CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

A Real-Time Testbed for Distributed Algorithms: Evaluation of Average Consensus in Simulated Vehicular Ad Hoc Networks

Master's thesis in the master programmes: *Computer Systems and Networks* and *Algorithms, Languages and Logic*

ALBIN CASPARSSON, DAVID GARDTMAN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

MASTER'S THESIS 2017

A Real-Time Testbed for Distributed Algorithms: Evaluation of Average Consensus in Simulated Vehicular Ad Hoc Networks

ALBIN CASPARSSON, DAVID GARDTMAN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

A Real-Time Testbed for Distributed Algorithms: Evaluation of Average Consensus in Simulated Vehicular Ad Hoc Networks
ALBIN CASPARSSON, DAVID GARDTMAN

Supervisors: Thomas Petig, Elad Michael Schiller; Department of Computer Science and Engineering. Markus Fröhle, Christopher Lindberg; Department of Electrical Engineering
Examiner: Olaf Landsiedel, Department of Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2017

A Real-Time Testbed for Distributed Algorithms: Evaluation of Average Consensus in Simulated Vehicular Ad Hoc Networks
Albin Casparsson, David Gardtman
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Intelligent transportation systems consist of applications which use communication capabilities of vehicles to solve tasks that require cooperation with other vehicles. One of the possible applications is cooperative positioning, in which vehicles increase the accuracy of their positions by sharing positioning information with each other. Previous research has suggested using average consensus to share this information. Average consensus is a type of distributed algorithm that, in a system where each node performs a measurement of some value, can make all nodes reach agreement on the average of the set of measurements. This thesis evaluates the performance of average consensus algorithms in vehicular ad hoc networks.

Full-scale experiments on vehicular systems are costly, but it is also not necessarily desired to fully simulate a vehicular system. This thesis presents a testbed where we opt to fully simulate the vehicular communication network. The vehicles that are part of the network can be simulated using either virtual nodes or a scaled down physical robot system. An 802.11p wireless network, which has been suggested for vehicular ad hoc networks, is simulated using the ns-3 network simulator. Additionally, some properties that cause the wireless network to be unreliable are simulated.

Furthermore, in this thesis, three average consensus algorithms are implemented with some modifications to account for the properties of vehicular ad hoc networks. These algorithms are evaluated in the created testbed, in order to study their performance in such a network. We observe that consensus converges asymptotically in a simulation of randomly moving nodes, and that the consensus states of the nodes oscillate around the true average when new nodes are allowed to enter the system during consensus. The consensus converges to a state that does not necessarily coincide exactly with the true average, which is to be expected since some packets are lost due the simulated wireless network not being fully reliable. We also demonstrate that performing average consensus on the position of an object can improve the precision of driving in a physical system of moving robots.

Keywords: Average consensus, Vehicular ad hoc networks, Network simulation, ns-3, Intelligent transportation systems, Distributed algorithms

Acknowledgements

We would like to thank the department of Electrical Engineering for providing the hardware and a workspace. We would like to thank our troop of supervisors, Thomas Petig, Elad Michael Schiller, Christopher Lindberg and Markus Fröhle, for steering us in the right direction and helping us make this thesis the best it could possibly be. We would also like to thank Tomasz Proczkowski and Nithin Syriac Kurien for always being helpful and making us laugh at least once a day.

Albin Casparsson and David Gardtman, Gothenburg, June 2017

Contents

List of Figures

List of Tables

1 Introduction

Simulations of vehicular systems are often limited in scale due to their computational complexity, and full-scale testing of such systems is associated with high costs [1]. This thesis presents a simulation platform for vehicular ad hoc networks (VANETs) that can be used for evaluating distributed algorithms. Using this testbed, an evaluation of the performance and applicability of average consensus algorithms in VANETs is performed.

In distributed systems, average consensus is a class of algorithms that are used to let agents or processes in a system reach agreement on the average of a set of values [2]. These algorithms are interesting in systems that contain sensors, such as wireless sensor networks, where nodes can use average consensus to combine their sensor data with data from neighbouring nodes and get a better estimation of the measured quantity.

This chapter presents the motivation for creating the testbed and the contributions made by this thesis. Chapter 2 presents background on wireless networks, intelligent transportation systems and average consensus algorithms. Furthermore, it also describes the systems that are used to create the testbed. In Chapter 3, one can find the details of how the testbed is defined and how it is used to evaluate consensus. Finally, the results of experiments on a few select average consensus algorithms in the simulation can be found in Chapter 4. These results are compared and discussed in Chapter 5, where we also discuss the qualities of the testbed itself.

1.1 Motivation

Autonomous driving is currently a widely researched topic, and has attracted significant interest from companies such as Google, Tesla and Volvo. The implementations in vehicles on public streets have mostly been limited to specific subsets of the complete driving task, such as highway driving, parking, or braking [3]. These tasks use only the sensors that the particular vehicle is equipped with. There also exists research to create simple vehicle awareness messages, which has been shown to be able to prevent accidents [4].

However, there exists a desire to make future autonomous vehicles able to perform tasks that require cooperation with other vehicles. For example, collision avoidance can be made more effective if vehicles can coordinate their localisation through wireless communication [5, 6]. These kinds of systems, where vehicles share their information to create smarter and safer driving, are referred to as intelligent transportation systems [6]. To make these cooperative tasks possible, it is necessary to improve the positioning accuracy, as GNSS is not precise enough in all types of environments.

In open sky areas, GNSS (such as GPS) can provide a positioning accuracy of 5-10 metres [7]. However, the accuracy can be significantly degraded by high buildings that block line of sight between vehicle and GNSS satellite. The accuracy is also degraded when multipathing occurs, which is when signals between receiver and satellite reaches either end-point by being reflected off an object. With the introduction of intelligent transportation systems where vehicles can communicate with each other, it becomes possible to use the communication capabilities of vehicles to improve positioning. This is referred to as cooperative positioning.

Intelligent vehicles can form beliefs about their surroundings by using sensors to measure the positions of other vehicles and objects. By using a protocol where this data is shared among vehicles, each vehicle in the system can improve the accuracy of its own position. However, this is a system with changing topologies, where vehicles may enter and leave the system and the vehicles cannot know in advance which vehicles are in range to receive data from them. Delays and dropped packets also occur in VANETs. Therefore, designing an algorithm to share and update positioning information is a complex task.

Average consensus is an interesting type of algorithm for this task. Consensus algorithms are distributed and can therefore be made more robust when imperfect communication links are used, compared to a centralised solution. There are multiple reasons for why a coordinator/leader is not desired in VANETs. Firstly, all nodes in the system are moving, so the leader might leave the network at any time. Since the network may change, it is impossible to know how many times a packet would need to be forwarded in order to reach the leader. Secondly, since the communication is wireless, the channel around the leader would become congested because of the large amount of communication to it, and some packets would be lost due to interference. Average consensus solves these problems since packets do not need to be forwarded, and each node performs its own calculations [2].

1.2 Related Work

In order to be able to evaluate the properties of average consensus algorithms or other distributed algorithms in practice on vehicular wireless network channels, it is necessary to either create or simulate a network with mobile nodes. However, creating a real VANET using full-scale vehicles is, as mentioned before, associated with high costs [1]. It is also not always desirable to simulate both the network and the nodes, since simulations of moving nodes have to be based on models which often depends on making simplifications of reality [8]. By combining a partly physical and a partly simulated system, some of these problems can be mitigated. Furthermore, by using scaled down units to represent vehicles, such as small robots, it is possible to reduce cost and setup time for experiments [9].

There exists previous research on these types of combined physical and simulated vehicular network systems [9, 1]. Some research has focused on creating standardised scaled-down physical testbeds [8]. There also exists multiple simulators that focus on different aspects of simulating VANETs. Some of them focus on simulating the interaction between moving vehicles [10], while there are also pure network simulators with 802.11p implementations (which is one of the proposed protocols for VANETs) [11, 12].

Work has also been performed on creating simulators for road traffic and integrating them with network simulators. One example is SUMO, an open source traffic simulator that can model road networks and vehicles [10]. Using this simulator, topics such as communication, navigation/route planning, traffic light algorithms and surveillance systems can be researched. For simulation of communication between vehicles or between vehicles and infrastructure, it is possible to connect SUMO to a network simulator that can simulate a VANET, such as ns-3 or OMNeT++, so that the nodes can communicate over a simulated network [10, 13]. Research has also been done to create a more sophisticated mobility model directly in ns-3, without the need to connect the network simulator to another application [14].

Two important aspects of studying algorithms in VANETs are that the nodes are highly mobile, and that link failures can occur through signal interference. The possibility of link failures when evaluating average consensus has previously been studied theoretically [15]. Link failures in average consensus has also been studied in the area of wireless sensor networks (WSNs) [16, 17]. How to implement average consensus for cooperative control has been evaluated as well [18]. Importantly, we have not found an evaluation of average consensus in either simulated nor real VANETs using the 802.11p protocol. However, the effects of switching topologies on consensus, i.e. in networks where nodes are moving, have been widely studied [2, 19, 20].

1.3 Aim

The project consists of three aims: defining a testbed in which distributed algorithms can be evaluated, evaluating average consensus by using the created testbed, and lastly, connecting a distributed system consisting of multiple physical nodes to the system.

A testbed will be specified that uses the ns-3 network simulator to simulate a network that is representative for real-world VANETs. The testbed shall have the capability to connect the simulator with applications running distributed algorithms so that their network communication is piped through the simulator in real-time. The intention is that the testbed can be used as a platform for developing, demonstrating and prototyping distributed algorithms for vehicular networks.

Using the implemented testbed, an evaluation will be performed on how average consensus algorithms perform on network channels representative for intelligent transportation systems. The network simulated in this scenario adds properties that might contradict a priori assumptions made by the algorithms. Therefore, some adjustments to the implementation of the algorithms could be necessary.

In order to demonstrate an application of consensus, a physical system will be created that uses consensus to improve driving in a cooperative positioning scenario. Furthermore, the communication in this network should be piped through the testbed, and the quality of the communication should be affected by the real positions of the robots. As a consequence, ns-3 should receive these positions during simulation so that they may be taken into account when simulating the network communication.

1.4 Limitations

The project will be limited to working with the predefined hardware, and other hardware configurations will not be tested. The robots used are of the type Pioneer DX-3. Cameras together with the GulliView software will be used for positioning and to simulate distance sensors.

This hardware is known to work with the ROS Kinetic framework and alternatives will not be considered. The physical system experiment will only be based on a simplified version of cooperative positioning, since the focus of the experiment is to show that average consensus can improve positioning and that the testbed can be connected to a physical system.

Network simulation will be limited to the ns-3 simulator. We assume that the ns-3 models used, including their default settings, are representative for what the documentation states that they simulate.

1.5 Contribution

This thesis provides a testbed which uses the real-time simulator provided by ns-3 that can be connected to either physical or virtual nodes. All communication between these nodes is then redirected through a simulated vehicular ad hoc network which uses the 802.11p protocol. The thesis also provides an evaluation of average consensus in this simulated VANET.

1 **Real-Time Simulation Testbed:** A customised version of the ns-3 simulator has been created which is configured to simulate a network representative for a VANET. The simulator has been modified to be able to update the node positions—which ns-3 uses for propagation and interference calculations—from an external application in real-time.

- **1.1.** The implemented testbed can be used to evaluate distributed algorithms where the algorithms run in virtual nodes on a host computer.
- **1.2.** A system with physical robots can be connected to the testbed, where the robots communicate over a simulated network and their real positions influence the simulated wireless network connection.

2 **Average Consensus Evaluation in VANETs:** By using the implemented testbed, we investigate the performance of three different average consensus algorithms in a vehicular ad hoc network.

- **2.1.** We propose modifications that allow the evaluated consensus algorithms to work in a representative VANET setting.
- **2.2.** Through experiments, we show that the modified algorithms converge consistently in this setting, even under challenging conditions with changing topologies and dropped packets.
- **2.3.** We observe that the algorithms that set their weights dynamically based on locally observable information generally converge faster than the algorithm which uses a global weight.
- **2.4.** Using a system with moving robots, we demonstrate a simple application of consensus in a cooperative positioning problem. We show that even a simple implementation of consensus can improve positioning.

2 Theory and Background

In this chapter, the theory behind average consensus and vehicular ad hoc networks is presented. The specific consensus algorithms used are also explained. Furthermore, this chapter also contains required background knowledge for understanding the software that is used in the system architecture.

2.1 Average Consensus

In distributed systems, average consensus algorithms (hereafter referred to as consensus algorithms) are used for fusing data from multiple nodes into a common picture [17]. Since nodes in a distributed system do not share any kind of memory, all information in such a system needs to be shared by message passing. In a system where every node performs a possibly noisy measurement of some value, the goal of average consensus is to let the nodes share their measurements and use this data to ultimately agree on a more accurate estimation. A secondary goal is to converge to a common estimation as quickly as possible, while limiting the amount of communication in order to avoid congestion in the network. In this section, a survey of different consensus algorithms is presented and properties such as convergence time are discussed.

A simple version of consensus is introduced by Olfati et al. (section II.C of [2]). This version does not deal with packet loss or link failures but is relevant since it guarantees convergence to the exact average in perfect networks. The equation for how node i updates its consensus state x_i each round is defined as follows,

$$x_i(k+1) = x_i(k) + \epsilon \sum_{j \in N_i} (x_j(k) - x_i(k)) \tag{2.1}$$

where k is the current time step, ϵ determines how much the node adjusts its consensus state to those of other nodes in each iteration, N_i is the set of neighbours of node i. The weight ϵ is defined as one divided by the degree of the node with the highest number of neighbours, i.e. the node with the highest degree. This algorithm will henceforth be referred to as simple consensus.

2.1.1 Convergence

In average consensus algorithms, convergence and stability are two important properties. Convergence is the process where all nodes asymptotically reach the average of all proposed values. Furthermore, stability determines whether the nodes oscillate around the average value. If an algorithm is not stable, a node's consensus state might oscillate around the average instead of converging [21]. The performance of an average consensus algorithm can be evaluated by studying the time to convergence and the error of the final converged consensus states compared to the true average of all initial consensus states.

If nodes in a distributed system are to communicate with each other, they require some form of connection to communicate over. However, this connection may be unreliable. This is why average consensus algorithms often include support for packet delays, packet drops and changing topologies [2, 22, 17]. One effect of a dropped packet containing consensus information, is that it may affect the final consensus state of the nodes. Clearly, if a consensus state is not part of the next round of consensus calculations because it was dropped, it will affect the results of that round and thus every round after it as well.

2.1.2 Fast Linear Consensus

The fast linear consensus algorithm aims to improve convergence time by setting individual weights on each connection [23]. In this algorithm, each node updates its consensus state every round differently. The formula for getting the consensus state of node x_i for the next timestep t is as follows:

$$x_i(t+1) = W_{ii}x_i(t) + \sum_{j \in N_i} W_{ij}x_j(t) \tag{2.2}$$

Where W_{ij} is the weight on the connection between node i and j, and N_i is the set of neighbours of node i. The edge-specific weight is what distinguishes this algorithm from simple consensus. These weights can be calculated in a number of ways, one of which is the local-degree weight calculation:

$$W_{ij}(t) = \frac{1}{max\{d_i(t), d_j(t)\}}$$

These weights are determined by which of the nodes x_i or x_j has the highest out-degree, i.e. the number of nodes that it can communicate to. The weight for the node's own consensus state is then set as one minus the sum of all the other weights.

2.1.3 Ratio Consensus

Another average consensus algorithm is ratio consensus; by performing consensus on two values at the same time, the algorithm aims to mitigate the consensus state error that arises when drops occur [19]. The consensus equation is equal to the one used in fast linear consensus (2.2). The weights are defined as one divided by the node's own out-degree for its own consensus state and $1/(1+d_j)$ for other consensus states, where d_j is the degree of the node who sent its state.

The first average consensus, called y, simply performs consensus on the consensus states of each node. In the second consensus being performed, called z, all nodes begin with the same initial consensus state 1. The actual state x_i of node i at timestep t can then calculated as $x_i(t) = y_i(t)/z_i(t)$ The states of consensus y and z at each timestep are sent to neighbours in the same message. If a message is lost, the z consensus will reflect this and will be able to compensate for the lost packet [19].

2.2 Intelligent Transportation Systems

Intelligent Transportation Systems (ITS) can be seen as a subset of the ongoing Internet of things research. While a VANET is simply the network over which a system communicates, ITS can be seen as the actual services and protocols that use the VANET to, for example, enhance safety and increase the quality of cooperative driving [4, 6].

2.2.1 Vehicular ad hoc Networks

One possible way to enable V2V communication in an ITS is the use of a Vehicular ad hoc Network (VANET). An ad hoc network is a decentralised network, where the nodes communicate directly with each other. This is in contrast to centralised networks where there exists some kind of access point to which all nodes send their data and which routes the data for the nodes. A VANET is a mobile ad hoc network in which the nodes' positions in the network can change. One of the message standards proposed for V2V communication is an extension of 802.11 (Wi-Fi) named 802.11p. It has been shown to offer acceptable performance while not being the perfect solution in all situations [24].

In all kinds of wireless networks, VANETs included, packet loss can occur due to interference on the channel. If the waves of multiple different signals interfere with each other at the receiving antenna, the receiver may be unable to extract the individual waveforms that were originally transmitted by the sender, and therefore unable to read the packets [25]. Interference can occur due to the same signal arriving in multiple copies due to multi-pathing. Furthermore, it can also occur due to multi-user interference, which is when transmissions from different nodes overlap on the channel.

It is possible to mitigate multi-user interference in multiple ways, and for 802.11p, CSMA/CA has been suggested. CSMA/CA (Carrier-Sense Multiple Access with

Collision Avoidance) works by checking the current activity on the channel and delaying the transmission of a message if an ongoing transmission is detected or the current signal strength exceeds a certain threshold and no valid message can be found. This is because it would indicate that interference is occurring on the channel [25]. However, some studies have been performed which show that Self-organising Time Division Multiple Access (STDMA) outperforms CSMA in most situations of both high and low congestion [26]. There is therefore some contention over which congestion avoidance method that should be used.

2.2.2 Communication Channel Modelling

A number of different models exist for modelling the loss of signal power between a sender and a receiver in a wireless radio channel. Path loss models account for the loss over distance from the transmitter [27]. A simple model for path loss is one where the attenuation in dB is proportional to logarithm of the distance, and can be configured with constants for path loss, antenna characteristics and the reference distance to the antenna far-field.

Shadowing, also referred to as slow fading, is the loss caused by obstacles in the environment such as walls, buildings and terrain. In order to make a model for this type of loss as accurate as possible, the location, shape, and material properties of the blocking objects would have to be known. Since this information is generally not known, statistical models are commonly used.

When a transmitted signal encounters obstacles in the environment, the interaction between the signal and the obstacles produces additional copies of the signal, called multipath components, which are reflected, diffracted or scattered before reaching the receiver. The components are also subject to different delays. This results in distortion of the received signal, since the signal and its multipath components are summed together at the receiver. This is referred to as multipath fading or fast fading. Similar to shadowing, multipath effects are usually modelled statistically. One such model is the Nakagami fading distribution. This distribution can be configured with a fading parameter, and if this parameter is set to 1, it is equivalent to the Rayleigh fading distribution.

2.2.3 Positioning

Currently, it is possible for vehicles to receive their positioning through different GNSS systems, such as GPS, Galileo or GLONASS. GPS accuracy can be as close as 5-10 meters in perfect conditions [7]. However, in urban environments, this can be greatly reduced. This is because some signals could be blocked or affected by multipathing.

In recently developed work, it has been proposed to apply a consensus algorithm and belief propagation to improve the positioning accuracy of vehicles [7]. This method is referred to as implicit cooperative positioning (ICP). When all vehicles have identified the object using sensors, belief propagation can be used to more accurately determine the location of the feature, which in turn can be used to improve vehicle positioning.

2.3 Software

This section presents the software used in this project. While similar software could be used, all software applications used in this project are distributed under a license such as GNU GPL or BSD-3, which means they are free to download and modify.

2.3.1 Network Simulator: ns-3

The ns-3 network simulator can be used to create realistic simulations of a variety of different types of networks [28]. It can be used for testing how a system would operate in a specific network environment. The simulator is implemented as a discrete event simulator and by default uses a non-real-time scheduler, meaning that after an event has been completed, the simulation time is immediately moved to the next event. In order to be able to interact with real networks outside the simulator, a real-time scheduler is also available. Using the real-time scheduler, it is possible to integrate ns-3 with real-world network stacks so that packets can be transmitted between the simulator and a real-world application. Various models exist in ns-3 that represent protocols, devices etc. from the real world [11]. These models are implemented in modules that are part of the ns-3 software.

2.3.2 Docker Containers

A Docker container is an executable package of software, which includes the application and libraries and settings needed to run it [29]. It allows applications to be run isolated from other applications on the system. Unlike a virtual machine, however, a container does not simulate hardware and does not run a full copy of an operating system. Instead, it uses the OS kernel of the system it is running on. Therefore, a container takes up less space. A docker container is also placed on a virtual subnet. This is done by creating a simulated network bridge to which the docker node is connected.

2.3.3 Robot Control: ROS

The Robot Operating System (ROS) is a framework that can be used for implementing software for robots [30]. It contains libraries that define a common message passing system and standard message formats which simplifies communication between parts of a robotic system. ROS can be connected to the ARIA framework, which is used for communication between a MobileRobots robot and a computer [31]. Using ARIA, control commands can be sent to the robot and sensor data can be read.

2.3.4 Camera-based Positioning: GulliView

GulliView is a software application that in combination with a camera can be used to find the location of AprilTags in the camera's field of view [32]. AprilTags are similar to QR codes and, depending on the used camera's resolution, can be a variety of sizes [33]. Four tags are used to create an origin of the coordinate system and how large one unit in the coordinate system is, and the relative position of another tag can then be calculated by GulliView [32].

3
Methods

This section describes the architecture of the testbed with the simulated network and the physical VANET representative system. How the parts are connected, what each part does and how we evaluate different consensus algorithms using the testbed is explained. It also presents the modifications made to the algorithms presented in Section 2.1 so that they can be run in the environment that is being simulated. Finally, the experiments which were run on the physical and simulated systems are also described.

Two systems for evaluation of consensus were created, a simulated one where average consensus algorithms are run on a single scalar, and a physical one where consensus is performed on the components of a position in 2D space which is used in a representative VANET setting in order to improve positioning. The physical system is to demonstrate a simplified application of consensus and to show how ns-3 can be used with a running system, while the simulation is used to evaluate the performance of each algorithm.

3.1 Testbed Architecture

In order to evaluate algorithms in a scenario which is not affected by the physical system properties, a system was created that consists of virtual nodes communicating over a network connection which is piped through ns-3. The network simulated by ns-3 is configured to represent an 802.11p network, since this standard has been proposed for use in VANETs. Figure 3.1 shows the components that are needed for communication to get from one node to another. Docker containers are the virtual nodes that are running the algorithms. One container is created for each node in the network and is needed in order to pipe the communication through ns-3. The figure shows the setup for only two nodes since additional nodes have the same structure as the ones shown. The guide on how to run this simulation, and the accompanying code, can be found in the thesis project's git repository [1].

The simulation system runs on a single host computer, and nodes in a distributed VANET system are simulated by Docker containers. Every container is connected

[1] `https://bitbucket.org/exjobb17consensus/consensus`

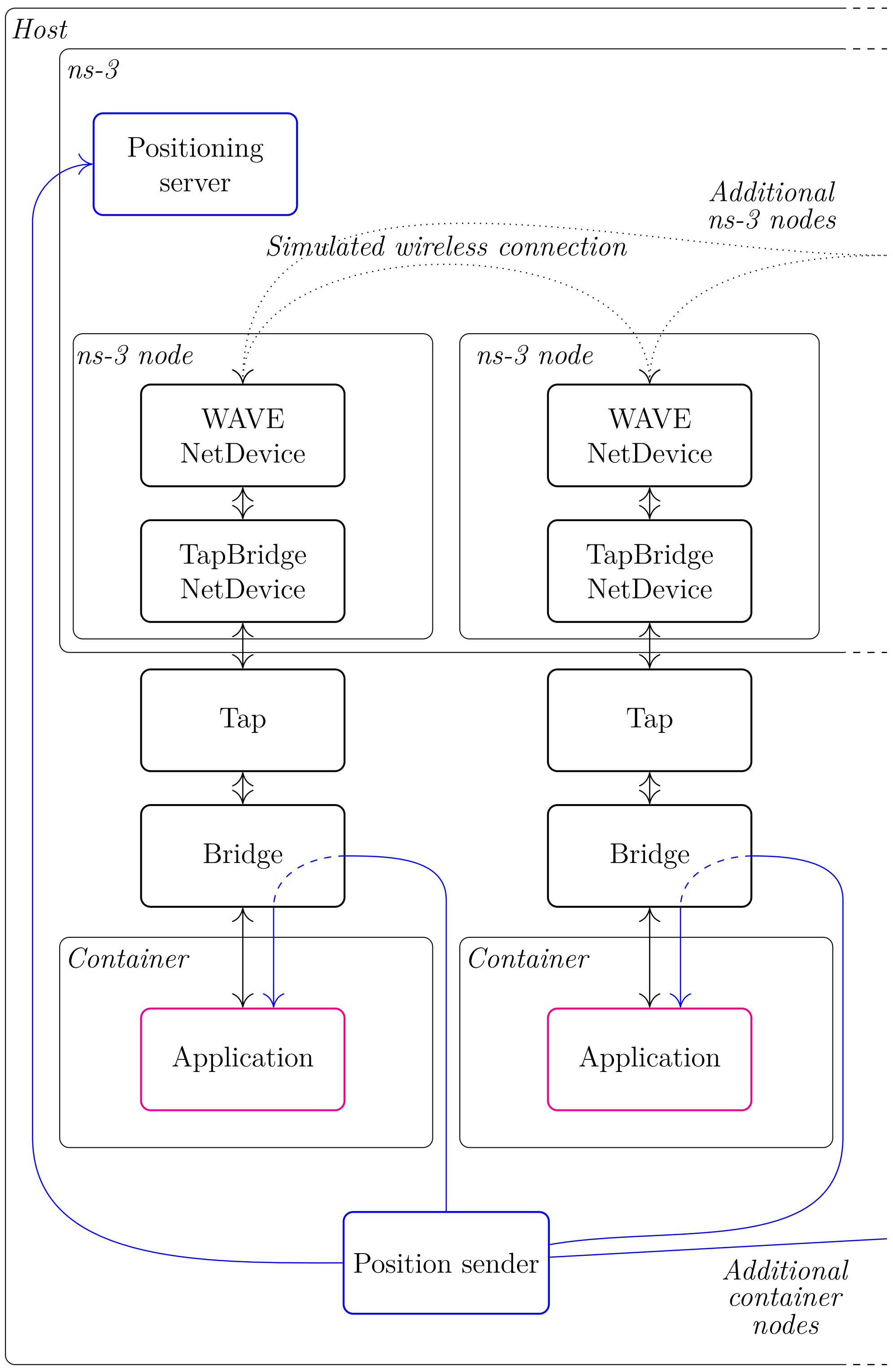


Figure 3.1: Architecture of the simulation system.

to a network bridge created with Docker, which lets applications inside the container communicate with applications connected to the bridge on the host. The position sender is an application which runs on the host and informs both the software running on the virtual nodes and the ns-3 simulation about the current position and velocity of each node by sending messages over TCP. This way, the system makes sure that the virtual nodes and the ns-3 simulation are synchronised and have the same view of where the nodes are.

Each container also has a representation in the ns-3 simulation and a communication channel to the corresponding node in ns-3. The bridge is connected to a tap interface, which the representation of the container in ns-3 is also connected to by the ns-3 module TapBridge NetDevice. This module enables ns-3 to connect and receive and send data through taps. Finally, the data arrives at the WAVE NetDevice. This ns-3 module simulates many aspects of the network, such as scheduling when the packets are to be sent in order to simulate sender delay. It is also the module that simulates the 802.11p network and also the medium access control layer, which is set to use CSMA. Through this channel, the nodes can communicate with other nodes over the simulated network in ns-3. What modules are used in the ns-3 simulation is further specified in Section 3.1.1.

The tap and bridge in Figure 3.1 are virtual devices that are created by the host and are needed for communication between the docker container and ns-3, and between host and docker container. Host to ns-3 position sender communication occurs over localhost. The docker container is put on a different virtual network than the host itself, which means the virtual bridge is needed to connect the two networks together. The bridge is also the docker container's standard gateway and through which the communication to ns-3 occurs. A tap device is also connected to each bridge, allowing ns-3 to send and receive packets over the bridges.

3.1.1 Configuration of ns-3

In the ns-3 simulation, the wireless communication channel and the physical and data link layers of the nodes are simulated. Additionally, ns-3 simulates the positions and movement of the nodes so that it can take into account how the connection between them would be affected by the distance and other nearby nodes whose traffic may cause interference. The simulation makes it possible to evaluate the performance of distributed algorithms in the presence of dropped packets, delays and congestion that would occur in a real VANET. The following table summarises what ns-3 modules are used and what properties of the system they simulate.

Property	**ns-3 module**
MAC layer	Wave
Wireless channel	YansWifiChannel
Physical layer	YansWifiPhy
Propagation delay	Propagation model (ConstantSpeedPropagationDelayModel)
Serialisation delay	YansWifiChannel
Propagation loss of signal power	Propagation model. Log distance model for loss over distance combined with Nakagami-m model for multipath fading
Receiver side drops (network)	NistErrorRateModel, based on signal power and SNIR (signal to noise and interference ratio) at the location of the receiver
Receiver side drops (probabilistic)	Error model. Drops with a specified probability. Modified Wi-Fi model that supports the error model
Position and movement of vehicles	Mobility model
Positioning data from external application	Custom implementation in ns-3

Table 3.1: The ns-3 modules used in the simulations and what properties they simulate.

The WAVE model in ns-3 is used to simulate devices that follow the 802.11p-2010 standard for communication between vehicles. This model uses a 5.9 GHz channel with a bandwidth of 10 MHz and an OFDM rate of 6 Mbps. Additionally, the mac layer has support for communication outside a basic service set, meaning that devices can communicate without setting up some association between them.

The communication channel is modelled using the YansWifiChannel model, and the physical layer of the nodes is modelled based on the YansWifiPhy model. When a packet is sent over the channel by a node with YansWifiPhy, it is received by other nodes (also modelled by YansWifiPhy objects) after the delay which is calculated based on serialisation delay and propagation delay, and with signal power that is reduced according to the propagation loss model(s) that the channel has been configured to use.

YansWifiPhy keeps track of incoming packets and calculates the probability of errors on received packets. It breaks each packet into a a number of chunks, for which it individually calculates the signal to noise and interference ratio (SNIR), using the NistErrorRateModel. Based on the SNIR values, the probability of error for each chunk is calculated, and these probabilities are combined to a final probability for the whole packet. When YansWifiPhy object receives a packet, it uses this probability to decide whether the reception was successful, and if it was, the packet is forwarded to the MAC layer.

For the wireless channel, we use a constant speed propagation delay model where the delay is calculated based on the distance and the speed of light. For propagation loss, we use a log distance model with its default settings, namely an exponent of 3 and a reference loss of 46.6777 dB at a reference distance of 1 m. The log distance model is combined with the Nakagami-m model for multipath fading. For the fading parameter of the Nakagami-m model, the default values of 1.5 for distances smaller than 80m and 0.75 for larger distances are used.

The error model is used to simulate random drops according to a predefined probability or pattern. The model drops packets at the receiver, meaning that the packet is sent over the channel before being dropped. By default, the error model is not supported by the Wi-Fi model. Therefore, the Wi-Fi model was modified to take the error model into account.

The propagation module is used to model propagation loss and propagation delay in the communication between simulated nodes based on their position and movement. The propagation loss model calculates the signal power at the receiver based on the signal power at the sender and the position of the respective antennas. Several propagation models can be used together and the simulation then takes both into account when calculating the receiving power.

The mobility model keeps track of the position and movement of the nodes in the simulation. Using this model, initial positions of the nodes can be allocated, and movement of nodes during the simulation can be predefined. The movement can either be defined in a trace file that is read by the simulator, or by defining constant velocity, constant acceleration or random movement. This model was modified to support reception of new node positions from an external application as explained in Section 3.1.2

While using ns-3, certain performance limitations were encountered. Therefore, the real-time scheduler was configured to use a hard limit so that it keeps track of how far the simulation has fallen behind real-time and generates a fatal error if it falls behind by more than a predefined tolerance threshold. Additionally, the real-time scheduler was modified so that it can log how far behind real-time it is.

3.1.2 Real-time Position Modification of ns-3

When using ns-3 to simulate wireless networks, it can calculate propagation loss and interference between signals. This requires the use of a position model which makes it possible for ns-3 to allocate a position to each node in the simulation. Originally, ns-3 only supports pre-calculated traces for node positions, which removes the possibility of certain kinds of experiments. The most significant limitation is the case where the paths are unknown, or can change dynamically during the simulation. Since the robots in the robot system in this thesis drive based on a controller output, it is impossible to tell ns-3 the exact paths of the robots in advance. It was therefore deemed necessary to modify ns-3 so that it may receive position updates in real-time.

The modification was implemented by creating a TCP server, running on a separate thread, which receives packets that contain the ID of the node whose position should be updated, the 2D position, and the velocity in each axis. Since the simulation originally only changes position in one thread, locks needed to be implemented where positions are altered in the mobility model. It was also necessary to add locks to the reference counting implementation in ns-3. The addition of this new thread and the locks to an otherwise sequential updating of positions might lead to a performance impact in the simulation. The impact depends on how often the original thread checks and updates the positions itself. However, the impact is difficult to calculate; the frequency of updating depends not only on the number of nodes, but also on how many packets are being sent through the simulation. The performance impact of this modification has been studied, and the results of these experiments can be seen in Section 4.4.

3.2 Simulation Architecture

In the simulation system, each Docker container runs one of the nodes that perform consensus. Each consensus node is implemented as a python program which communicates with the other nodes, keeps track of the current consensus state and performs the updating of this state as defined by the consensus algorithms. Consensus is performed on a scalar value, since the result can easily be compared to the exact average value.

The position sender in this case is implemented as a python program which communicates through a TCP connection with ns-3 and each consensus node. It reads positioning updates from a predefined trace file and sends them to ns-3 and the corresponding consensus node, and also sends the command to the nodes to begin consensus.

3.2.1 Simulation Scenarios

This system will be used for two different simulation scenarios, referred to as the *random movement simulation* and the *intersection simulation*. In the *random movement simulation*, all nodes participate in average consensus at all times. By running experiments with moving nodes, the effect of time-varying topologies can be evaluated. The paths of each node is predefined according to a randomly generated trace file. Every second, nodes are instantly moved by up to 21 metres in a random direction. However, there is a restriction on the nodes which makes sure they will stay inside a square-shaped region of 200 by 200 metres.

The *intersection simulation* is designed to represent a possible cooperative positioning situation. The paths of the nodes are defined to simulate vehicles driving in an intersection, as can be seen in Figure 3.2. In order to be able to reuse nodes, they turn around after driving through the intersection so that they may go through it

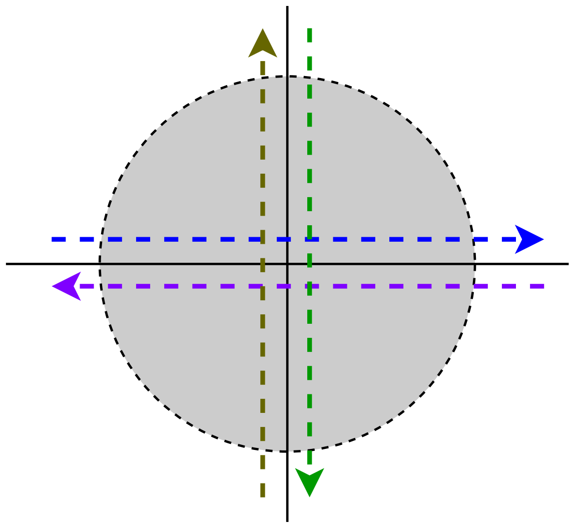


Figure 3.2: How the nodes move in the intersection simulation. The two black lines represent the lanes in which the nodes are moving back and forth. Additionally, nodes are moving in both directions simultaneously in both lanes. The inside of the ring represents the area in which the nodes are able to perform consensus.

again and act as new nodes, which means few nodes can be used to simulate a much larger system. All nodes drive at a speed of 10 m/s and they turn at slightly different times in order to create a randomness to when they reenter the intersection. The roads are 200 metres long and a node is considered being in the intersection when it is at most 70 metres from the middle point. The nodes perform consensus on noisy measurements of the position of an imagined object, which is defined to be in the middle of the intersection. Because of this, not all nodes in the system are performing consensus; only those that are close enough to the intersection can observe the object and perform consensus.

The result of having moving nodes in an ad hoc network is that some modifications to the algorithms are required, which will affect their properties (these modifications can be seen in Section 3.2.2). Furthermore, since the focus of this simulation is to evaluate consensus algorithms, a few surrounding assumptions and simplifications are made. We assume that the object that consensus is performed on is decided on and found by the nodes with the help of another algorithm. There is also no collision avoidance in the intersection; the paths are only for ns-3 to calculate how the network would be affected in this kind of situation.

3.2.2 Implementation of Average Consensus Algorithms

While this thesis uses the three algorithms introduced in Section 2.1, simple, ratio, and fast linear consensus, they are implemented with some modifications. This is due to the random movement and intersection scenarios adding new properties to the system that do not directly fit with the original definitions of the algorithms. The largest of these problems being *a priori* assumptions such as knowledge about the connections in the graph. Moreover, it was also decided that these changes should not focus on being improvements. The goal was to make the implementations simple and with as few assumptions as possible. These implementations, combined with the results gathered during experimenting, could be used as a baseline for future modifications to average consensus. If the algorithms were already modified greatly during experimenting, it would be harder to compare and contrast them to each other.

3.2.2.1 Random Movement Implementation

In the random movement simulation, the nodes have no knowledge about who their neighbours are. The algorithms therefore use UDP broadcast to send their consensus state. Since they do not know how many neighbours they have, they cannot know how long they should wait for messages from all neighbours to arrive. Therefore, an interval-based consensus round is implemented where they receive messages for a certain time interval and then perform consensus on the oldest message from each node that arrived inside that time interval.

The unknown number of neighbours also creates a problem for the algorithms that require an out-degree to calculate their weights (fast linear and ratio consensus). This is solved by having an announcement round before the first round of consensus state messages, where they broadcast a message without state information. From that point on they use the in-degree (the number of nodes they received messages from the last round) from the last round as their out-degree. In this simulation, all nodes perform consensus at all times and almost synchronously since the host commands all the nodes to start at roughly the same time.

3.2.2.2 Intersection Simulation

In the intersection simulation, the same problem arises as in the moving node simulation. We therefore need to use UDP broadcasts, update consensus at a time interval and set out-degree as the last observed in-degree. However, it also adds the problem that only the nodes close enough to the intersection should perform consensus. There will need to exist an initiator that starts the consensus when it reaches the intersection. When a node receives a message, it checks if it can see the object consensus should be performed on, or in other words, it checks if it is in the intersection. The node then starts consensus and continues for as many intervals as it is inside the intersection. Since nodes are joining at random times, it is important to note that the time intervals for the nodes in the system are asynchronous as opposed to the moving node system.

A further modification is made to the simple consensus algorithm in the intersection simulation. In this scenario, the consensus states of new nodes weigh less than the nodes already performing consensus. The argument for this is that nodes that have already performed consensus for any amount of rounds higher than one, should have a more accurate state of the value than a node that just arrived. An added effect is also that the consensus states inside the intersection should be more stable, and not change as much when a new node enters, since the new consensus state will weigh less. This is implemented by raising the ϵ-value for the new node the first two rounds. The effect of this is that the node will weigh the consensus states of the existing nodes more at first.

3.3 Robot System Architecture

An overview of the architecture of the system is shown in Figure 3.3[2]. The idea is that the robots drive in a circle around a centre point with the use of a P-controller that controls on the current position error and angle with regards to the intended circle. The position of the reference object and the robots is measured with a camera hanging from the ceiling and the GulliView software by placing tags on the robots, and another tag placed at an arbitrary location to represent the reference object. This data is used to simulate imprecise GNSS data and precise measurements of the distance between the robots and the reference object. The position of the reference object is then estimated by using these two measurements and is used as the initial estimation. Consensus is then used to improve the robots' estimations of the position of the reference object. The improved estimation is then used by the robots to more accurately determine their own position, which should make it possible for them to more accurately follow the circle around the reference point.

A laptop is placed on top of each robot. This laptop performs all controlling of the robot by using the ROS environment. It sends commands to the robot by using the ROSARIA library and a serial connection. The laptop is also used for average consensus communication to the laptops of other robots and for receiving the position of itself and the robot from GulliView. In order to simulate a network environment representative for a real VANET, communication between the laptops is piped through ns-3. Therefore, all data needs to be sent to a host containing ns-3 and the docker containers. The ns-3 simulation is set up as defined by our testbed (shown in Figure 3.1). The Docker containers are in this case only used to forward the communication between the robots and ns-3.

In order for the robots to reach a Docker node with its own subnet on the host, ip routing has to be performed. An example of how a packet is sent from one robot to another can be seen in Figure 3.4. The laptops and the host computer are all connected to the same wireless network. However, the laptops cannot immediately connect to the docker node which is connected to the simulated network bridge. It is therefore necessary to tell Node$_1$ that for it to send packets to Docker$_1$ it has to route all packets to the Bridge$_1$ network via the host.

The communication between the laptop and docker container is implemented using

[2]The accompanying code for this architecture and how to run it can be found in the thesis project's second repository: `https://bitbucket.org/exjobb17consensus/consensusros`

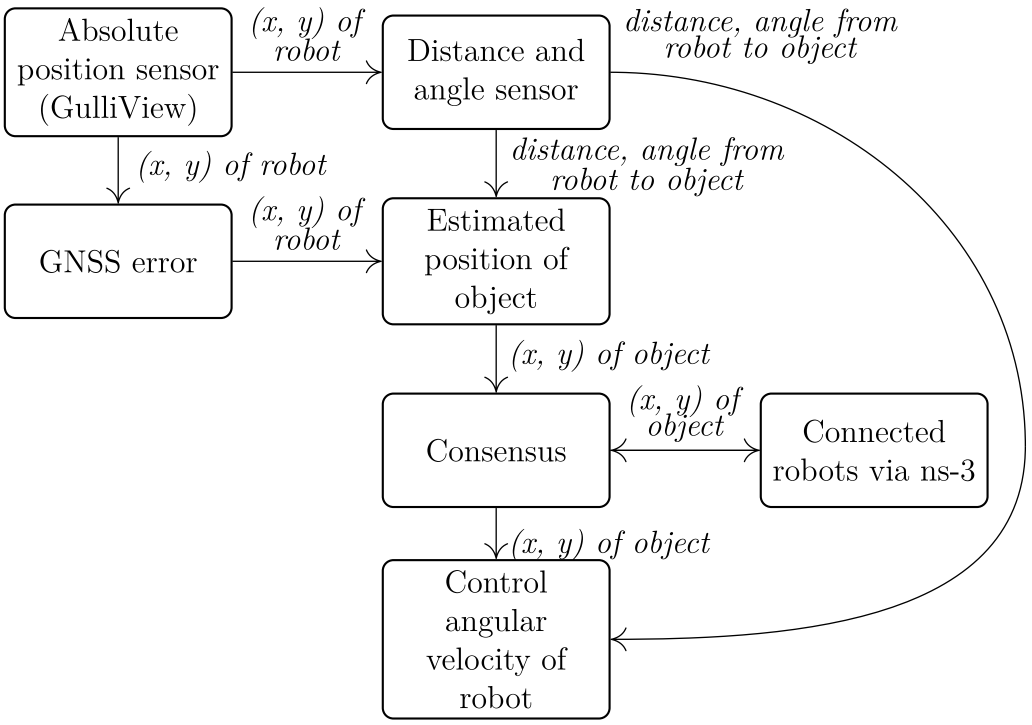

Absolute position sensor (GulliView)
(x, y) of robot
Distance and angle sensor
distance, angle from robot to object
(x, y) of robot
distance, angle from robot to object
GNSS error
(x, y) of robot
Estimated position of object
(x, y) of object
Consensus
(x, y) of object
Connected robots via ns-3
(x, y) of object
Control angular velocity of robot

Figure 3.3: Architecture of the robot system.

TCP to make sure that no packets are dropped outside of ns-3. However, this can lead to delays if a packet is dropped and is required to be resent. There is also a small delay that is added because the data from the robots has to be sent over a network in order to reach the docker nodes (robot, to host through wireless network, to docker). It is therefore important to run this system on a reasonably reliable network, so that the results, which are dependent on communication in the system, have only been affected by the network simulated by ns-3.

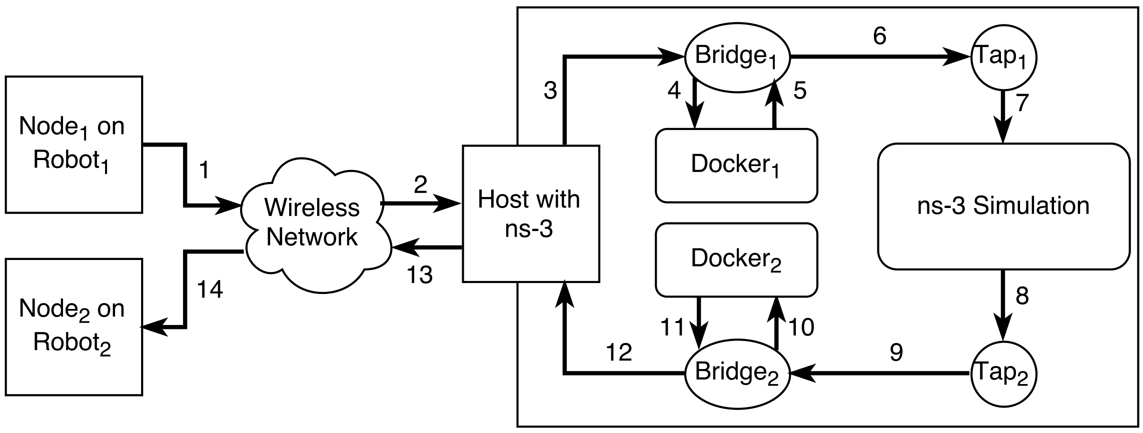


Figure 3.4: The path taken by a packet sent by Node$_1$ with Node$_2$ as destination.

Since both the robots and ns-3 need the positioning data, it was necessary to modify GulliView. The modified version sets up a TCP connection to ns-3 and also broadcasts positions with UDP so that the robots can receive their positions.

3.3.1 Experiment Scenarios

The robot system is run with and without consensus to evaluate how consensus improves the positioning. As with the simulation, it is assumed that the finding and recognition of the object that consensus is to be performed on is done by some other algorithm. In the setting without consensus, the robot calculates an estimated position of the object once every second by the distance given by GulliView and its own skewed GNSS position. During this second it controls itself by estimating its own position using the distance to the object. After one second, it once again calculates an estimation of the reference object position by taking half of its old estimation and half of the new estimation calculated as last time.

In the setting with consensus, the robot calculates the position of the object in the same way. However, during the second it is controlling on the old value, the robot is also performing consensus on the reference object position with the robots that it is able to communicate with. After one second, the robot starts controlling on the results of the consensus algorithm. The new estimation that is sent to the next second of consensus is set as half of the old consensus results and half of the newly performed measurement of the object.

The consensus algorithm used in this system is mostly similar to the modified simple consensus algorithm for the random movement simulation presented in Section 3.2.2: It uses UDP broadcast since it does not know its neighbours, and it performs consensus in a set interval. However, since the robots in this system introduce new values to the consensus on every second, this will cause fluctuations in the consensus states of other nodes. This effect would be significant if the new consensus state would weigh as much as the ones already in the system. Therefore, a small change to the algorithm is made, so that when a node introduces a new estimation during consensus, the node raises its ϵ-value during the following two rounds. This way, the node trusts the estimations of the other nodes more than its own estimation these rounds, which should lead to a more stable consensus estimation.

3.4 Experiments

By creating the simulated and the physical system as specified, experiments on consensus algorithms can be performed. To investigate the performance of consensus algorithms, it is possible to look at how the algorithm converges in different situations and configurations. The effect of using consensus to improve the driving of a robot can be evaluated by looking at the position error from the desired trajectory and the estimation of its position.

In the random movement simulations, the consensus algorithms are compared by finding at what point all nodes in the system have reached a consensus state that is within a certain range from the final average that the nodes converged to. Because of drops and delays, a system with the same set of starting consensus states may converge to different averages. It is also uncertain when the system actually will converge. The simulations are therefore run multiple times and for a longer time than is expected to be needed for convergence. After simulations are finished, the history of consensus states for each node is checked in order to find out when all nodes in that specific simulation reached a consensus state within the range.

The results that are gathered from the simulations are: the number of rounds until all nodes are within the range, a convergence plot which shows the rate of convergence, the error from the achieved convergence compared to the true average of the initial consensus states. The intersection simulation convergence graph is slightly different; since nodes forget the results when they leave the intersection, they start with their initial consensus states and act as new nodes when they rejoin. All nodes will therefore never converge at the same time. However, it will be possible to see if the consensus states of the nodes inside the intersection converge and if new nodes joining in the ongoing consensus converge faster over time.

In summary, the following are the different configurations and experiment scenarios that are used to evaluate consensus algorithms with the testbed.

- **Random movement simulation**
  - Convergence time (number of rounds) and skew of the final consensus state in a changing topology with different number of nodes and added drops
  - Comparison to exhaustive flood with regards to convergence time
- **Intersection simulation**
  - Error over time from the true average in a system where nodes enter and leave the system at different times, with different number of nodes and added drops
  - Comparison to exhaustive flood with regards to error over time from true average
- **Robot system**
  - Comparison of distance error over time from the circle with and without consensus
  - Show how the estimation of the position changes when using consensus

3.4.1 Exhaustive Flood Comparison

The average consensus algorithms are compared to an exhaustive flood. They are compared in regards to convergence time for the two different types of algorithms. The exhaustive flood describes an algorithm in which every new message that is received is forwarded, so that all nodes in a connected network with perfect communication will ultimately have received the value of every node in the network. When running exhaustive flood in the simulation, each node starts by broadcasting a message containing their ID and value. By checking the ID, a node can see if they have received the value before. If it is a new value, they append it to their own message and broadcast this newly combined message. The nodes compute the average of all values they have received when a new value is arrives, $\frac{\textit{Sum of known values}}{\textit{Number of known nodes}}$, but keep sending the list of original values. When and if all values have propagated to a certain node, this computation will yield the true average of all values. At a certain amount of nodes, the message with the gathered values will become too large for a single packet. In order to be able to send the message at this point, it will have to be split up in parts and will be sent in separate packets in a round-robin fashion.

In our experiments, flood does not rebroadcast values immediately when receiving them. Instead, each node only listens for packets for a defined interval, similarly to how consensus is implemented. It then sends out its own value and the whole list of all values it has ever received at the end of each interval. Since new nodes arrive continuously in the intersection simulation, the list of values would grow continuously. At a certain point the list would become unmanageable and even a round-robin sending model would not be able to send all values before the interval finished. It is therefore necessary to, at some point, remove old values from the system. Therefore, we add a lifetime number to each value which is also sent to the other nodes. This lifetime number is decreased each round until it reaches zero, at which point it is removed from the list of values.

Comparing average consensus to an exhaustive flood is important since it shows at what network connectivity, and at which number of nodes, average consensus is faster in a network with limited capacity. Average consensus updates consensus states gradually between neighbours while exhaustive flood requires all data to have reached all nodes. Furthermore, in average consensus, the nodes do not need to forward the consensus states of other nodes. Therefore, the packets are smaller which might mitigate some interference and waiting in the network. By comparing the implemented consensus algorithms to exhaustive flood, it is possible to see *when and even if average consensus is helpful.*

Consensus and flooding are compared on how long it takes for all nodes in the system to reach a value within a specific range from the final value when convergence is complete. For exhaustive flood, the final value is always the true average, since this is the the situation where a node has received the values of all other nodes. For average consensus, as explained before, the final value may not be the true average since a single dropped packet will skew the final consensus state. The simulations are therefore run until both average consensus and exhaustive flooding have reached their final value but the comparisons are made at the point where all nodes in the system have reached the specified range.

4 Results

In this chapter, the results gathered from experiments, as defined in Section 3.4, are presented. For simulations, the time needed for convergence and how much the result is skewed from the average is shown. Due to the large number of experiment scenarios, only some of the data is presented in graphs. These graphs have been selected to give a general idea of the behaviour of each algorithm, and to demonstrate what each scenario looks like. For the robot system, data on how the robots drive is presented in addition to data on the consensus process.

4.1 Random Movement Simulation

This section presents results from the simulation scenario where all nodes move according to a predefined pattern of random movement. Experiments were performed with 20 and 48 nodes, and five experiments were run for each algorithm and situation. This was done in order to get the average time to convergence and error from average, since these varied from run to run. The initial consensus states were defined randomly according to a uniform distribution and adjusted so that the true average was equal to zero. The same predefined movement pattern and set of initial consensus states were used for all experiments with the same number of nodes.

Tables 4.1 and 4.2 present the performance in terms of average time to convergence, measured in number of rounds, and average error from the true average of the initial consensus states for all evaluated algorithms in the random movement simulation, with 0% and 20% added drop chance. These drops are simulated as receive drops and are simulated by the ns-3 rate error module which described in Section 3.1.1. It should be noted that this drop chance is in addition to the modules that create the chance of drop due to interference and propagation fading. Simple consensus requires an ϵ-value and is set to to 1/20 for 20 nodes, and 1/48 for 48 nodes.

	Convergence time	Error from average
Ratio	7.6	0.57
Fast linear	7.8	0.28
Simple	13.0	0.26
Flood	4.0	0.00

(a) 0% added drop chance

	Convergence time	Error from average
Ratio	7.6	0.44
Fast linear	7.8	0.48
Simple	14.8	0.20
Flood	4.8	0.00

(b) 20% added drop chance

Table 4.1: Convergence time in number of intervals and skew from true average for random movement simulation with 20 nodes for the evaluated consensus algorithms.

	Convergence time	Error from average
Ratio	4.2	0.29
Fast linear	5.4	0.20
Simple	9.8	0.40
Flood	3.6	0.00

(a) 0% added drop chance

	Convergence time	Error from average
Ratio	4.6	0.30
Fast linear	5.4	0.22
Simple	12.0	0.26
Flood	4.2	0.00

(b) 20% added drop chance

Table 4.2: Convergence time in number of intervals and skew from true average for random movement simulation with 48 nodes for the evaluated consensus algorithms.

Figures 4.1, 4.2, 4.3 and 4.4 show the consensus update process of the evaluated algorithms in the random movement simulation. For the sake of brevity, only the experiments with 20 nodes and 0% added drop chance are shown. However, the general behaviour of the algorithms is similar when run with 48 nodes or 20% drop chance, even though the number of rounds to convergence and final average is different. The reason why ratio consensus and fast linear consensus wait for one round before updating their consensus states is due to the modifications made to the algorithms when implementing them for distributed simulation (these modifications are explained in Section 3.2.2). One of these modifications being the addition of an announce round to let the nodes find how many neighbours they have. Since simple consensus and flood do not have this requirement, they can begin updating their state immediately.

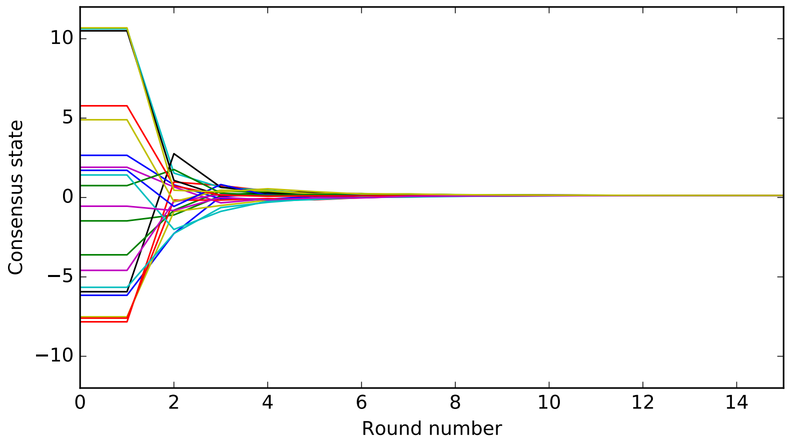


Figure 4.1: Ratio consensus with 20 nodes and 0% added drops.

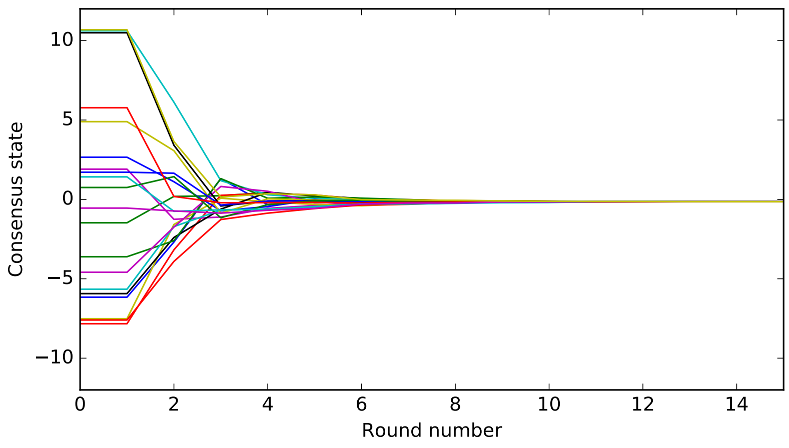


Figure 4.2: Fast linear consensus with 20 nodes and 0% added drops.

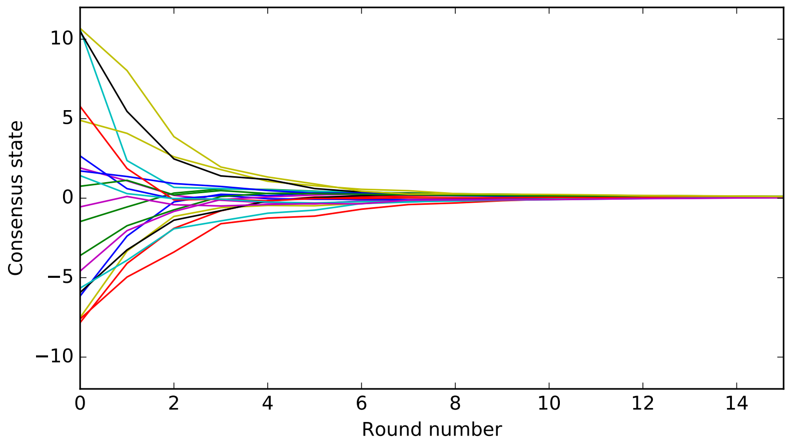


Figure 4.3: Simple consensus with 20 nodes and 0% added drops.

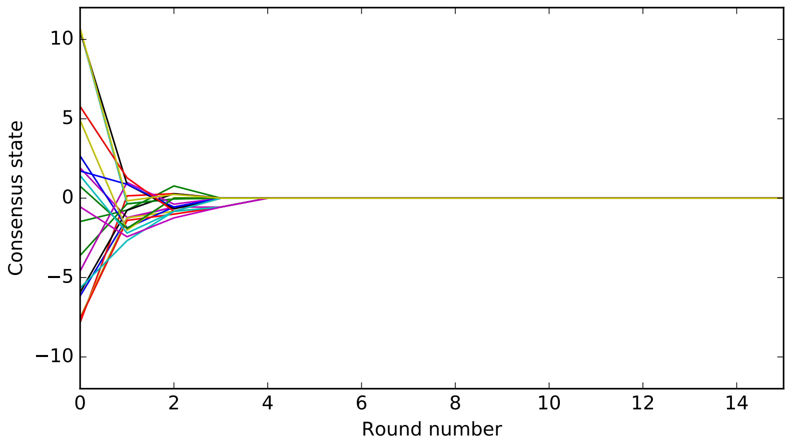


Figure 4.4: Flood with 20 nodes and 0% added drops.

4.2 Intersection Simulation

In this section, we present the results from the scenario where nodes represent vehicles driving through an intersection and participate in consensus only while they are within a certain distance from the centre point of the intersection. We used 20 and 48 nodes for this scenario as well. These numbers and the intersection range defined in Section 3.2.1 translate to an average of 14 and 34 nodes being inside the intersection and performing consensus during simulation. Initial states were randomised according to a uniform distribution and adjusted to make the true average equal to zero.

The ϵ-value for the simple algorithm was set based on the expected number of nodes inside the intersection and the slow weights. When slow weights were used, the ϵ-value was increased by 0.04 for the first interval, and 0.02 for the second interval after a node joins consensus. With 48 nodes, the ϵ-value was set to $0.025 \approx 1/34 - 0.04$, and for 20 nodes, it was set to $0.067 \approx 1/14 - 0.04$.

The average error, presented in Table 4.3, represents how far from the true average the nodes are, from the first consensus update they do when entering the intersection to the last update they do before they leave it. Two different variations of the simple algorithm are included. Simple slow refers to the modified version of the simple algorithm, explained further in Section 3.2.2, where new nodes give a higher weight to nodes already in the system for the first rounds after joining consensus, while simple refers to the version where the weight is constant.

	48 Nodes		20 Nodes	
	0%	20%	0%	20%
Ratio	0.42	0.37	0.58	0.67
Fast linear	0.35	0.35	0.51	0.60
Simple	0.36	0.36	0.65	0.69
Simple slow	0.27	0.30	0.64	0.68
Flood	0.40	0.38	0.86	0.95

Table 4.3: Average error from true average during a 100 second run of the intersection simulation for 20 and 48 nodes with 0% or 20% added drop chance.

Figure 4.5 demonstrates the consensus update process in the intersection simulation. Lines that start after 0 seconds represent new nodes joining the consensus. Intersection simulation plots for the other algorithms can be found in Appendix A.

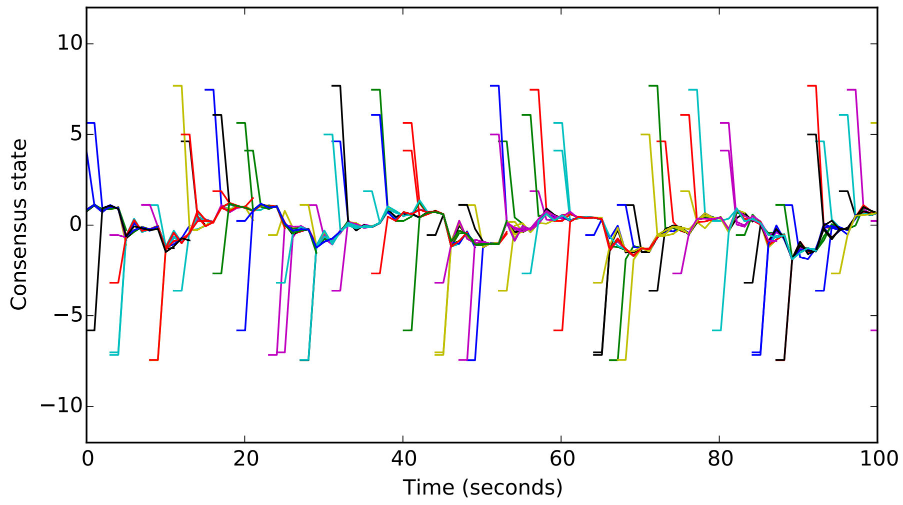


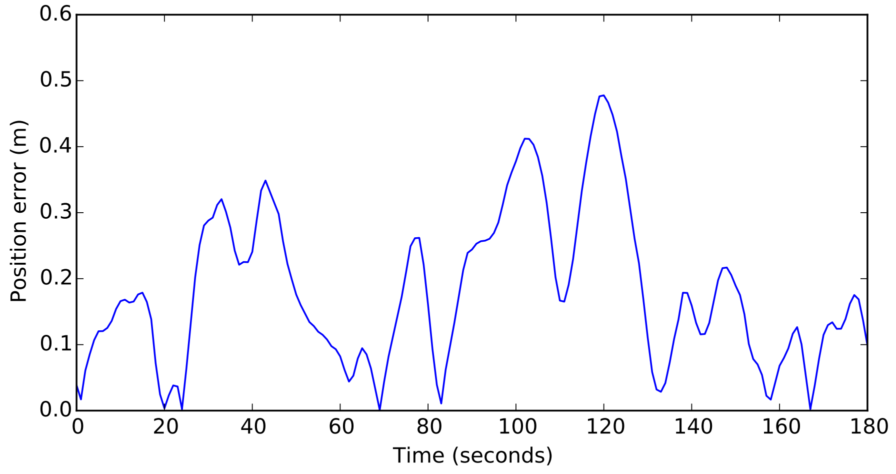
Figure 4.5: Ratio consensus with 20 nodes and 0% added drops in the intersection simulation.

4.3 Robot System

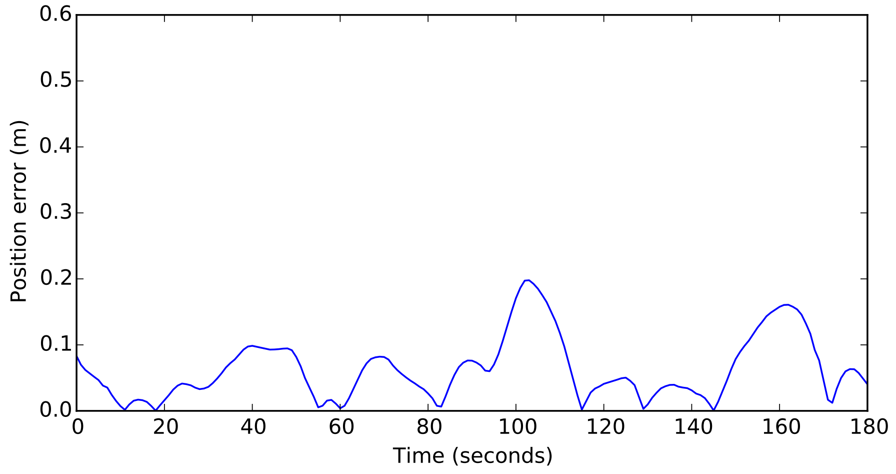
This section presents the results from experiments performed with the robot system. The scenario was as follows. The robots move in a circle around a predetermined point, and use measurement data from a reference AprilTag placed on the floor to orient themselves, as described in Section 3.3. A simulated measurement noise with a Gaussian distribution and standard deviation of 0.8 metres is used to simulate an imprecise GNSS system. The reference tag is initially placed at position (0.79, 0.72) metres. The system is run for 30 seconds before the logging starts. After an additional 90 seconds, the reference tag is moved to (0.21, 0.72). The system then runs for an additional 90 seconds. The ϵ-value is set to 0.3 while a node has been part of the consensus for more than two intervals. As explained in Section 3.3.1, this value is increased for the first two rounds when a new estimation is introduced. For the first interval, it is set to 0.4, and 0.35 for the second.

Each robot logs its estimation of the position of the reference tag, and how far it is from the circle it should follow (position error). The logged position error is based directly on the position data sent by GulliView, rather than the simulated noisy measurement that the robot controller uses. Additionally, the host which runs ns-3 and Docker logs the estimations of the reference object position sent by every node to the other nodes. Experiments were performed with one, two and three robots.

Figure 4.6 shows how the position error evolves throughout the experiment for one of the robots. The position error is defined as the absolute value of the difference between the actual and desired distance between the robot and the middle point of the circle.



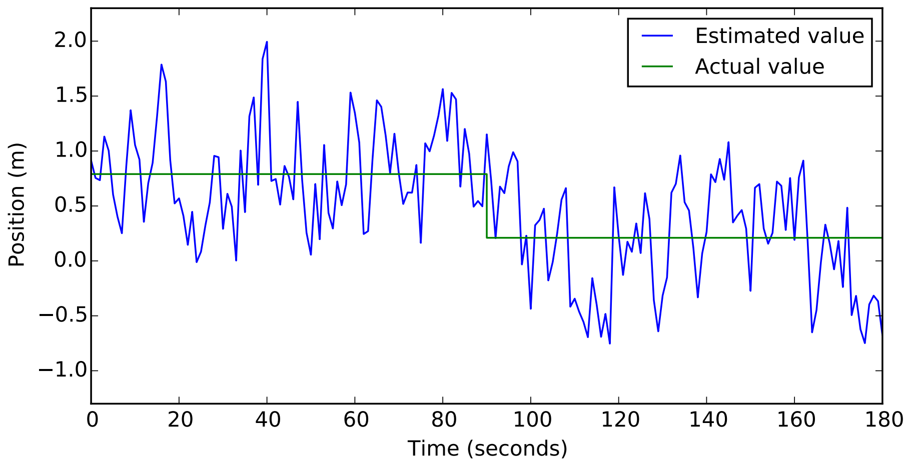
(a) Position error without consensus. Average error is 0.18 metres and standard deviation 0.15.



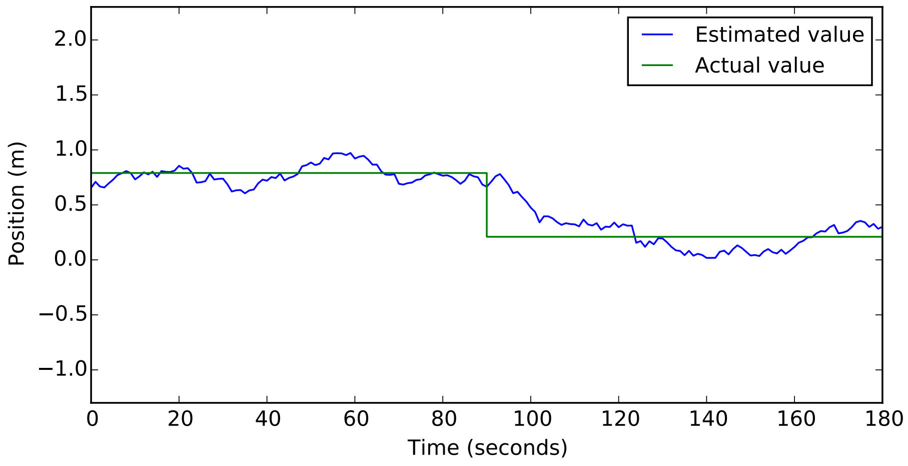
(b) Position error with consensus. Average error is 0.06 metres and standard deviation 0.08.

Figure 4.6: Error from desired position, with and without consensus.

The estimation of the reference object position by one of the nodes is shown in Figure 4.7. Only the estimations in the x dimension are shown, but the error for the y dimension is similar. The values are what is actually used by the controller, and do not show the whole consensus process. This is because the controller of a node gets the estimation from consensus one second after the node last added a new measurement to the consensus.



(a) Estimation without consensus. Average estimation error is 0.39 metres and standard deviation is 0.57.



(b) Estimation with consensus. Average estimation error is 0.11 metres and standard deviation is 0.30.

Figure 4.7: Estimation of reference object position in the x dimension, without consensus and using consensus with two additional nodes.

While the system was also run with two nodes, the corresponding plots have not been included for the sake of brevity. However, the performance for two nodes is generally in between the single node and three nodes with consensus. Both the step from one to two nodes and the step from two to three nodes show notable improvements both to the estimation of the reference object position and the position error of the robots. The average estimation error with two nodes was 0.19 m, standard deviation of the estimation error 0.36, the average position error was 0.14 m, and the standard deviation of the position error was 0.16 m.

Figure 4.8 demonstrates the consensus process when running the system with three nodes. The figure shows the estimations sent by all nodes in each iteration, instead of just the consensus result value used by the controller (as in Figure 4.7).

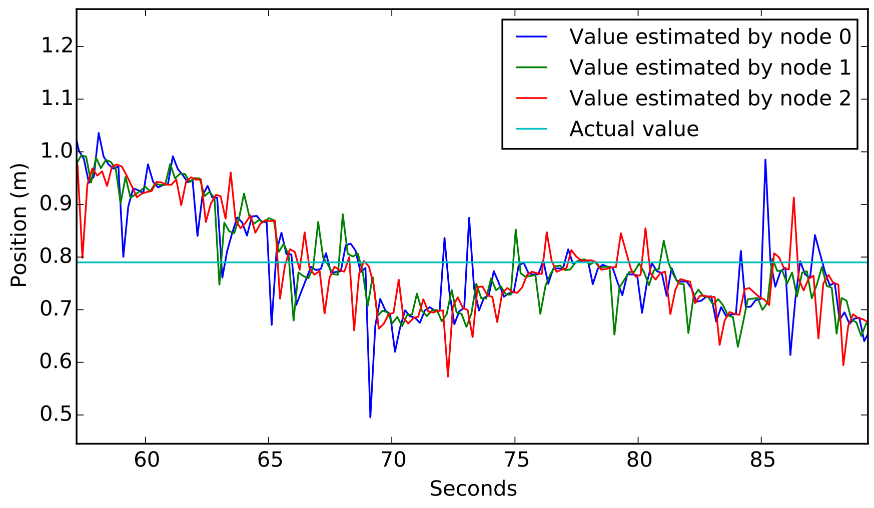


Figure 4.8: Estimation of reference object position in the x dimension by all three nodes in the system.

4.4 Ns-3 Modification Performance Evaluation

These results show how the real-time position server that was implemented in ns-3 affects the performance of the simulation. Table 4.4 shows the internal delay ns-3 suffers from when too many packets enter ns-3 and it cannot keep up. This experiment were performed on a system with an Intel(R) Core(TM) i5-6300 CPU @ 2.40 GHz.

	Average delay		Highest observed delay		Standard deviation of delay	
	Without	With	Without	With	Without	With
20 Nodes	9	21	39	51	4	10
48 Nodes	35	98	81	251	16	49

Table 4.4: Internal ns-3 delay in milliseconds with and without the positioning server for 20 and 48 nodes.

It can be seen that a delay occurs both with and without the positioning server, but increases when the positioning server is active and it slows down the simulation significantly for a large number of nodes.

Because of this, an experiment using a system with an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz was also performed. This experiment did show some promise; the maximum delay was up to 116 ms for 48 nodes with random movement, a round interval of 0.4 seconds and the position sender enabled.

5

Discussion and Conclusion

In this chapter, we discuss the results and what can be derived from them. The advantages and disadvantages of each algorithm in the different experiments is discussed. Problems and important findings about the testbed and ns-3 are discussed. Furthermore, some comments about the robot system are brought up. This chapter also discusses some possible future applications of and extensions to the project. Finally, some conclusions regarding our testbed and consensus in VANETs are made.

5.1 Discussion of ns-3

More modules exist in ns-3 that could be interesting to add in order to better resemble a VANET scenario. For example, there are a number of propagation models that could be used other than logarithmic and Nakagami-m propagation. So for future use of this project, one could investigate whether other models or different parameters for the models used would be more appropriate for the scenario one wants to simulate. Furthermore, while ns-3 simulates path loss due to the distance between sender and receiver and fast fading due to multipath effects, it does not simulate slow fading due to objects blocking the signals.

We are not aware of any model in ns-3 that simulates slow fading statistically for the frequency we use. There exists a building model which can be combined with a propagation loss model such as Okamura-Hata in order to simulate an environment for slow fading. However, none of the propagation models that the building model can be combined with support frequencies of more than 2600 MHz. If such a model were implemented, it could be used to make the intersection simulation more realistic. Interference from signals from nodes outside the system is also not simulated. The possibility to add extra receiver-side drops according to a configurable probability was implemented in order to account for such additional phenomena that are not simulated, and to be able to evaluate algorithms under even more challenging conditions.

5.1.1 Real-time Simulation Performance

The ns-3 performance results (Section 4.4) show that the implemented position modification actually slows down the simulator. However, it also shows an existing delay in the real-time simulator in ns-3 even without the position modification. Our performance experiment also only evaluates how ns-3 is affected when all nodes send packets at approximately the same time. It is likely that the performance impact would be smaller if packets are not sent in simultaneously. Since ns-3 processes events sequentially, it cannot take advantage of a multi-core system. A system with a strong single-core performance should therefore improve the ns-3 simulation performance. Other network simulators exist that can simulate an 802.11p network, such as Omnet++ [12], which can also be configured to interact with real network devices in real-time [34]. Thus, as a future extension of the project, it might be interesting to investigate whether there is a better alternative for real-time simulation than ns-3.

Because of the processing delay created by ns-3, the experiments could not study all aspects of consensus. Inspecting the effects of delays created by the network is impossible, since one cannot determine if the delay was created by features of the simulated network such as CSMA or simply by processing in ns-3. In the random movement simulation, a workaround was used. By setting the interval higher than the observed ns-3 processing delay for a specific number of nodes, we guarantee that all messages that are sent at the start of an interval are also received during that same interval. However, by setting such a large interval, we are unable to observe what happens when packets are received one round later because of delays that should occur (such as CSMA). Consequently, the effect of delays on the algorithms has not been studied at all. We could not guarantee that ns-3 would not delay the packets more than it should and therefore affect the results.

The ns-3 processing delay is also the reason for why flood is implemented so that it does not immediately send a value every time it receives a new one. By making flood only send in intervals, we should get the same delay to occur in ns-3 as for average consensus. If all nodes were allowed to send messages when a new value was received, ns-3 would be delayed even further. It might even be the case that the extra processing delays would make flood converge slower. As a consequence, it was decided that flood should only send at the beginning of the intervals to ensure correctness when comparing to average consensus.

For the intersection simulation, the same workaround of long intervals was used. Unfortunately, the problem of the processing delay cannot be mitigated completely in this experiment. All nodes in this experiment are not necessarily synchronised because they only start consensus when they enter the intersection and receive a message. As can be seen in Figure 5.1, when Node i sends a message at the start of its interval, it will be received by Node j either in Region A or Region B. We know for certain that the packet will be received in A or B because of how long the interval is set in our experiments.

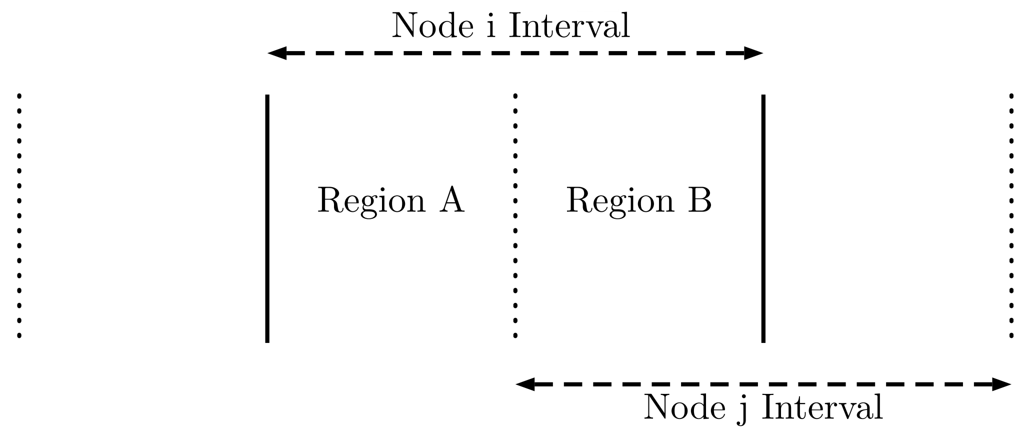


Figure 5.1: Example of asynchronous intervals in the intersection simulation. Node i sends a packet at the start of its interval and Node j receives this packet either during Region A or Region B. The packet could be received even later if the intervals are short.

As explained above, the interval is set so that all messages sent at the beginning of a certain node's interval are received by others during that interval. The probability of the message being received in these regions differ depending on how close the intervals are to being synchronous and how much the packet is delayed by the network. However, the processing delay caused by ns-3 affects the simulation so that the probability for Node j receiving the packet in Region B will be higher. As a result, the algorithms in the intersection simulation will converge slower than they should have, which is important to note when comparing the results of these experiments.

5.2 Discussion of Average Consensus Algorithms

The scenarios in which the algorithms were evaluated represent a somewhat different set of assumptions than the algorithms were originally designed for, since the scenarios were primarily defined to resemble a VANET as closely as possible. Therefore, the nodes do not have knowledge of:

- The maximum degree of the network topology
- Their out-degree (the number of nodes that will receive a broadcasted message)
- Which nodes they are connected to
- When the network topology changes
- When other nodes perform their consensus updates (i.e. they are not synchronised)

The absence of this knowledge means that the experiment scenario is generally more challenging than what the algorithms were created for. Despite this, the evaluated algorithms all converge to a value that is close to the true average of initial consensus states. It can also be noted that the algorithms still converge even if an additional 20% chance for packet drops is added, and convergence speed is only slightly affected.

While the round interval in the simulations was set to one second to account for the performance limitations of ns-3, it could be set lower when applied in a real network. Since several algorithms consistently converge in 5-10 rounds, they can be expected to converge within a few seconds or even less than a second if the round interval can be set lower. The algorithms were observed to generally converge faster if the number of nodes is higher. Since the size of the area on which the nodes are distributed is constant in the simulation, the nodes will have more neighbours if the number of nodes is higher, and the connectivity of the network increases. Thus they get more data in each round.

The simple algorithm is generally slower than the fast linear and ratio consensus algorithms. This is primarily explained by how the ϵ-value (weight) is set. Since this value is defined in a way that depends on the maximum degree of the network topology and this information is not known, we made the assumption that the maximum degree can be at most equal to the number of nodes in the network minus one. However, it is possible that the maximum degree is actually lower which would allow for a higher ϵ-value. The way the ϵ-value is defined does not account for variations in connectivity, so a node with few neighbours will update its consensus state more slowly than a node with many neighbours. If an algorithm similar to the simple algorithm were to be applied in a system where neither the maximum degree nor the number of nodes are known, a new way to define the weight dynamically based on information that each node can observe would need to be developed.

The ratio and fast linear consensus algorithms need to know their out-degree, and therefore it was necessary to implement an announce round before consensus starts up. In the case where a node joins an already running consensus, it will need to passively listen without contributing its own consensus state until it knows its number of neighbours. Therefore, its contribution to the consensus will be delayed by one round, while the simple algorithm and flood do not suffer from this delay.

In the random movement simulation, flood always reaches the true average and converges at a speed similar to the fastest consensus algorithms. The drawback of flood compared to consensus is the amount of data that needs to be transmitted. Since every node needs to transmit not only its own value, but also forward values from other nodes, they will either need to send very large or many packets, increasing the risk of interference. While this did not turn out to be a significant enough problem in the simulations to make it perform worse than consensus, it may become a problem if applied in a network with a larger number of nodes.

Flood was found to deviate more from the true average than the consensus algorithms in the intersection simulation, even though it always converged to the true average in the random movement scenario. This difference comes from the way data from new nodes is handled by our implementation of flood. We deemed it necessary that the nodes do not keep old values forever, otherwise the amount of data they store and send on the network would eventually become very large. Therefore, they disregard old values after a set number of seconds. Because of this, the average can change abruptly when a node receives a new value or disregards an old one. Since flood keeps the original values received, it would be possible to design a more sophisticated way for the nodes to use the data without affecting the protocol.

5.2.1 Our Results and Related Work on Consensus

As mentioned in Section 1.2, some studies have been performed on how average consensus should work in wireless sensor networks. Some of this research has focused on how additive noise on consensus packets affects the results and how to mitigate the effects of noise [16]. We use checksums in our system, which means we notice if a packet has been changed but this also means that we ignore and drop the packet. If checksums were not desired, one could use their solution which almost surely converges, and where the average error can be made arbitrarily small at the cost of convergence rate [16]. However, WSNs are typically considered to communicate digitally with coded packets, rather than using pure analog transmissions; hence, the packet drop model seems more appropriate than additive noise.

We described in Section 3.2.2 how we modified the algorithms, and that the modifications were made with regards to making no assumptions, and not trying to improve but rather make a simple implementation that will work in VANETs. As a result, it should be easier to add some improvements for average consensus from previous research such as [2, 19, 20], since our low amount of modifications should lower the amount of incompatibilities. Nevertheless, conflict is still possible if the improvements require the *a priori* assumptions we removed when implementing.

Hadjicostis and Charalambous [19] present a modified version of the ratio consensus algorithm which can derive the true average even in switching topologies. They assume that no new nodes arrive during consensus, and further study is required to see if the implementation is correct without that assumption. Furthermore, they also assume that every node knows its out-degree, which is also one of the assumptions we removed during our implementation. This could be 'fixed' by adding an echo request just before each consensus round, but is not desired since it would raise the number of packets in the system substantially.

It is also important to mention that delays affect average consensus, and how they should be handled has been previously researched [20, 19, 35]. It has been shown that a modified ratio consensus algorithm converges asymptotically in the presence of changing topologies and communication delays, provided that the delays are bounded [19]. Their findings about the convergence speed without delays appear to agree with ours (5-10 rounds to convergence), and the number of rounds to convergence increases as the length of the delays increases.

Other researchers have also explored consensus in an asynchronous setting, such as our intersection simulation [36]. Similar to us, Mehyar et al. identify problems with the simple algorithm (described in Section 2.1) that make implementation in a real network impractical [36]. They also propose improved algorithms that are robust to asynchronous nodes and changing topologies. Furthermore, their results show that asynchronous consensus convergence in changing topologies is possible.

5.3 Robot System Discussion

The robot system demonstrates an example application of consensus in a cooperative positioning scenario. Since this system was mostly to show how the testbed can interact with the real world and that positions from a physical system can be sent to ns-3 in real-time, a number of simplifications were made. We consider it important to mention some of them here, since our choices affected the results in a few ways.

Because of how the simple consensus algorithm was implemented, where it gives a higher weight to the current consensus value than to new measurements, it made consensus fairly stable even if the new measurements have a significant amount of noise. However, it also slows down the speed with which the estimations are adjusted to movement of the actual object being measured. This can be seen in Figure 4.7, where it takes more time for the nodes using consensus to adjust their estimations to the new tag position compared to the single node which does not use consensus. However, as can be seen in Figure 4.6, this slow update did not make the standard deviation or average error higher than what is was for the position error when running without consensus.

Since we knew the number of nodes in the physical system, we could set the ϵ-value for the consensus algorithm optimally. This would, as mentioned before, not be possible in a real system. However, one might instead use the fast linear or ratio consensus which do not require a predefined global weight value. Furthermore, in the original cooperative positioning paper we based our simplified system on, consensus is only one part of a more complicated system of belief propagation [7]. Moreover, consensus in that system is also not performed on the estimated x and y position but on the components of a node's belief of an object, where the belief is represented as a probability density function. Nodes with a better belief accuracy actually weigh higher in this system, which means adding slow weights would not be as necessary. For systems where consensus is performed on scalar values or similar, the concept of slow weights is still interesting since the intersection results showed that slow weights does improve the average. Slow weights for ratio and fast linear is therefore also interesting, and implementing it should be possible, but would require a different implementation than for simple consensus.

5.3.1 Related Work on Vehicular Testbeds

Simulators such as SUMO that can integrate a simulated network with simulations of vehicles and traffic have previously been implemented. A testbed has also been implemented that connects miniature vehicles with SUMO and a network simulator [9]. However, the network simulator employed (TOSSIM) has a quite simplistic error model compared to what ns-3 uses, and does not use sophisticated propagation models [37]. Our implemented testbed also allows the robots to communicate directly through the network simulator, without involving SUMO.

5.4 Future Work

The current docker node assumes that all data that is being received should be broadcasted by UDP in ns-3. If communication to specific nodes (for example through TCP) would be necessary for a future experiment, a new node would need to be implemented. However, it would make the docker nodes actually need to parse the message sent by the outside node instead of just passing it on which would add a slight processing delay to the communication. It would also mean the system connected to ns-3 would have to create packets differently (it would for example need to specify where it would want the message sent). This would lead to more modifications being needed to the original system, but is probably still desired since only UDP broadcasts limits the usage somewhat.

In Section 2.2.1, it was mentioned that the congestion avoidance method STDMA has been suggested as an alternative to CSMA. Gaugel et al. have studied this further and implemented an STDMA model for ns-3 [38]. Their findings agree mostly with previous studies that show that STDMA is a better alternative than CSMA. However, they see that STDMA degrades in performance to levels of CSMA when packet reception errors are simulated. Nevertheless, they argue that further investigation is required, and by using our testbed, their STDMA model could also be evaluated in real-time.

5.4.1 Security Aspect

Security is another important aspect that needs to be considered when using consensus in VANETs, and for any type of VANET communication. In this project, no security has been added to the packets, such as encryption. Neither is there any way for the nodes to check if incoming packets are from an authentic node. It would therefore be possible for someone malicious, that is in range of the network, to sniff packets and either pretend to be one of the existing nodes or pretend to just have joined the network.

For consensus specifically, a malicious node could send a faulty consensus state that is much higher, or lower, than the true value. A possible way to prevent this attack is to ignore states that are not within a certain range of the node's own state. However, an attacker might also create a large number of fake nodes, whose consensus states are all within the range. This attack would be much harder to detect and defend against. In a paper by Sundaram and Hadjicostis, they study the effects of malicious nodes (byzantine and faulty nodes) and find the theoretical maximum of possible malicious nodes in the system without affecting the result of consensus, which depends on the connectivity of the graph [39]. Furthermore, they also study this effect in a system similar to ours; only broadcasts are used in the network. It would therefore be interesting to use our testbed to evaluate how their theoretical results are affected by a simulated network.

5.5 Conclusion

Through experimenting, it has been shown that our testbed is functional: docker to docker container communication through ns-3 is possible and sending positions to ns-3 has been implemented. However, the positioning modification has also been shown to slow down ns-3 significantly when a large number of nodes are simulated. Nevertheless, we believe that optimisation of the implementation could be made to improve performance. Furthermore, ns-3 already exhibited quite a large delay for a high number of nodes. A more thorough inspection of ns-3 as a whole would be needed in order to conclude if certain elements of ns-3 could be parallelised to improve real-time performance.

The simplified physical cooperative positioning system has shown that the testbed can be connected to a physical system where communication between the nodes is piped through a simulated network. It also demonstrates that sending positioning data to ns-3 in real-time is viable. Since position data can be scaled when sending it to ns-3, it is possible to simulate the effects of communicating on a network over long distances while the actual hardware is just moving a few metres. Piping through ns-3 can be added as described in this thesis and in a way which is transparent to the physical system in itself. The system can therefore act as it did before, without any added limitations by ns-3.

Average consensus has been shown to be viable for VANETs. With some modifications, the average consensus algorithms we have studied can run on a representative simulated network with both changing topologies and drops. The algorithms have been shown to converge consistently, even in an unreliable network. It was observed that the consensus algorithms where the nodes can adjust their weights based on locally observable information (fast linear and ratio) generally converge faster than the simple algorithm that uses a global weight value. Finally, we have shown that consensus works as a tool in cooperative positioning, even when the system is simple.

Bibliography

[1] M. Pahlavan, M. Papatriantafilou, and E. M. Schiller, "Gulliver: A test-bed for developing, demonstrating and prototyping vehicular systems," in *2012 IEEE 75th Vehicular Technology Conference (VTC Spring)*, May 2012, pp. 1–2.

[2] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[3] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 2011, pp. 163–168.

[4] S. Biswas, R. Tatchikou, and F. Dion, "Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety," *IEEE Communications magazine*, vol. 44, no. 1, pp. 74–82, 2006.

[5] H. Wymeersch, J. Lien, and M. Z. Win, "Cooperative localization in wireless networks," *Proceedings of the IEEE*, vol. 97, no. 2, pp. 427–450, 2009.

[6] N. Alam and A. G. Dempster, "Cooperative positioning for vehicular networks: facts and future," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 4, pp. 1708–1717, 2013.

[7] G. Soatti, N. Garcia, H. Wymeersch, M. Nicoli, B. Denis, and R. Raulefs, "Implicit cooperative positioning," unpublished, in review.

[8] C. Berger, "From a competition for self-driving miniature cars to a standardized experimental platform: concept, models, architecture, and evaluation," *arXiv preprint arXiv:1406.7768*, 2014.

[9] C. Berger, E. Dahlgren, J. Grunden, D. Gunnarsson, N. Holtryd, A. Khazal, M. Mustafa, M. Papatriantafilou, E. M. Schiller, C. Steup *et al.*, "Bridging physical and digital traffic system simulations with the gulliver test-bed," in *International Workshop on Communication Technologies for Vehicles*. Springer, 2013, pp. 169–184.

[10] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo–simulation

of urban mobility: an overview," in *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation.* ThinkMind, 2011.

[11] ns-3 project, "ns-3 model library," Accessed: 2017-03-21. [Online]. Available: https://www.nsnam.org/docs/release/3.26/models/html/index.html

[12] D. Eckhoff and C. Sommer, "A multi-channel ieee 1609.4 and 802.11 p edca model for the veins framework," in *Proceedings of 5th ACM/ICST international conference on simulation tools and techniques for communications, networks and systems: 5th ACM/ICST international workshop on OMNet++.(Desenzano, Italy, 19-23 March, 2012). OMNeT*, 2012.

[13] C. Sommer, R. German, and F. Dressler, "Bidirectionally coupled network and road traffic simulation for improved ivc analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, 2011.

[14] H. Arbabi and M. C. Weigle, "Highway mobility and vehicular ad-hoc networks in ns-3," in *Simulation Conference (WSC), Proceedings of the 2010 Winter.* IEEE, 2010, pp. 2991–3003.

[15] F. Fagnani and S. Zampieri, "Average consensus with packet drop communication," *SIAM Journal on Control and Optimization*, vol. 48, no. 1, pp. 102–133, 2009.

[16] S. Kar and J. M. Moura, "Distributed consensus algorithms in sensor networks with imperfect communication: Link failures and channel noise," *IEEE Transactions on Signal Processing*, vol. 57, no. 1, pp. 355–369, 2009.

[17] C. Chen, S. Zhu, X. Guan, and X. S. Shen, *Wireless sensor networks: Distributed consensus estimation.* Springer, 2014.

[18] W. Ren, R. W. Beard, and E. M. Atkins, "Information consensus in multivehicle cooperative control," *IEEE Control Systems*, vol. 27, no. 2, pp. 71–82, 2007.

[19] C. N. Hadjicostis and T. Charalambous, "Average consensus in the presence of delays and dynamically changing directed graph topologies," *arXiv preprint arXiv:1210.4778*, 2012.

[20] Y. G. Sun, L. Wang, and G. Xie, "Average consensus in networks of dynamic agents with switching topologies and multiple time-varying delays," *Systems & Control Letters*, vol. 57, no. 2, pp. 175–183, 2008.

[21] R. A. Freeman, P. Yang, and K. M. Lynch, "Stability and convergence properties of dynamic average consensus estimators," in *Decision and Control, 2006 45th IEEE Conference on.* IEEE, 2006, pp. 338–343.

[22] Y. Cao, W. Yu, W. Ren, and G. Chen, "An overview of recent progress in the study of distributed multi-agent coordination," *IEEE Transactions on Industrial informatics*, vol. 9, no. 1, pp. 427–438, 2013.

[23] L. Xiao and S. Boyd, “Fast linear iterations for distributed averaging,” *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[24] Z. H. Mir and F. Filali, “Lte and ieee 802.11 p for vehicular networking: a performance evaluation,” *EURASIP Journal on Wireless Communications and Networking*, vol. 2014, no. 1, p. 89, 2014.

[25] F. Schmidt-Eisenlohr, *Interference in Vehicle-To-Vehicle Communication Networks: Analysis, Modeling, Simulation and Assessment.* KIT Scientific Publishing, 2010.

[26] K. S. Bilstrup, E. Uhlemann, and E. G. Strom, “Scalability issues of the mac methods stdma and csma of ieee 802.11 p when used in vanets,” in *Communications Workshops (ICC), 2010 IEEE International Conference on.* IEEE, 2010, pp. 1–5.

[27] A. Goldsmith, *Wireless communications.* Cambridge university press, 2005.

[28] ns-3 project, “ns-3 manual,” Accessed: 2017-03-22. [Online]. Available: https://www.nsnam.org/docs/release/3.26/manual/html/index.html

[29] Docker Inc., “What is docker?” Accessed: 2017-06-09. [Online]. Available: https://www.docker.com/what-docker

[30] Open Source Robotics Foundation, “Ros core components,” Accessed: 2017-06-09. [Online]. Available: http://www.ros.org/core-components/

[31] Omron Adept MobileRobots, LLC. , “Aria,” Accessed: 2017-06-09. [Online]. Available: http://www.mobilerobots.com/Software/ARIA.aspx

[32] E. Olson and M. Zucker, “Gulliview,” Accessed: 2017-06-09. [Online]. Available: https://bitbucket.org/thpe/visionlocalization

[33] E. Olson, “Apriltag: A robust and flexible visual fiducial system,” in *2011 IEEE International Conference on Robotics and Automation (ICRA).* IEEE, 2011, pp. 3400–3407.

[34] A. Varga, “Omnet++,” *Modeling and tools for network simulation*, pp. 35–59, 2010.

[35] C. N. Hadjicostis and T. Charalambous, “Average consensus in the presence of delays in directed graph topologies,” *IEEE Transactions on Automatic Control*, vol. 59, no. 3, pp. 763–768, 2014.

[36] M. Mehyar, D. Spanos, J. Pongsajapan, S. H. Low, and R. M. Murray, “Distributed averaging on asynchronous communication networks,” in *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC’05. 44th IEEE Conference on.* IEEE, 2005, pp. 7446–7451.

[37] P. Levis, N. Lee, M. Welsh, and D. Culler, “Tossim: Accurate and scalable simulation of entire tinyos applications,” in *Proceedings of the 1st international*

conference on Embedded networked sensor systems. ACM, 2003, pp. 126–137.

[38] T. Gaugel, J. Mittag, H. Hartenstein, S. Papanastasiou, and E. G. Strom, "In-depth analysis and evaluation of self-organizing tdma," in *Vehicular Networking Conference (VNC), 2013 IEEE.* IEEE, 2013, pp. 79–86.

[39] S. Sundaram and C. N. Hadjicostis, "Distributed function calculation via linear iterative strategies in the presence of malicious agents," *IEEE Transactions on Automatic Control*, vol. 56, no. 7, pp. 1495–1508, 2011.

A

Additional Intersection Plots

These plots show the consensus update process of the algorithms in the intersection simulation scenario. They correspond to the results in Table 4.3.

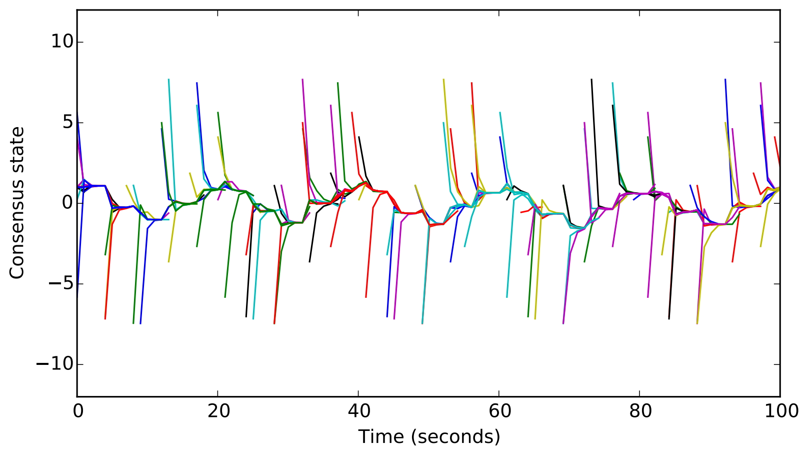


Figure A.1: Simple consensus with 20 nodes and 0% added drops in the intersection simulation.

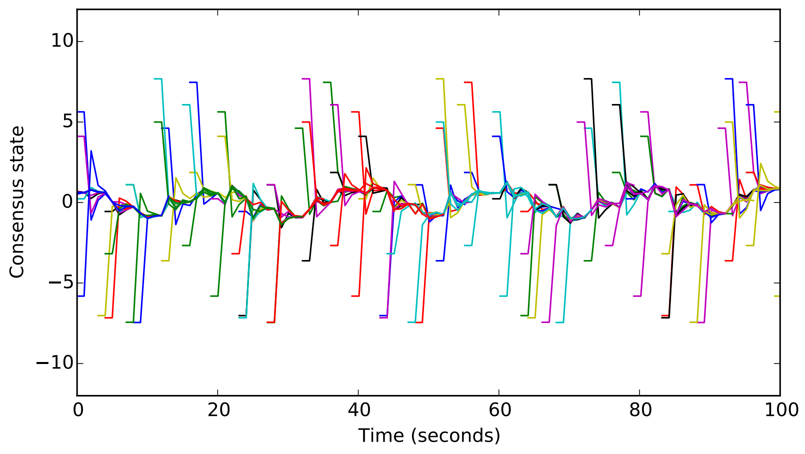


Figure A.2: Fast linear consensus with 20 nodes and 0% added drops in the intersection simulation.

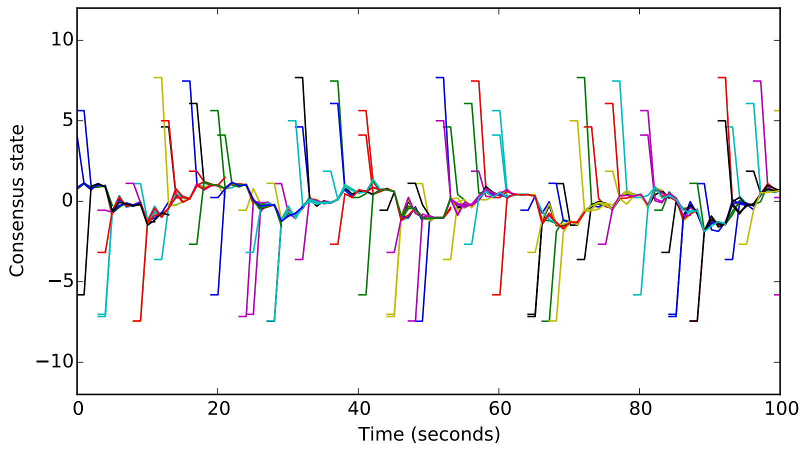


Figure A.3: Ratio consensus with 20 nodes and 0% added drops in the intersection simulation.

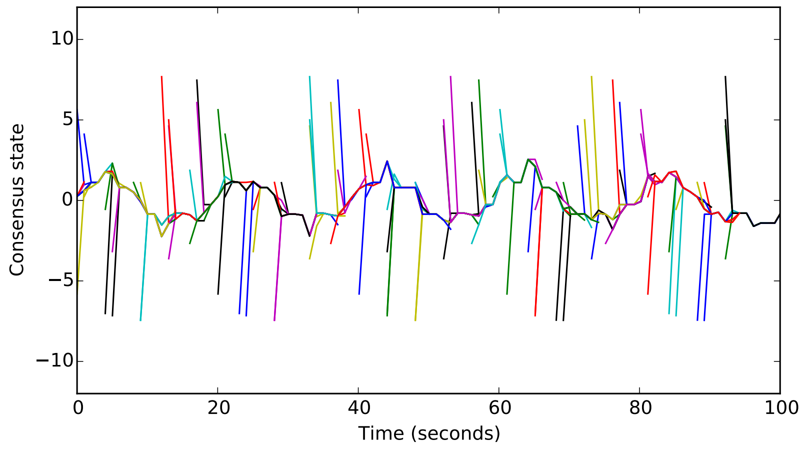


Figure A.4: Flood with 20 nodes and 0% added drops in the intersection simulation.