID:$CondList" $BROADCAST_ADDR
$sport
                                lappend messages_seen $Timestamp+$Origin_ID
                                set ::TX($Node_adr) [expr $TX($Node_adr) + 1]
                        }
                }

                if {($Direction == "RET")} {
                        lappend messages_seen $Timestamp+$Origin_ID+$ReplyID+$ReplyCond
```

```
                        if {([lsearch $Status $ReplyCond] >= 0)} {
                                $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]
                        }
                }
        } else {
                $ns_ trace-annotate "[$node_ node-addr] received redundant message
$Timestamp+$Origin_ID+$ReplyID+$ReplyCond from $source"
                puts "[$node_ node-addr] received redundant message
$Timestamp+$Origin_ID+$ReplyID+$ReplyCond from $source"
                set ::DISCARD($Node_adr) [expr $DISCARD($Node_adr) + 1]
                #do nothing
        }
}

# ==================================================================
#Sending of mesages
# ==================================================================
Agent/MessagePassing/Flooding instproc send_message {size Timestamp Origin_ID Cond Direction X Y
ReplyID ReplyCond port} {
   $self instvar messages_seen node_
   global ns_ MESSAGE_PORT BROADCAST_ADDR TX
   set ::TX([$node_ node-addr]) [expr $TX([$node_ node-addr]) + 1]
   lappend messages_seen $Timestamp+$Origin_ID
   $self sendto $size "$Timestamp:$Origin_ID:$Cond:$Direction:$X:$Y:$ReplyID:$ReplyCond"
$BROADCAST_ADDR $port
}

# ==================================================================
# 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 {}
}

# ==================================================================
#Set the status of each node
# ==================================================================
$a(0) set Status [list RAIN ICE]
$a(1) set Status [list RAIN ICE]
$a(2) set Status [list RAIN]
$a(3) set Status []
$a(4) set Status []
$a(5) set Status [list ICE]

for {set i 6} {$i < $optNodes} {incr i} {
   $a($i) set Status []
}

# ==================================================================
# set up some events
# ==================================================================
set x_pos [$node_(0) set X_]
set y_pos [$node_(0) set Y_]
```

```
$ns_ at 0 "$node_(0) color gold"
set now [$ns_ now]

$ns_ at 0.03 "$a(0) send_message 10 0.03 0 {RAIN+ICE} {OUT} {$x_pos} {$y_pos} 0 {RAIN+ICE+!}
$MESSAGE_PORT"

# ============================================================================
# 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] "finish"

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

proc finish {} {
        global ns_ tracefd nf RX TX LAST FIRST DISCARD optNodes

        for {set i 0} {$i < $optNodes} {incr i} {
                puts " Vehicle$i sent:              $TX($i) messages"
                puts " Vehicle$i received:          $RX($i) messages"
                puts " Vehicle$i total time:        [expr ($LAST($i) - $FIRST($i))*1000] ms"
                puts " Vehicle$i rejected           $DISCARD($i) messages \n"
        }

        $ns_ flush-trace
        close $tracefd
        close $nf

        exec nam nam-output.nam &
        exit 0
}

puts "Starting Simulation..."
$ns_ run
```

Code 2 Simulator code for "intelligent flooding" with 16 vehicles

APPENDIX C: Replacement code

This code replaces the “intelligent flooding” algorithm in the simulator code to achieve “pure flooding”.

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

# ==================================================================
#Reciveing 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

        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]

        #
        # extract Information from the message
        #
        set Timestamp    [lindex [split $data ":"] 0]
        set Origin_ID    [lindex [split $data ":"] 1]
        set Cond         [lindex [split $data ":"] 2]
        set Direction    [lindex [split $data ":"] 3]
        set x_pos_rec    [lindex [split $data ":"] 4]
        set y_pos_rec    [lindex [split $data ":"] 5]
        set ReplyID      [lindex [split $data ":"] 6]
        set ReplyCond    [lindex [split $data ":"] 7]

        #$ns_ trace-annotate "$Node_adr received {$data} from $source"
        #puts "$Node_adr received $data from $source"

        if {([lsearch $messages_seen $Timestamp+$Origin_ID+$ReplyID+$ReplyCond] == -1)} {

                lappend messages_seen $Timestamp+$Origin_ID+$ReplyID+$ReplyCond

                $ns_ trace-annotate "$Node_adr received {$data} from $source"
                puts "$Node_adr received $data from $source"


                $self sendto $size $data $BROADCAST_ADDR $sport
                set ::TX($Node_adr) [expr $TX($Node_adr) + 1]

                if {($Direction == "OUT")} {
```

```
                        set x_pos [$node_ set X_]
                        set y_pos [$node_ set Y_]

                        $self sendto $size
"$Timestamp:$Origin_ID:$Cond:RET:$x_pos:$y_pos:$Node_adr:Status" $BROADCAST_ADDR $sport
                        lappend messages_seen $Timestamp+$Origin_ID+$Node_adr+Status
                        set ::TX($Node_adr) [expr $TX($Node_adr) + 1]
                        #puts "$Node_adr sent RET"

                        #$ns_ trace-annotate "$Node_adr received {$data} from $source"
                        #puts "$Node_adr received $data from $source"

                }
        } else {
                $ns_ trace-annotate "$Node_adr received redundant message
$Timestamp+$Origin_ID+$ReplyID+$ReplyCond from $source"
                set ::DISCARD($Node_adr) [expr $DISCARD($Node_adr) + 1]

                puts "$Node_adr received redundant message
$Timestamp+$Origin_ID+$ReplyID+$ReplyCond from $source"
                #do nothing
        }
}
```

Code 3 Replacement code for "pure flooding"