Ayman Elkadi

Modeling and Simulation of Autonomous Intersection Management Protocols

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 1.5.2015

Thesis supervisor and advisor:

Assoc. Prof. Stavros Tripakis

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Ayman Elkadi

Title: Modeling and Simulation of Autonomous Intersection Management Protocols

Date: 1.5.2015 Language: English Number of pages: 6+46

Department of Communications and Networking

Professorship: Computer Science Code: T-79

Supervisor and advisor: Assoc. Prof. Stavros Tripakis

The advent of autonomous vehicles enables the possibility for autonomous intersection management technologies, which should provide safe, collision-free crossing, while also reducing traffic delays compared to traffic lights or stop signs. This delay reduction is the product of algorithms and communication protocols that make use of increased precision of autonomous vehicles as opposed to human drivers, allowing vehicles to make split-second decisions on their speed while crossing an intersection. Accordingly, deploying such technologies raises concerns about the safe passage of vehicles in an intersection.

In this thesis, two autonomous intersection management protocols were studied and evaluated. First, a decentralized protocol called AMP-IP and developed at Carnegie Mellon University was studied. Based on a simulator developed at University of Texas at Austin, we developed a simulation environment for AMP-IP. Through simulations, we show that our model of AMP-IP satisfies the safe passage of vehicles in a four-way cross intersection and decreases the delay faced by vehicles at the intersection compared to a traffic light model. Then, a centralized intersection management protocol called AIM and developed at University of Texas at Austin was modeled in the UPPAAL model checker. Using statistical model checking we show that our model of AIM has no collisions between vehicles crossing a four-way cross intersection.

Keywords: Intersection Management, V2V, V2I, Simulation, Statistical Model Checking, Autonomous Vehicles

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor, Dr. Stavros Tripakis, for his invaluable mentorship and support throughout my thesis, and for his patience and time spent on guiding my work.

I would like to thank my research group for their feedback and support throughout my thesis.

I would like to thank as well the students service members at the School of Electrical Engineering in Aalto University, especially Jenni Tulensalo, for their guidance and help throughout my master's studies.

Last, but not least, I am truly thankful to my amazing family for their ever lasting love, support and encouragement, especially my parents who I dedicate this thesis to.

Espoo, 1.5.2015

Ayman Elkadi

Contents

Acronyms

AIM: Autonomous Intersection Management
AMP-IP: Advanced Maximum Progression Intersection Protocol
FCFS: First Come, First Serve
API : Application Programming Interface
LICP: Look-ahead Intersection Control Policy
MCP: Minimal Concurrency Protocols
HCP: High Concurrency Protocols
HCPS: High Concurrency Protocols with Slowdown
GPS: Global Positioning System
TCL: Trajectory Cells List
CDAI: Collision Detection Algorithm for Intersections
TIC: Trajectory Intersecting Cell
VIN: Vehicle Identification Number
GUI: Graphical User Interface
TCATL: Trajectory Cells Arrival Time List
V2I: Vehicle-to-Infrastructure
I2V: Infrastructure-to-Vehicle
V2V: Vehicle-to-Vehicle
SMC: Statistical Model Checking

1 Introduction

One of the major causes of death worldwide is road accidents. Each year, approximately 1.24 million people lose their lives due to road accidents [3]. In a plan to cut down road fatalities, the European Commission has embraced a Road Safety program aiming at significantly lowering European road fatalities between 2011 and 2020 [4]. One of the suggested solutions by the program, is the use of Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communications to provide drivers with context-information about their surroundings. This information helps drivers to make better decisions while driving, reducing the risk of accidents. Using V2V and V2I communications on our roads open the door for the development of autonomous driving vehicles, which are able to act autonomously on behalf of human drivers. Moreover, since distracted driving, speeding, and drunk driving are the top three causes of automobile accidents [5], relying on properly designed autonomous vehicles has the potential to reduce road fatalities through eliminating those human errors.

Already several autonomous vehicles have been demonstrated at the DARPA Grand Challenge [3], and by Google [4], and others, which show that autonomous vehicles are no longer a dream. Some reports suggest that the technology could be available for ordinary people within a decade from now [6].

In addition to safety, autonomous vehicles open up the possibility for efficient autonomous intersection management technologies to be deployed resulting in lower traffic delays than current intersection management technologies such as traffic lights and stop signs. These technologies should ensure safe passage through the intersection as well.

1.1 Autonomous Intersection Management

One of the earliest research works in the area of intersection management for autonomous vehicles was established by Dresner and Stone in the University of Texas at Austin in 2004 [7]. They created a reservation-based system which they later called Autonomous Intersection Management or AIM for short, that can be used by autonomous vehicles to cross intersections, outperforming current intersection management technologies such as traffic lights and stop signs [1]. The system is centralized, where vehicles communicate with intersection managers placed at each intersection of the road. Vehicles planning to enter the intersection try to reserve a space-time block in the intersection by sending information to the intersection manager about their time of arrival, velocity of arrival, and their capabilities such as their maximum acceleration/deceleration and size. The intersection manager runs an intersection control policy to compute whether a reservation should be granted or rejected depending on earlier accepted reservations, on a First Come, First Serve(FCFS) basis; and sends the result, accept or reject, back to the requesting vehicle. A vehicle cannot enter the intersection unless it receives a confirmation message about its reservation request, otherwise it stops before entering the intersection and keeps on sending requests until one of them is confirmed.

The FCFS intersection control policy in AIM was changed to a look-ahead intersection control policy(LICP) by another research group [9]. In LICP, the main concept is that if the average intersection delay would be improved by delaying or canceling a reservation, then this will be done even if the vehicle had a higher priority than all the other conflicting vehicles. This is usually done when a vehicle has conflicting trajectories with many other vehicles, so it will be better for all the other vehicles if this vehicle was denied access to the intersection for some time. The LICP policy is shown by the authors to make around 25% average performance improvement on intersection delay than the FCFS policy. The authors also address the issue of fairness in their policy, by allowing vehicles which have waited for a long time to cross the intersection even if this will negatively affect the average intersection delay.

Other research works have proposed decentralized intersection management systems for autonomous vehicles. In those systems, vehicles communicate with one another and cooperatively decide which vehicle should cross the intersection and which should wait, without the help of an intersection manager. Azimi and others from Carnegie Mellon University cooperating with General Motors Company have proposed several V2V autonomous intersection management protocols designed to reduce traffic delays at intersections while providing safe passage through the intersection. The protocols are divided into three categories, Minimal Concurrency Protocols(MCP), High Concurrency Protocols(HCP) and High Concurrency Protocols with Slowdown (HCPS) [10]. This division depends on how vehicles which have trajectory conflicts with other crossing vehicles decide how to cross the intersection. In all of Azimi's protocols, vehicles are assigned priorities based on their arrival time at the intersection, with vehicles reaching the intersection earlier assigned higher priorities than vehicles coming later [11]. Moreover, the intersection area is divided into small cells, where each cell is given a unique identifier. The route the vehicle plans to take along the intersection will be described in the vehicle's messages by an ordered list of cell numbers which the vehicle will occupy along its trajectory [11].

In MCP, messages received from lower-priority vehicles, will be neglected by higher-priority vehicles [10]. On the other hand, a vehicle with trajectory conflicts will come to a complete stop before entering the intersection, and waits until all vehicles having higher priority than this vehicle have crossed the conflicting areas. At which time, the vehicle can cross the intersection [10].

In HCP, in order to increase the intersection throughput, more vehicles are allowed to cross the intersection at the same time [2]. This is done through allowing potentially conflicting vehicles to enter the intersection area. In this case, a lower-priority conflicting vehicle will stop just before a conflicting cell and will wait until the higher-priority conflicting vehicle has crossed the conflicting cell before it crosses the intersection [2].

One of two protocols in the HCP category, is the Advanced Maximum Progression Intersection Protocol [2], or AMP-IP for short, which will be discussed in more detail later in the thesis.

In HCPS, the goal is to decrease the amount of delay faced by lower-priority conflicting vehicles due to coming to complete stops inside or outside the intersection area [10]. This is done through allowing lower-priority vehicles to decelerate in advance, on their way to the intersection entrance, in order to provide higher-priority vehicles with the necessary time to cross the conflicting cell, minimizing a vehicle's chance to come to a complete stop [10].

Milos Mladenovic and Montasir Abbas have developed a decentralized autonomous intersection management protocol similar to Azimi's HCPS [12]. The main difference between both protocol categories is that instead of having a priority policy based on assigning higher priorities to vehicles arriving earlier at the intersection, priorities are assigned based on a number of factors, including the estimated urgency of the journey, vehicle occupancy, vehicle type, and vehicle's capabilities [12]. The estimated urgency of the journey is user defined , and non-monetary priority credits are used to encourage users to set correct urgency levels for their trip [12].

This thesis focuses on intersection management for fully autonomous vehicles, i.e. no human drivers are allowed to cross the intersection. This comes from the fact that the intersection management technologies that we deal with assume that vehicles are able to follow the protocols to high degrees of accuracy, with only small margins of error. This is not possible when humans are in control of driving. Naturally for the same reason, cyclists and pedestrians are not allowed as well. Nonetheless, some of the systems that were mentioned above propose ways to handle humans. For example, in AIM when intersection managers detect vehicles that are driven by humans, the intersection control policy changes to a traffic light policy [8], and other intersection control policies can be added for pedestrians and cyclists [8].

1.2 Objectives

The main aim of this thesis is to study and evaluate the performance of two autonomous intersection management protocols namely AMP-IP and AIM, in order to find out if and why both protocols result in lower traffic delays than current intersection management technologies, and whether these protocols provide collision-free crossing through the intersection or not.

This was done first through modeling and simulation of both protocols, followed by verification using statistical model checking.

1.3 Contributions

The contributions of this thesis are the following:

First, for the AMP-IP, a model and a simulation environment based on the AIM-Simulator have been developed.

Second, performances of AMP-IP, AIM and traffic light have been compared through simulation tests.

Third, a model for AIM and another model for traffic light have been developed in UPPAAL model checker [30], and statistical model checking [28] has been used to check for vehicles' safe passage along the intersection.

Finally, a traffic replay tool that takes UPPAAL output from simulation queries of the traffic light and the AIM models and show them in the AIM-Simulator has been developed. This helps in visualizing both intersection management models.

1.4 Structure

In this section we have discussed the motivation behind the thesis. We have also given an overview of recent autonomous intersection management publications. Then we have mentioned the objectives, and contributions of the thesis.

In the following section, section 2, we will explain two intersection management protocols that we have studied in the thesis: a decentralized protocol called AMP-IP and developed at Carnegie Mellon University, and another centralized protocol called AIM and developed at University of Texas at Austin.

Then in section 3, we will discuss the simulation environments that we have used to simulate both intersection management protocols, and the simulation results we have obtained.

In section 4, we will give a brief overview of statistical model checking, delving into why we have used it to check for collisions between vehicles crossing a 4-way cross intersection using different intersection management protocols. Moreover, we will give a brief overview of the UPPAAL model checker, an integrated software tool environment that we have used for modeling and verification of those protocols.

In section 5, we will explain our traffic light model in UPPAAL, the first intersection management policy we have modeled in UPPAAL. Then we will discuss the verification results we have obtained.

Then in section 6, we will explain our AIM model in UPPAAL, and the verification results we have obtained.

Finally, we will conclude and mention the future work in section 7.

2 Autonomous Intersection Management Protocols

In this section we will describe two autonomous intersection management protocols that we have studied and evaluated in this thesis. For each protocol, we start by mentioning the devices required to be installed in vehicles to allow the protocol to work, the wireless communication standards used, states and messages of the protocol, then finally we discuss the structure and operation of the protocol.

2.1 Advanced Maximum Progression Intersection Protocol (AMP-IP)

As we mentioned in the previous section, AMP-IP [2] is part of a family of distributed V2V autonomous intersection management protocols developed by Azimi and others from Carnegie Mellon University cooperating with General Motors Company, to reduce traffic delays at intersections while providing safe passage through the intersection.

2.1.1 Devices and Wireless Standards

In AMP-IP vehicles are all equipped with Global Positioning System (GPS) devices and access a digital map database to provide them with context information about themselves such as their position, velocity and heading at any point of the road and at any time, and about their environment such as road and lane information. Vehicles use the IEEE 802.11p standard [13] for communication between vehicles. Each vehicle broadcasts safety messages, defined by SAE's J2735 standard [14]; at 10Hz containing information about the route it plans to take along the intersection.

2.1.2 Messages and States

Each vehicle goes into three different states along its path, with each state the vehicle sends a different type of safety message. The different states are:

1. Intersection-Approach: A vehicle enters this state when its distance to the next intersection is smaller than a certain value. In this state the type of safety message broadcasted by a vehicle is called an ENTER message, announcing its approach to the intersection.
2. Intersection-Enter: A vehicle enters this state and leaves the previous state when the vehicle enters the intersection area. In this state the type of safety message broadcasted by a vehicle is called a CROSS message, announcing it has already entered the intersection.
3. Intersection-Exit: A vehicle enters this state and leaves the previous state when the vehicle leaves the intersection. A vehicle leaves the intersection when it passes the intersection boundary on the exit side with a certain distance. In this state the type of safety message broadcasted is called an EXIT message, announcing it has left the Intersection.

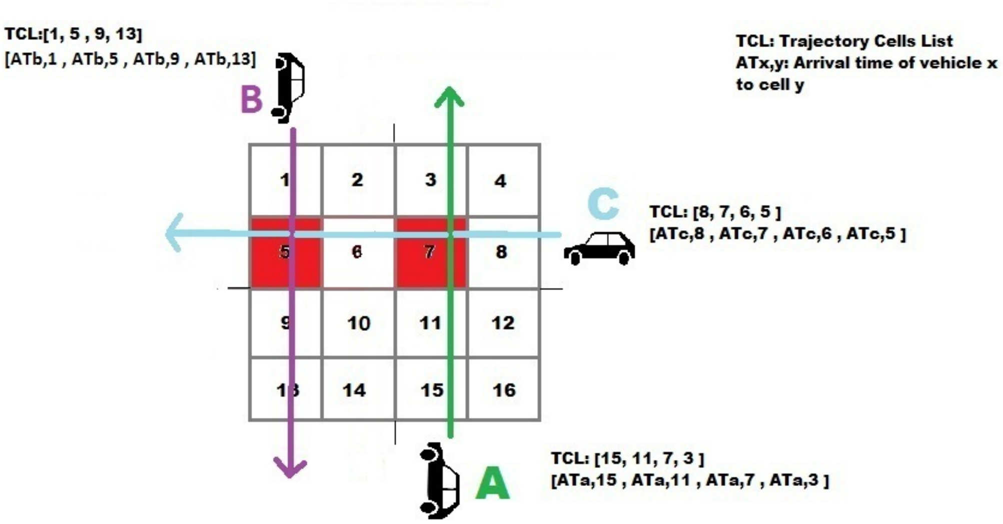

TCL:[1, 5 , 9, 13]
[ATb,1 , ATb,5 , ATb,9 , ATb,13]
B
TCL: Trajectory Cells List
ATx,y: Arrival time of vehicle x to cell y
1
2
3
4
C
TCL: [8, 7, 6, 5]
[ATc,8 , ATc,7 , ATc,6 , ATc,5]
5
6
7
8
9
10
11
12
13
14
15
16
TCL: [15, 11, 7, 3]
[ATa,15 , ATa,11 , ATa,7 , ATa,3]
A

Figure 1: Scenario in AMP-IP – extension of Fig.1 of [2]

For a list of safety message fields inside each safety message type, refer to Appendix A.

2.1.3 Structure and Operation

In order to describe the trajectory of the vehicle along the intersection using few bits (processed quickly by receiving vehicles), the intersection area is divided into cells, where each cell is given a unique identifier. So the route the vehicle plans to take along the intersection will be described in the safety messages by an ordered list of cell numbers which the vehicle will occupy along its trajectory. The list is updated periodically while the vehicle is crossing the intersection. This list is called Trajectory Cells List (TCL).

In addition to the TCL which only gives spatial information about the trajectory of the vehicle, additional temporal information are included in the safety messages as well. These are the estimated arrival time of the vehicle to each cell in its TCL.

The authors of AMP-IP have proposed a Collision Detection Algorithm for Intersections (CDAI) [2] that runs on all vehicles. The algorithm compares the TCL of the receiving vehicle with that of the sender vehicle, and decides whether any common cell along the trajectories of both vehicles exist or not. If a common cell does exist, then the first conflicting cell number called Trajectory Intersecting Cell (TIC) is returned by the algorithm.

Figure 1 shows a scenario in AMP-IP where three vehicles are planning to cross an intersection, highlighting information included in the safety messages of each vehicle about the TCL and the arrival times to each cell in the TCL. As shown in the figure, vehicle C has two conflicting cells with vehicles A and B, with cell 7 being the TIC since it is the first conflicting cell for vehicle C.

The algorithm runs a FCFS priority policy. Vehicles arriving earlier at the intersection are assigned higher priorities than those arriving later. If two or more vehicles arrive at the same time at the intersection, then a Vehicle Identification Number(VIN) uniquely assigned to each vehicle will be used to break the tie.

After the algorithm returns the TIC, the receiver decides whether to stop before the TIC or not depending on its relative priority to the sender and on its arrival at the TIC relative to the arrival of the sender at the TIC.

If the receiver is assigned a lower priority than the sender, it stops right before it enters the TIC unless it can cross and exit the TIC before the sender arrives at the TIC. In order to make sure that the receiver (having lower priority) can enter and exit the TIC before the arrival of the sender (having higher priority) a safety time interval is used.

Figures 2 and 3 show AMP-IP algorithms followed by sending vehicles, and followed by a vehicle B when it receives messages from another vehicle A, respectively. These figures follow closely the pseudo-code of the AMP-IP algorithms found in [2].

2.1.4 Deadlock Avoidance

From the protocol properties, the authors derive this rule: If a vehicle A has a lower priority than a vehicle B, and there exists a common cell along their trajectory, then vehicle A cannot enter the common cell, unless it is able to exit this cell before vehicle B arrives at that cell. Using this rule the authors prove that AMP-IP is free from deadlock [2].

Next we move to the second protocol we have studied and evaluated in this thesis, namely Autonomous Intersection Management protocol, created by Dresner and Stone at the University of Texas at Austin [1].

2.2 Autonomous Intersection Management (AIM) protocol

Unlike AMP-IP, AIM is a centralized protocol, where vehicles communicate with an intersection manager placed at each intersection of the road.

2.2.1 Devices and Wireless Standards

Vehicles are equipped with internal sensors such as GPS devices, to provide them with context information about themselves such as their position, velocity, heading, etc. The vehicles are also equipped with external sensors, such as laser range finders, to provide them with information about their environment such as their proximity to the vehicles in front of them.

Vehicles use DSRC to communicate with Roadside Infrastructure (V2I) [15]. Roadside Infrastructure in AIM are the intersection managers.

2.2.2 Messages and States

When a vehicle is approaching an intersection, it can send 3 types of messages, REQUEST, CHANGE-REQUEST and CANCEL messages.

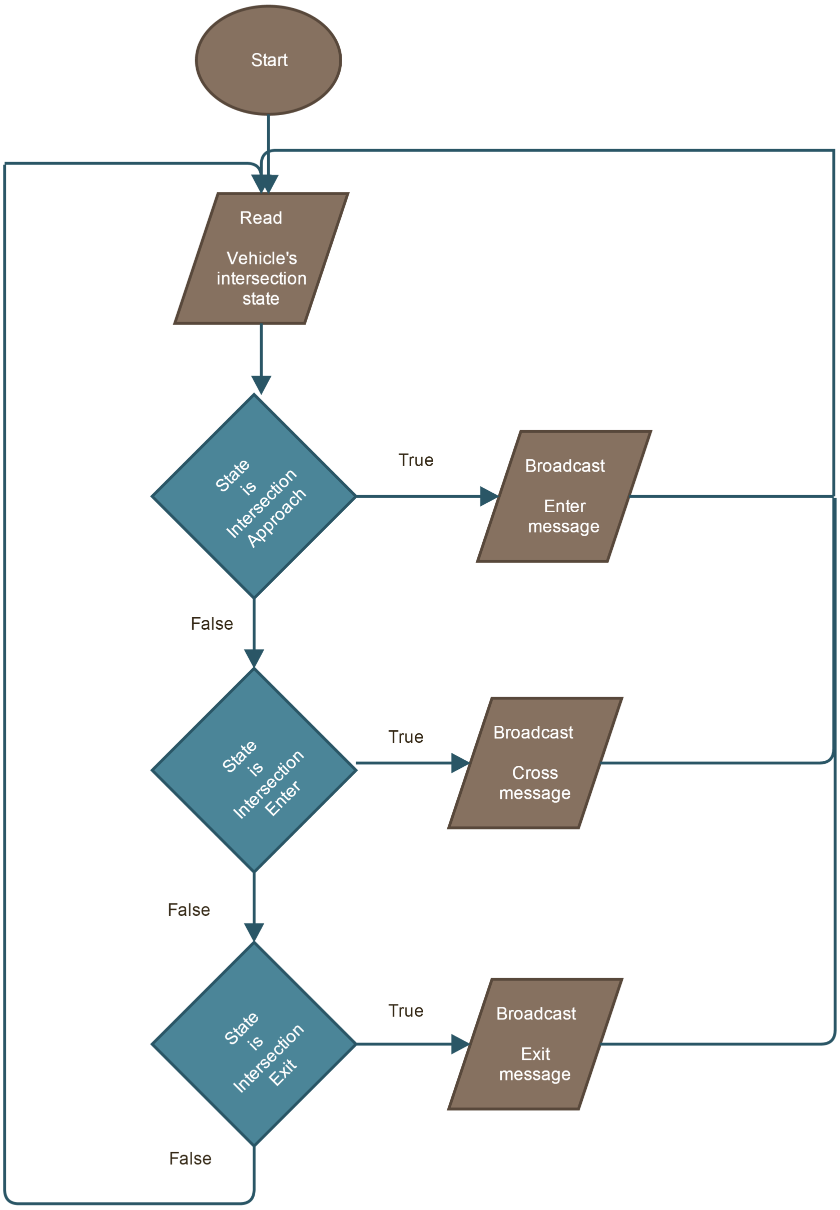
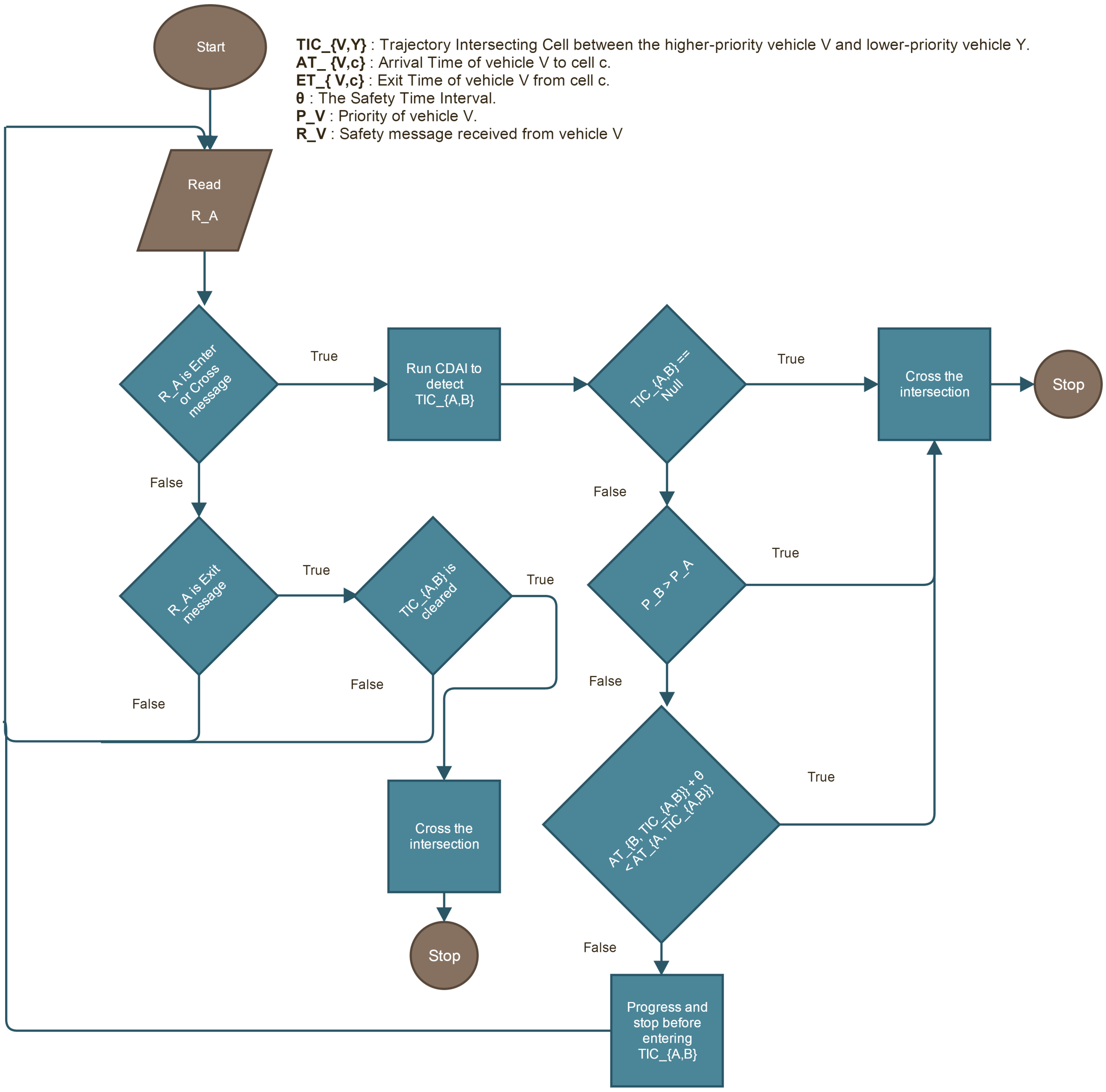


Figure 2: Sender vehicle in AMP-IP



TIC_{V,Y} : Trajectory Intersecting Cell between the higher-priority vehicle V and lower-priority vehicle Y.
AT_ {V,c} : Arrival Time of vehicle V to cell c.
ET_{ V,c} : Exit Time of vehicle V from cell c.
θ : The Safety Time Interval.
P_V : Priority of vehicle V.
R_V : Safety message received from vehicle V

Figure 3: Receiver vehicle in AMP-IP

A REQUEST message is sent when a vehicle has no reservation and would like to make one. It contains vehicle's attributes such as Identification number, size, vehicle dynamics,etc. As well as some reservation attributes, such as, vehicle's arrival time to the intersection, arrival velocity, arrival lane, etc.

A CHANGE-REQUEST message is sent when a vehicle has already a reservation but wishes to change the reservation to an earlier arrival time. If the new arrival time was not accepted by the intersection manager, then the vehicle can keep its earlier reservation, and should arrive at the intersection at the arrival time included in the earlier reservation. The message is exactly the same as the REQUEST message, except for the addition of a unique reservation ID for the earlier reservation that needs to be changed.

A CANCEL message is sent when the vehicle is not able to arrive at the intersection at the supposed time. It contains only 2 fields, a vehicle unique identifier and a reservation identifier of the reservation to be cancelled.

While the vehicle is inside the intersection area it does not send any type of messages.

After exiting the intersection, the vehicle sends a Done message, to inform the intersection manager that it has already exited the intersection.

As for the intersection manager, it replies to a REQUEST or a CHANGE-REQUEST message with a CONFIRM or a REJECT message, and to a DONE or a CANCEL message with an ACKNOWLEDGE message.

The CONFIRM message does not always signal that all the parameters included in the REQUEST or CHANGE-REQUEST messages are accepted. It could signal a counter-offer by containing slightly different attributes included in the message. If the vehicle is not able to meet these attributes, it should send a CANCEL message, otherwise its the responsibility of the vehicle to follow these attributes in order to cross the intersection safely. In addition to the attributes mentioned, the message includes acceleration values and corresponding durations that the vehicle should follow while crossing the intersection.

The REJECT message informs a vehicle that it cannot cross the intersection with the attributes included in the latest reservation request, and that the intersection manager was not able to or did not wish to offer a counter-offer. The message contains only a single boolean field that informs the vehicle about whether it needs to come to a complete stop before entering the intersection or not. This is an indication that the vehicle should not send further requests till it reaches the intersection.

The ACKNOWLEDGE message informs the vehicle of the reception of DONE or CANCEL messages. It contains only a single field which is a reservation identifier of the completed or canceled reservation.

For a list of message fields for each message type, refer to Appendix A.

2.2.3 Structure and Operation

Vehicles planning to enter the intersection try to reserve a space-time block in the intersection by sending data to the intersection manager about their time of arrival, velocity of arrival, and their capabilities such as maximum and minimum acceleration and deceleration values and size. The intersection manager runs an intersection control policy to compute whether a reservation should be granted or rejected depending on earlier accepted reservations, on a FCFS basis, and sends the result (confirm or reject) back to the requesting vehicle. If the intersection manager accepts the vehicle's request by sending a CONFIRM message, this does not mean that all the parameters included in the request are accepted. The CONFIRM message could contain slightly different parameters, which is considered to be a counter-offer. If the vehicle is not able to meet these parameters, for example by slowing down to arrive at a later time to the intersection corresponding to the arrival time included in the CONFIRM message; then it will send a CANCEL message, to cancel its reservation. A vehicle cannot enter the intersection unless it receives a confirmation message about its reservation request, otherwise it stops before entering the intersection and keeps sending requests until one of them is confirmed.

In addition, outside the intersection area, a vehicle is responsible for keeping a safe distance from the vehicle in front of it in the same lane, and it should start to decelerate if it came too close.

In AIM, the intersection area is divided into small cells, called reservation tiles. The number of reservation tiles in the intersection area is the square of the granularity of the policy. Using reservation parameters received from a vehicle planning to cross the intersection, an internal simulation of the trajectory of the vehicle across the intersection is run by the policy. The policy discovers the reservation tiles which will be occupied by the requesting vehicle at each time step of the internal simulation. Accordingly, the policy accepts the request, if during all time steps of the simulation the requesting vehicle did not occupy any reservation tile which had been reserved earlier by another vehicle. Otherwise, the policy rejects the reservation.

Acceleration Profiles

The policy chooses what acceleration profiles the vehicles should follow while crossing the intersection. When a request is sent, the policy first runs an internal simulation with the requesting vehicle having maximum acceleration(until reaching a maximum velocity) while crossing the intersection. If the simulation results in no collisions with earlier reservations, the reservation is accepted; otherwise, another internal simulation is run by the policy with the requesting vehicle having a constant velocity, which equals the Arrival Velocity included in the reservation request. If the simulation results in no collisions in the intersection, then the reservation is accepted; otherwise the reservation is rejected. So all in all two internal simulations are run by the policy. This way the policy provides more chances for the requesting vehicle to obtain a reservation decreasing the possibility of slowing down or stopping, while limiting the number of internal simulation runs, so that the policy does not take a long time in processing a request.

In addition, the policy sets a minimum value on the allowed velocity that a vehicle can take while crossing the intersection. In case the first internal simulation (with vehicle accelerating in the intersection) results in a collision, a second internal simulation would not take place, with the intersection manager replying with a RE-

JECT message; if a vehicle is estimated to arrive at the intersection with a velocity that is lower than the minimum allowed velocity. This is done in order to decrease the delay faced by vehicles at an intersection.

Safety Buffers and Edge Tiles

Due to errors in sensor readings from noise, and time approximations from discretization of time, tiles reserved by one vehicle could be susceptible to being occupied by other vehicles at the same time, leading to collisions. For example, if a vehicle is not able to arrive at the intersection boundary precisely on time for its reservation, and its distance to the intersection is lower than the distance it needs to come to complete stop before the intersection, then this vehicle will enter the intersection jeopardizing the safety of other vehicles crossing the intersection.

To mitigate these deficiencies, the authors introduced the notion of *safety buffers* for vehicles crossing an intersection.

Two types of buffers are used along each other. First type is a static buffer, having a constant size. In that type, the internal simulation run by the policy assumes a vehicle's size larger than it really is. The policy should avoid choosing a static buffer that is so large that it reduces the efficiency of the intersection by introducing unnecessary delays, especially when vehicles with no path conflicts(for example, vehicles traveling in opposite directions) are denied reservations due to having buffer conflicts. Meanwhile, the static buffer should not be so small that it does not counteract the deficiencies mentioned earlier. The second type of buffers is a time buffer. Unlike the first type, this buffer is not static. Its size changes with the vehicle's anticipated motion along the intersection. If the intersection manager assumes a high velocity then it chooses a big buffer size in the direction of motion.On the other hand, if it anticipated low velocity, then it chooses a small buffer size in the direction of motion. In both cases, on the direction orthogonal to the direction of motion, there would be no buffer allocated.

As we mentioned earlier, the AIM policy assumes that outside of the intersection area, vehicles are responsible for keeping safe distance from the vehicles in front of them in the same lane. This is not the case inside the intersection as the intersection manager is responsible for how vehicles will move in the intersection, and it might plan trajectories that look like potential collisions for the vehicles even if they are not. In such a system, a serious safety problem that might arise is when vehicles are exiting the intersection in high speeds to encounter, at a close distance, slow vehicles which have just exited the intersection before them. The former vehicles would then be unable to stop in time to avoid a collision.

In order to solve this problem with minimal effect on efficiency, the authors increase the time buffer for a certain set of reservation tiles. This set is called *Edge Tiles* which are tiles that have at least one side touching the intersection boundary. The reason that increasing the time buffer only for edge tiles solve the problem mention above, is that the problem arises between vehicles which exit the intersection by the same lane, so these vehicles will occupy (at some point) the same edge tiles.

Reservation distance

One issue affecting the performance of the prototype implementation of the AIM system as described by the authors, is when vehicles closer to the intersection have to stop due to losing competition to earlier reservations made by vehicles further away from the intersection in the same lane. This issue arises due to the fact that the system as described until now does not control in what order vehicles in the same lane should get reservations. A vehicle closer to the intersection might send a request with attributes that will lead to (a) conflict(s) in the intersection, while a vehicle planning to arrive later at the intersection at the same lane might reach the intersection at a time where there are no conflicts due to earlier reservations. A vehicle at the back will have to, at some point, cancel its reservation due to not being able to reach the intersection entrance at the arrival time mentioned in the confirmation message because of slowing down (or stopping) when it comes too close to a vehicle in front of it(having no reservation). So the vehicle in front will be able to get a reservation eventually. However, this leads to unnecessary delay.

Since vehicles do not send their position values to the intersection manager, so there is no direct way for the intersection manager to implement a policy which rejects reservations coming from vehicles if there are other vehicles in front which have not been granted reservations. However, as vehicles send their estimated arrival times and velocities to the intersection, so the intersection manager could compute an approximation of the distance between the vehicle and the intersection. The authors call this distance *reservation distance*. This distance is an approximation, as in calculating it the vehicle is assumed to be driving at a constant speed, which could be not true in many cases.

In the following lines, we will explain how the *reservation distance* is computed, and how the policy will use it to increase the likelihood of denying vehicles reservations if the vehicles in front of them do not have reservations.

$$ReservationDistance = V_a * (T_a - T) \tag{1}$$

V_a: Velocity at which the vehicle is expected to arrive with at the intersection.
T_a: Time at which the vehicle is expected to arrive at the intersection.
T: Current time.

The policy is described in the following Pseudo-code:

D_i: Smallest allowed distance for each lane i.
D(R): Reservation Distance computed for every REQUEST R in lane i.

```
$D_i$ = Infinity ;
If (D(R) > $D_i$)
    reject R ;
else
    Process R ;
    If(R is rejected)
```

```
        $D_i$ = min($D_i$,D(R));
    else
        $D_i$ = Infinity ;
```

An example to show how this policy could lead to better performance, is a vehicle stopping at an intersection with no reservation. It has D(R) close to zero since T_a is approximately equal to T. As the vehicle has no reservation(reservation is rejected) so D_i=min(D_i,D(R)), approximate to zero as well. So for all vehicles behind this vehicle in the same lane, D(R) will be larger than D_i leading to rejection by the policy. This is done for each lane separately.

Figures 4, 5 and 6 are flow charts showing how the driver agent, the intersection manager and the FCFS policy in AIM operate, respectively. These figures follow closely the pseudo-code of the AIM algorithms found in [1].

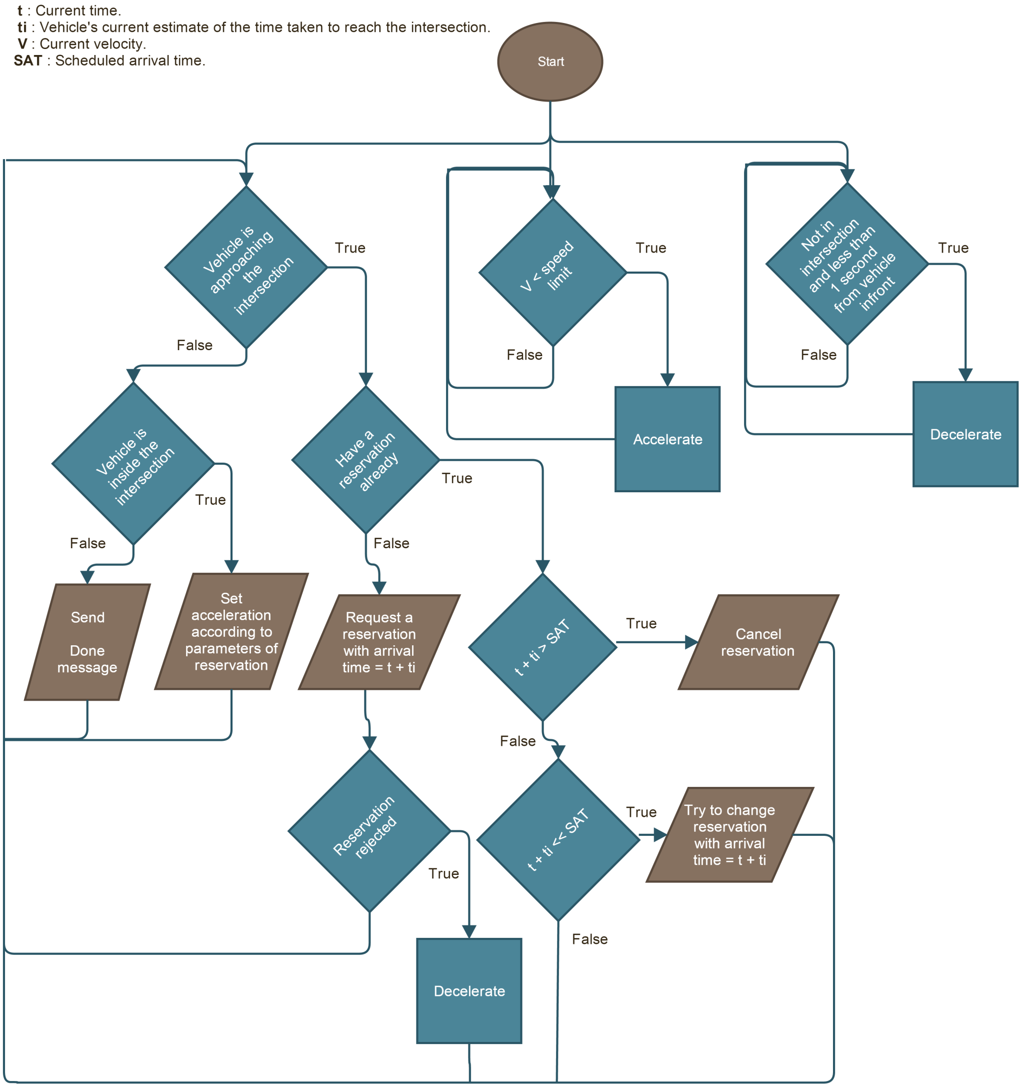

t : Current time.
ti : Vehicle's current estimate of the time taken to reach the intersection.
V : Current velocity.
SAT : Scheduled arrival time.
Start
Vehicle is approaching the intersection
True
False
V < speed limit
True
False
Not in intersection and less than 1 second from vehicle infront
True
False
Accelerate
Decelerate
Vehicle is inside the intersection
True
False
Have a reservation already
True
False
Send Done message
Set acceleration according to parameters of reservation
Request a reservation with arrival time = t + ti
t + ti > SAT
True
Cancel reservation
False
Reservation rejected
True
t + ti << SAT
True
Try to change reservation with arrival time = t + ti
False
Decelerate

Figure 4: Driver Agent in AIM

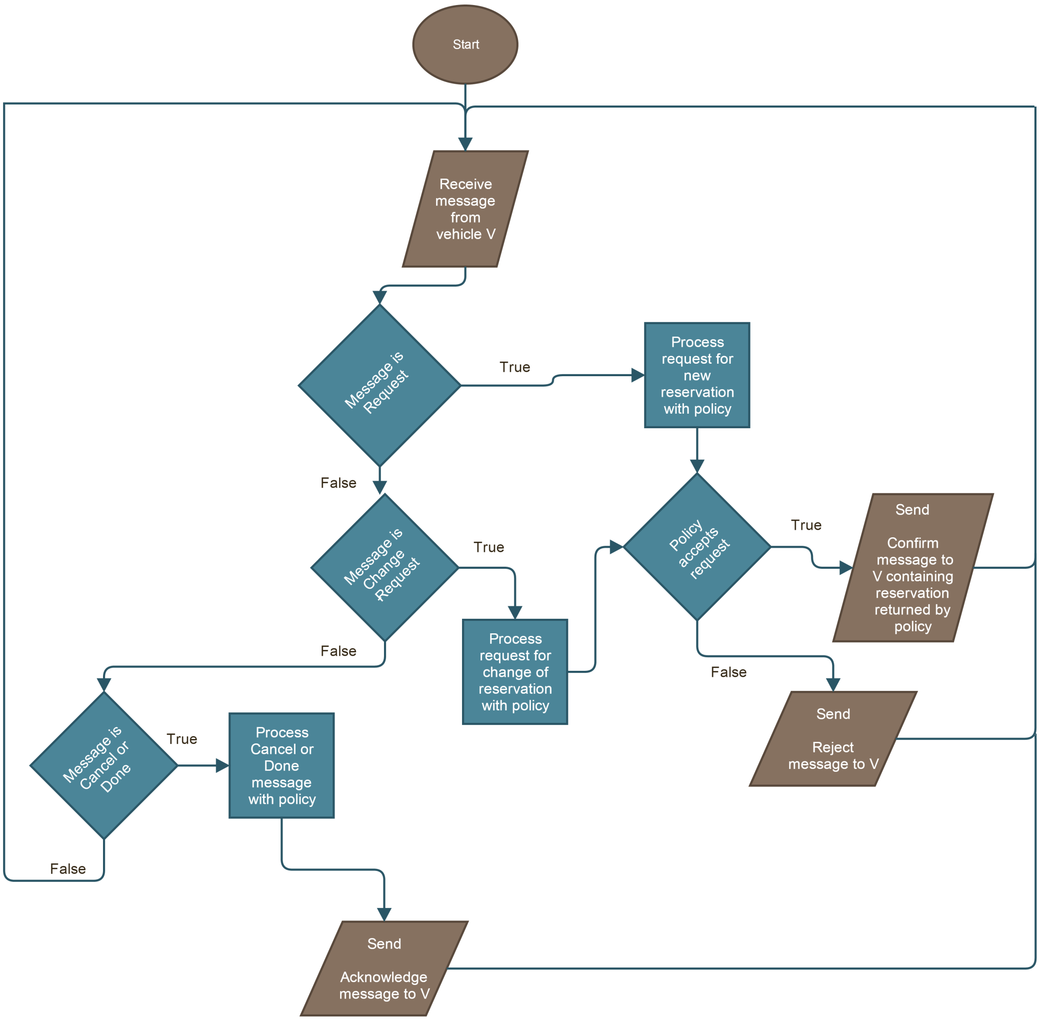
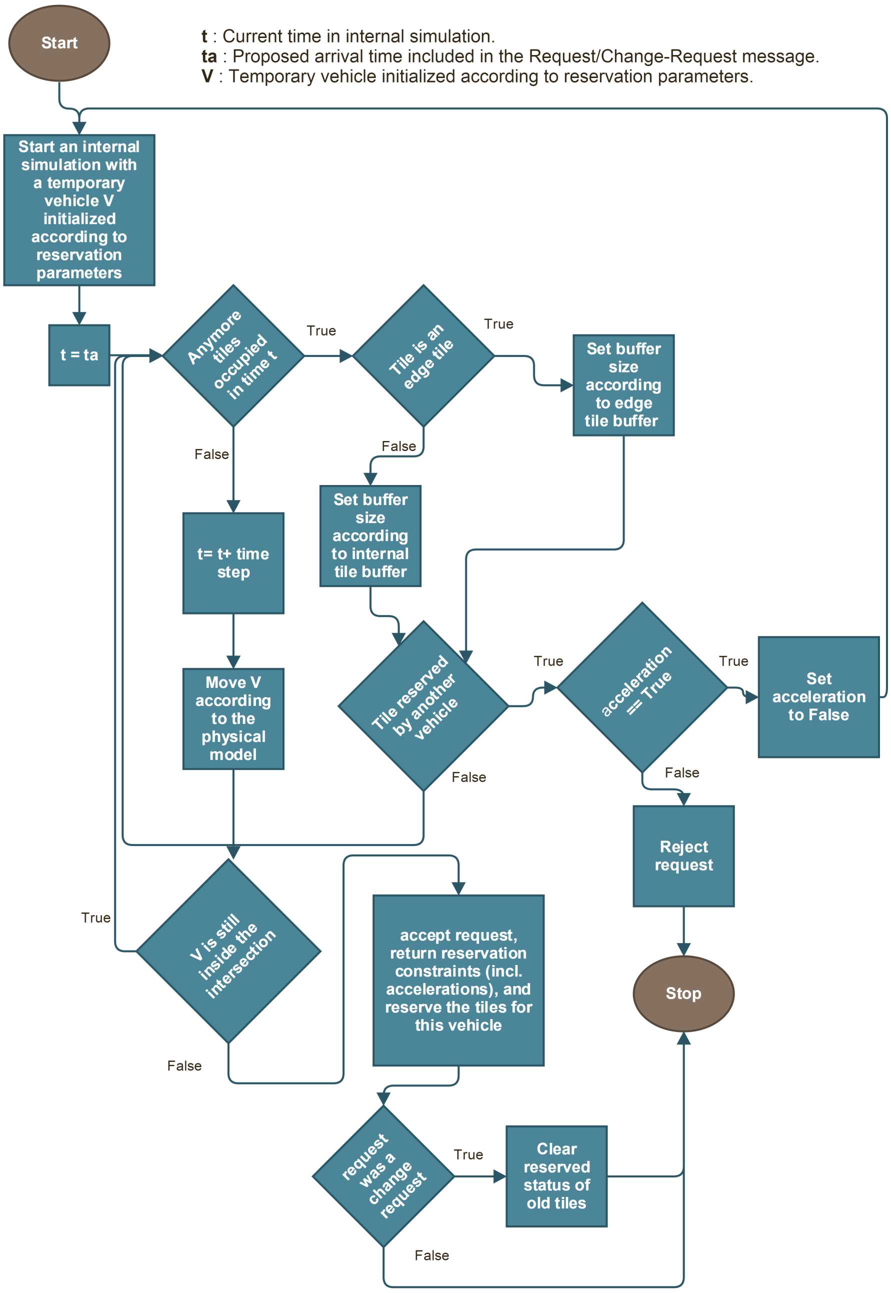

Start
Receive message from vehicle V
Message is Request
True
False
Process request for new reservation with policy
Message is Change Request
True
False
Process request for change of reservation with policy
Policy accepts request
True
False
Send
Confirm message to V containing reservation returned by policy
Send
Reject message to V
Message is Cancel or Done
True
False
Process Cancel or Done message with policy
Send
Acknowledge message to V

Figure 5: Intersection Manager in AIM



t : Current time in internal simulation.
ta : Proposed arrival time included in the Request/Change-Request message.
V : Temporary vehicle initialized according to reservation parameters.

Start
Start an internal simulation with a temporary vehicle V initialized according to reservation parameters
t = ta
Anymore tiles occupied in time t
True
Tile is an edge tile
True
Set buffer size according to edge tile buffer
False
Set buffer size according to internal tile buffer
False
t= t+ time step
Move V according to the physical model
Tile reserved by another vehicle
True
acceleration == True
True
Set acceleration to False
False
False
Reject request
True
V is still inside the intersection
False
accept request, return reservation constraints (incl. accelerations), and reserve the tiles for this vehicle
Stop
request was a change request
True
Clear reserved status of old tiles
False

Figure 6: FCFS policy in AIM

3 Simulation Environments for Autonomous Intersection Management Protocols

In this section we will discuss the simulation environments that we have used to simulate the AIM protocol and AMP-IP, and the simulation results we have obtained.

3.1 AIM-Simulator

The authors of AIM protocol have developed an open-source custom time-based simulator written in Java, which they call the *AIM-Simulator* [1], that we have used to test the performance of the AIM protocol.

A 250 m * 250 m area is modeled by the simulator, having the intersection at the center of that area.
Following are ordered steps that take place during each time step of the simulator:

1. Spawn new vehicles probabilistically
2. Provide sensor input through vehicles sensors/actuators to all vehicles
3. Allow driver agents controlling the vehicles to act
4. Updates vehicles' positions based on a physical model
5. Remove vehicles after reaching the end of the simulation area

A Driver Agent is a computer software that controls and pilots an autonomous vehicle, taking the role of a human driver. In order for driver agents to take the wheel, they have access to vehicle's properties(e.g. VIN, size, and acceleration capabilities), and state variables(e.g. velocity, heading and acceleration). In addition they have access to a set of simulated external sensors.
One of those sensors is a simulated laser range finder, which determines close-by vehicles and provides the distance and angle to the point on the nearby vehicle closest to the sensing vehicle. This provides the driver agent with the information needed to control the vehicle so that it does not hit vehicles in front.
Vehicles positions are updated each time step based on a physical model. This model assumes that vehicles do not slide across the road. It also assumes that vehicles move based on a set of differential equations for non-holonomic motion, as shown below:

$$\frac{\partial x}{\partial t} = v * cos(\phi) \tag{2}$$

$$\frac{\partial y}{\partial t} = v * sin(\phi) \tag{3}$$

$$\frac{\partial \phi}{\partial t} = v * \frac{tan(\psi)}{L} \tag{4}$$

x: vehicle's position in the x direction
y: vehicle's position in the y direction
ϕ: vehicle's orientation
v: vehicle's velocity
ψ: vehicle's steering angle
L: Distance between the vehicle's front and rear wheels.

In what follows we list the main packages in the AIM-Simulator source-code, describing the functionality of the main classes and methods.

3.1.1 AIM-Simulator Main Packages

For a more detailed explanation of the classes of the AIM-Simulator please refer to the simulator's API documentation [25].
1. **Driver**: Implements all the driver agent functionality that we mentioned earlier in section 2.

2. **GUI**: Implements the Graphical User Interface (GUI) of the simulator.
It includes the main visual area of the simulator drawing fixed and moving elements on the screen, such as roads, intersection and vehicles. This visual area is called the 'Canvas'. In addition the GUI package includes a 'Viewer' class allowing real time user interaction with the simulator. Finally the package includes two panels , one for setup and the other for showing statistics and status of the simulator.

3. **IntersectionManager**(im): Main classes of this package include:
V2I manager: which manages requests sent by vehicles to Intersection managers and coordinates their movement in the intersection making sure there are no collisions. It uses an intersection control policy for its decisions.
Policy: Implements the AIM control policy that we mentioned earlier in the previous section.
Reservation: which accepts/rejects reservation requests made by vehicles after running an internal simulation to find out if the requesting vehicle will occupy any tile that is already on the trajectory tile list of another vehicle.
Intersections: Deals with properties of the intersection, such as intersection area, roads and lanes controlled by this intersection.

4. **MAP**: This package Implements the map of the simulator that is used by the simulator's GUI.

5. **Messages**(msg): The msg package creates different types of messages specified by the AIM protocol; these include messages sent from vehicles to Intersection Managers and messages sent from Intersection Managers to vehicles.

6. **Simulator**: The simulator package is the main loop of the simulator. It calls a sequence of functions, in order, during each time step of the simulator. These functions (in order) are:
-Spawn new vehicles probabilistically
-Provide sensor input through vehicles sensors/actuators to all vehicles
-Allow driver agents controlling the vehicles to act
-Updates vehicles' positions based on a physical model
-Remove vehicles after reaching the end of the simulation area

7. **Vehicle**: This package Implements the vehicle model in the simulator. The vehicle model describes the vehicle's current movement (velocity, acceleration, heading, steering angle) and the vehicles specifications(maximum acceleration, length, width, maximum steering angle,etc.).

Next we describe our work in developing a simulation environment for the AMP-IP based on the AIM-Simulator.

3.2 Simulation Environment for AMP-IP based on AIM-Simulator

Instead of developing a simulator from scratch to simulate the AMP-IP, we thought of using an existing traffic simulator. Several commercial powerful traffic simulators exist like PARAMICS [17], AIMSUN [18],VISSIM [19] and CORSIM [20] but their price, copyright nature, rigidity in modifying their source code make them a less favorable option [16]. Other open-source alternatives exist as well, such as MOVE [21], Trans [22], MobiSim [23] and NCTUns [24] but their setup is difficult and generally they are hard to use [16]. Another open-source traffic simulator designed specifically to be modular and easy to use is ISR Traffic Simulator (ISR-TFS) [16], but it lacks good API documentation making it hard to understand and modify for our purpose.
Finally, the open-source simulator we have used, which overcomes the disadvantages of all previous simulators we mentioned; is the AIM-Simulator [1], discussed at the beginning of this section.

Although AIM-Simulator is modular, it was designed for a centralized protocol (AIM protocol), so we could not use it directly to simulate a decentralized protocol like the AMP-IP. So we have used the AIM-Simulator to develop a simulation environment in which AMP-IP could be simulated. The source code for the simulation environment that we developed is found at this link: `https://github.com/aymannedaa/AMP-IP_Simulator`

Following is a list of new packages that we have developed for the new simulator environment in order to simulate the AMP-IP, and the modifications we did for some AIM-Simulator classes and methods.

3.3 Additions and Modifications

msg.v2v: Based on AMP-IP specification we have developed a message class that creates the three types of messages sent between vehicles , which are Enter, Cross and Exit messages.
Trajectorycells: As each vehicle has to send an ordered list of cell numbers which it will occupy along the intersection, in addition to its estimated arrival time to each of those cells, each vehicle runs an internal simulation simulating its trajectory along the intersection using data from its current movement(velocity,acceleration,heading,etc,.) and data collected from received messages. The lists are Trajectory Cells List (TCL) and Trajectory Cells Arrival Time List (TCATL). These lists will be sent to nearby

vehicles in the same time step of the simulation. Note that TCL and TCATL will be updated at each time step of the simulation.
Policy: This class implements the AMP-IP policy discussed earlier in section 2.

3.3.1 Changes to AIM-Simulator packages

1. **Simulator**: we added a controller method to control the vehicle's movement instead of relying on the simulator's own driver agent package. Unlike the driver agent which controls the vehicle's communication and movement aspects, the controller only controls the movement of the vehicle and the communication aspect is added instead to the Vehicle package. We did not remove the Driver package from the code though as it includes several methods which are still of use by current packages.

2. **Vehicle**: In addition to adding communication capabilities to the Vehicle package by adding a V2VInbox and V2VOutbox to each vehicle, we added a state counter that will identify which state the vehicle is in (Intersection-approach, Intersection-Enter, or Intersection-Exit) depending on its position with respect to the Intersection. Finally, we added several other variables to the 'BasicVehicle' class part of the Vehicle's package to make the Vehicle a separate entity independent from the simulator's driver agent.

3. **IntersectionManager(im)**: As AMP-IP does not have an Intersection Manager we had to modify this package to include only the aspects concerning the Intersection properties (area,roads,lanes, etc,...) and remove the manager part which deals with aspects of the AIM protocol such AIM policy and reservations. That is why we have renamed the package, changing it from 'IntersectionManager' to 'Intersection' and changed corresponding classes names to suit their new functions.

4. **message**(msg): msg.v2i and msg.i2v are removed and the *msg* package now includes 'msg.v2v'

3.4 Simulation Results

We have run several simulation tests for a 4-way cross-intersection with 4-lane roads, with 2 lanes in each direction; for different traffic rates. Each simulation took 18 minutes of simulation time. We used the same configuration for running tests for the AIM protocol and for a traffic light model with two different green time intervals. Since the AIM-Simulator has a traffic light model, the AIM protocol and the traffic light model were simulated using the AIM-Simulator. The traffic light model follows the 'ALL-Lanes' model described in [1] which resembles some current traffic light systems. It uses constant green, yellow and red light intervals, with all lanes in one direction getting green lights, while the other 3 directions getting red lights, and after one cycle(green interval, then yellow change interval and finally red clearance interval), the second direction gets green lights for all of its lanes, while the other 3 directions get red lights. This continues in succession for the rest of the directions, and then repeats. We ran simulations for the traffic light model, once using 30 seconds of green interval, and a second time using 10 seconds of green interval, while in both cases we used 3 seconds of yellow change interval and 1 second of red clearance interval. The yellow and red interval values are based on the Traffic Engineering Manual, published by the Department of Transportation in the State of Florida [32].

Other traffic light systems have been developed in recent years which change the duration of green interval based on an actual level of current traffic on each direction, measured by sensors embedded in the roads; or anticipated level of traffic on each direction, based on statistics of the change in level of traffic over a day. Even though some of these traffic light systems and others might result in better performance than the models we have used, many traffic light systems around the world in use today still use static intervals. Therefore, it is valid, from our point of view, to compare AIM and AMP-IP to the traditional traffic light model.

In all the simulations, vehicles have equal probability of being spawned in each lane on each spawn road, with a turn ratio of 1:10, meaning that 10 percent of the vehicles turn right or left(with equal ratios) while the rest go straight. In AIM the granularity for the 20 m * 20 m intersection is set to 20, so each side of a reservation tile has a length of 1 m. As for AMP-IP the number of cells in the intersection grid is 16, allowing a vehicle to fit completely inside a cell.

Moreover, in all of our tests we have assumed an ideal communication channel, where no messages are lost during transmission. Simulating message loss is out of the scope of this thesis.

From the tests we ran, we have plotted the Average Delay against the Traffic Level, shown in figure 7. The Average Delay is the summation of the delay faced by each vehicle due to waiting at the intersection divided by the total number of vehicles which have been cleared by the simulator due to reaching the end of their journey. The delay faced by each vehicle is calculated by subtracting from the actual journey time of the vehicle the ideal journey time where the vehicle stays with a constant speed (which equals the speed limit of the road) throughout the journey without stopping or slowing down. The plot shows that AMP-IP decreases the delay faced by vehicles at the intersection compared to the traffic light model, but not as much as the AIM protocol.

3.4.1 Performance Analysis

AIM and AMP-IP reduce the delay faced by vehicles in an intersection compared to the traffic light thanks to allowing more parallelism inside the intersection. Specifically, a vehicle that has no potential conflict with crossing vehicles is able to cross the intersection in parallel with other vehicles already crossing the intersection.

In AMP-IP a vehicle which has a trajectory conflict with a higher priority vehicle has to come to a complete stop before the Trajectory Conflicting Cell (TIC). After the conflicting vehicle exits the TIC, the vehicle can cross the intersection, assuming it has no more trajectory conflicts with other higher-priority vehicles. Since the vehicle comes to a complete stop before accelerating again, this increases the vehicle's

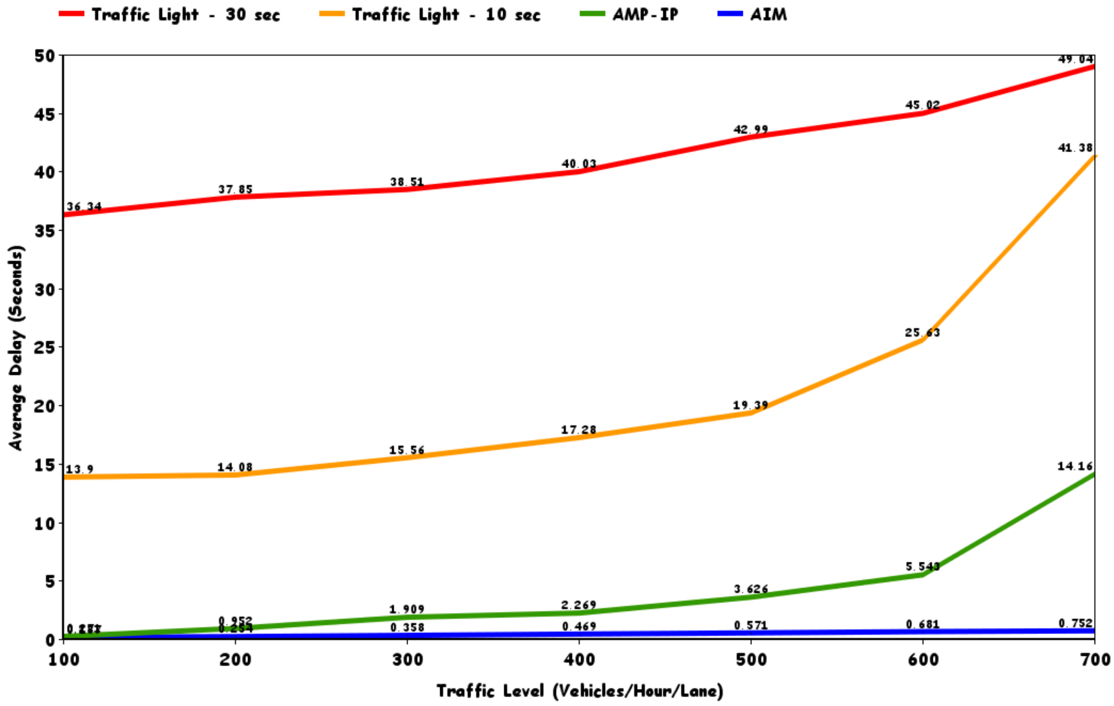


Figure 7: Average delay for different intersection management policies

delay due to waiting at the intersection. In AIM, on the other hand, a conflicting vehicle that receives a reject message from the intersection manager will slow down providing the crossing conflicting vehicle which received a confirmation with enough time to cross the conflicting space minimizing a vehicle's need to get to a complete stop. In addition to allowing vehicles to slow down, in AIM even when a vehicle gets to a complete stop at the intersection due to a trajectory conflict with a higher priority vehicle, this vehicle will start to accelerate even before the conflicting higher priority vehicle leaves the occupied tile(s), reaching the occupied tile(s) right after the higher priority vehicle has exited the tile(s). However, in AMP-IP a lower priority vehicle will only start to accelerate after the higher priority vehicle has exited the TIC, which introduces additional delay compared to AIM. We believe the two reasons mentioned in this paragraph explain why the average delay plot in figure 7 shows AIM having better performance in terms of delay than AMP-IP.

The sharp increase in Average Delay between 600 vehicles per hour per lane (veh/h/ln) and 700 veh/h/ln in the AMP-IP curve in figure 7, is due to the fact that before and including 600 veh/h/ln, the traffic rate is low enough that when vehicles stop before or inside the intersection they usually are able to cross the intersection before vehicles at the back reach the intersection. So most of the times throughout the simulation before and including 600 veh/h/ln, there are no queues at the intersection. However, somewhere between 600 veh/h/ln and 700 veh/h/ln the traffic level is high enough that vehicles reach the intersection before the vehicles in front of them have crossed the intersection. So at that rate small queues start to form at the intersection. These queues get longer with the increase of traffic level.

As for the traffic light with 10 seconds green interval curve, the sharp increase in Average Delay between 600 veh/h/ln and 700 veh/h/ln is due to the fact that queues at the intersection become long enough that a vehicle waiting in a queue might not be able to cross the intersection when the traffic light for its approach becomes green. Such a vehicle will have to wait at least until the next green cycle before it can cross the intersection.

The maximum traffic level in the Average Delay plot is 700 veh/h/ln. The reason for not showing the Average Delay for higher vehicle rates, is that for both traffic light models, traffic level higher than 700 veh/h/ln leads to queues building up for certain lanes that vehicles reach the start of these lanes. At that point, the AIM-Simulator stops spawning vehicles on these overcrowded lanes as there is no more room for any more vehicles on these lanes. For the AMP-IP queues build up to the start of some lanes for traffic levels higher than 800 veh/h/ln. As for AIM, this happens at very high traffic rates, higher than 2000 veh/h/ln.

3.4.2 Checking Safety by Simulation

In addition to the simulation tests shown in the Average Delay plot, we have run longer tests (around one hour of simulation time) just for the purpose of checking for collisions between vehicles in case of AMP-IP and AIM. In all of our simulations for both protocols we have observed no collisions between vehicles.

4 Statistical Model Checking

In this section we will give a brief overview of statistical model checking, and why we have used it. Then we will give a brief overview of the UPPAAL model checker which is the tool we have used to model and check (by statistical model checking) whether AIM results in any collisions between vehicles crossing an intersection or not with a certain degree of confidence.

In our simulation tests in the previous section, we observed no collisions between vehicles crossing an intersection using AIM protocol and AMP-IP. But we wanted to take this further and prove this through formal verification methods, which allow us to be more confident about whether these protocols cause collisions or not compared to simulations. Formal verification methods are different from simulation methods in that they investigate the whole state space of the system instead of a small area of the space, allowing us to prove or disprove the correctness of a system in satisfying its requirements [26].

However, due to the complexity of the protocols we are modeling, using formal verification methods would probably lead to the well-known problem called State Explosion Problem [27]. This problem is due to the fact that the number of system states grows exponentially in the size of the system (number of state variables, protocol rules, etc.). Because of this exponential blow-up, exhaustively exploring all system states is often infeasible in practice. So we turned to another verification technique called *statistical model checking* [29] which is a compromise between simulation and formal verification, avoiding exhaustive exploration of the state-space of the model.

4.1 Statistical Model Checking

In statistical model checking, a finite number of simulation runs of the system are executed, then hypothesis testing is used to decide, based on the simulation results, if the system satisfies or violates a requirement with some level of confidence [28].

In the following lines, we will define two terms that we will use in the rest of this section.

Timed Automata

Timed Automata is a model proposed by Rajeev Alur and David L. Dill in [36] to model the timing behavior of real-time systems. In this model transitions are constrained by time using finite set of clocks having real values [36]. In the start of the automata run, clocks are initialized with zero, then throughout the run all clocks in the automata increase their value with the same rate. Clocks can be reset to zero following a transition.

Figure 9 shows an example of a timed automaton, where the automaton does not perform a transition until the clock (x) reaches a value equal to *spawn_time*. After this, a function is called(spawn Vehicle) then the clock is reset to zero, and the cycle repeats.

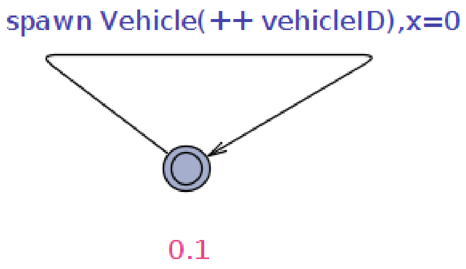


Figure 8: Stochastic Timed Automata example

Stochastic Timed Automata

Stochastic Timed Automata are more general forms of timed automata in which probability distributions for time transitions could be used [37].

Figure 8 shows how SMC is used to define the vehicles' spawn rate. In the figure, vehicles are spawned according to a rate of exponential probability distribution, where the probability of transition is distributed according to the exponential distribution: $1 - e^{-\lambda t}$, where t is the time and λ is the fixed rate, with λ being 0.1 in the example shown in figure 8.

4.2 UPPAAL Model Checker

Developed in collaboration between Uppsala University in Sweden and Aalborg University in Denmark, the UPPAAL model checker is a tool environment used for modeling, simulation and verification of real-time systems. A system in UPPAAL is represented as networks of timed automata (several timed automata working in parallel), where the system behavior depends on clock variables and data variables, that are either local or shared between automata [30]. It includes useful data types like integer and double data types, useful C-like structures and offers channel synchronization. An automaton in UPPAAL is a graph, consisting of locations (graphically represented as circles) connected by edges [38]. These edges could optionally have guards which confine the behaviour of the automaton. Locations in UPPAAL could be normal, urgent, or committed locations. Time may not progress in either an urgent location or a committed location. A committed location is more restrictive than an urgent location, in that if any process is in a committed location, the next transition from this location has to include an edge from one of the committed locations [39]. Channels in UPPAAL are used for synchronization between senders and receivers with a label C! denoting a send operation, and C? denoting a receive operation on a channel C.

UPPAAL uses a simplified version of Timed Computation Tree Logic (TCTL) [40] to express model checking queries. The query language in UPPAAL consists of two formulas, state formulas and path formulas [39].

State formulas test whether one or more properties are satisfied for specific states or not, while path formulas test whether one or more properties are satisfied for specific paths or not. For path formulas we can test for reachability, safety and liveness [39]. Reachability is the ability to get from one location to another. This is only satisfied if there exists a path between the two locations. For example, a vehicle at the start of its spawn road has to reach the intersection at some point in its trip. Safety means that "something bad will never happen" in the execution of a program. For example, vehicles crossing an intersection will not collide. Liveness means that "something good" will happen eventually. For example, a vehicle eventually exits the intersection.

UPPAAL academic version 4.1 has an 'SMC extension' which allows checking whether properties of the system are satisfied or not with certain degree of confidence, using statistical model checking. [31].

In Uppaal-SMC, networks of timed automata are extended to networks of stochastic timed automata, where non-deterministic choices of time delays could be added to the timed automata through probability distributions as been shown earlier in figure 8.

The query language is extended in Uppaal-SMC to express queries related to the stochastic interpretation of timed automata [31].

4.2.1 Editor and Verifier

The GUI in UPPAAL includes an *Editor*. The Editor allows the user to edit different automata in the system, including declarations of variables(local and global), and functions used by these automata. The GUI also includes a *Verifier*, which is used to run model-checking queries. In addition to the verifier in the GUI, UPPAAL offers a stand-alone command line verifier called 'verifyta', which is helpful when running verification tasks remotely [39].

5 Modeling Traffic Light in UPPAAL

After discussing UPPAAL SMC in the previous section, we move forward to describe our first model in UPPAAL SMC, explaining all the automata in the model. We have started by modeling a simple intersection management policy, namely the *traffic light*. Then we have used UPPAAL SMC verifier tool to calculate the probability of having collisions on the intersection and on the road.

Following are bullets containing all automata in the traffic light model. Under each automaton is a list of (main) functions used by the automaton:

- **Spawning Traffic Automaton**
  - *Spawn Vehicle*: Create vehicles dynamically and spawn them on the start of their spawn roads.
- **Vehicle Automaton**
  - *chooseSpawnLane*: Choose the spawn lane based on the direction the vehicle wants to take, and set the destination road and lane.
  - *setVehInitialState*: set vehicle's initial state.
  - *move*: Move the vehicle according to *Newtonian kinematics for uniform acceleration*.
  - *runPolicy*: Run the traffic light policy.
  - *endOfJourney*: Check if the vehicle has reached the end of its trip.
  - *terminatevehicle*: Reset the vehicle's global variables before termination.
  - *exit*: Terminate the dynamically created vehicle.
- **Collision Detection Automaton**
  - *checkCollisions*: Count the number of collisions on the intersection and on the road.
- **Traffic Light Controller Automaton**

UPPAAL XML file of the traffic light model can be downloaded from this link: `https://github.com/aymannedaa/UPPAAL`

5.1 Spawning Traffic Automaton

The Spawning Traffic Automaton is shown in figure 9 . This automaton is used to spawn vehicles. When the clock (x) reaches a predefined value *spawn_time*, a vehicle is spawned dynamically by calling UPPAAL's internal *spawn* function followed by the automaton we want to spawn (Vehicle automaton), and passing an argument of vehicle ID after incrementing it by one. This means that vehicles spawned are allocated vehicle IDs in an ascending order. After the *Spawn* function is called, the clock resets. This cycle is repeated throughout the run.

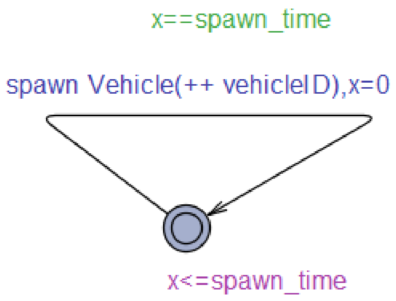


Figure 9: Spawning Traffic Automaton

5.2 Vehicle Automaton

The Vehicle Automaton is shown in figure 10. Each Vehicle automaton created dynamically runs in parallel with existing Vehicle automata. The automaton starts from the initial state, shown in figure 10 on the top by a circle inside the location. A vehicle has equal probability to be spawned at any of the 4 approaches of the intersection (North, South, East and West). This is done by assigning equal weights of '1' over the edges reaching each of the 4 locations. And it has a double chance of turning than going straight. This is done by assigning a weight of '2' over the edge reaching the *Turn* location compared to a weight of '1' on the edge reaching the *Straight* location as shown in figure 10. If the vehicle turns then the vehicle will have an equal chance to turn right or left, indicated on the figure by equal weights of '1' over the edges reaching the *right* and the *left* locations. The direction of turn influences the vehicle's spawn lane. For example, in case of three lanes per road in each direction, a vehicle turning right will be spawned in the far right lane of the road, while a vehicle turning left will be spawned in the far left lane of the road. Afterwards we set the initial state of the vehicle by setting parameters such as the vehicle's velocity, acceleration, position and orientation.

All states until and including the *Initialize* state are 'committed locations' that do not affect time. But when we reach the *Move* state, the automaton stays in this state for one time unit before calling the 'move' function, which is responsible for moving the vehicle forward by updating the vehicle's position, followed by a call to 'runPolicy' function which allows the vehicle to take the correct action according to the traffic light condition. Each automaton has its own internal clock that is independent of other clocks(all the automata use the same variable name 'x' for the clock). Accordingly, 'x' (shown in figure 10) is an internal clock for the Vehicle automaton. After reaching the *Move* state, clock x resets every 1 time unit. This will loop until the vehicle reaches the end of its journey(exits its destination road). When the vehicle reaches the end of its journey UPPAAL's internal 'exit' function is called to terminate the vehicle.

Vehicles move according to *Newtonian kinematics for uniform acceleration*, and the *bicycle model* [35] is used to model vehicle steering.

Vehicles keep a safety distance between them and the vehicles in front, and decelerate accordingly to keep this safety distance.

5.3 Collision Detection Automaton

The Collision Detection Automaton is shown in figure 11 . This is a simple automaton that is used to calculate at each time unit whether collisions happen between vehicles on the intersection or on the road or not by calling the 'checkCollisions' function. With every collision a counter for the number of collisions *no_ of_ collisions* is incremented.

The 'checkCollisions' function checks for collisions through testing if any of the first vehicle's edge(corner) points are found to be inside the polygon of the second vehicle. If that was false, another test is done, this time by checking if any of the

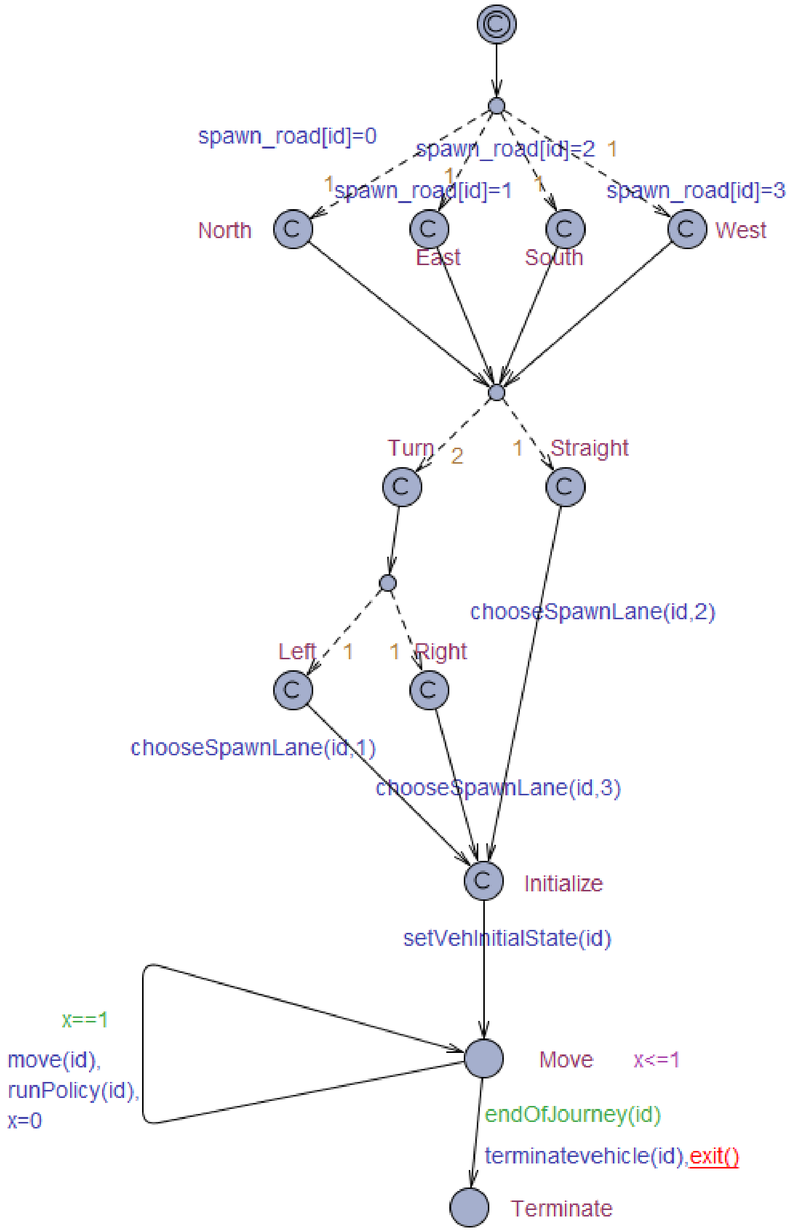


Figure 10: Vehicle Automaton

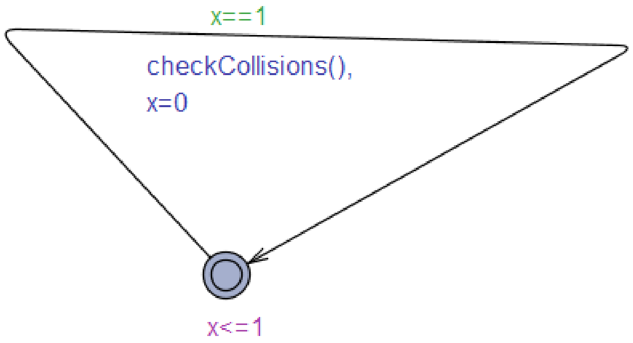


Figure 11: Collision Detection Automaton

second vehicle's edge points are found to be inside the polygon of the first vehicle.

In order to test if a test point is found inside a polygon defined by a set of points, we use an implementation of the *Jordan Curve Theorem* [33], which counts the number of edges of a polygon that will be crossed by a ray coming out of a test point. If the count was an odd number then the point intersects the polygon, otherwise it does not.

5.4 Traffic Light Controller Automaton

The Traffic Light Controller Automaton is shown in figure 12 . The automaton starts by the *North_ Green* state, which sets the traffic light signal for the North approach to green and sets all the traffic light signals for all the other approaches to red. Note that when the automaton is in a certain state, all the approaches except for the approach that is named in the state (e.g. the North approach in *North_ Green*, *North_ Yellow* and *North_ Red* states) are assumed to have a red traffic light signal. After a predefined 'green_time' the traffic light signal for the north becomes yellow. Then after a predefined 'yellow_time' the traffic light signal for the north becomes red. The automaton stays in this *North_ Red* state for a predefined 'red_time' then it moves to the next state (*East_ Green*) where the traffic light for the East approach becomes green while all other approaches are red. This continues for the rest of the states until the final state. Then the automaton starts over from the first state and the loop goes on.

In the traffic light model, there is no explicit communication taking place between the Traffic Light Controller and the Vehicle Automaton. A vehicle is assumed to implicitly know the traffic light state(*green*, *yellow* or *red*) for the road it is spawned on. In reality, autonomous vehicles could use camera-based detection of traffic lights to figure out the state of the traffic light [34]. When traffic light state is *yellow*, vehicles that have a distance from the intersection that allows them to stop before entering the intersection will stop, while others will cross, similar to how humans react to a traffic light in a *yellow* state. That is why the yellow change interval should be long enough to insure that all vehicles, which are not able to stop when the traffic light changes from green to yellow, will cross.

5.5 Results

For a *spawn rate* of 1.786 vehicles per second, which translates to a traffic volume of about 535 veh/h/ln, with 6 lane roads, 3 lanes in each direction, we ran a query in UPPAAL's SMC verifier to check if our traffic light model results in any collisions between vehicles or not.

The query (shown below) computes the probability of the *no_ of_ collisions* counter being zero for all 3000 time units (one time unit is 20 milliseconds) of running time.

Query: `Pr ([][0,3000] no_of_collisions == 0)`

The results of this query(shown below) is given as an interval that shows an estimate plus or minus a *precision* value (here the precision is set to 0.05).

Results: `[0.95,1]`

The interval is believed with some *level of confidence* (confidence level is set to 95%) to contain the *true probability* of having no collisions over a period of 3000 time units. A true probability is the actual probability that an event will happen in a given situation [41] which is not computable in practice as it needs infinite number of trials. Our estimates can only come close to the true probability when we increase the number of runs.

The results are the highest interval values for such a precision, so it is deemed to be satisfactory.

Depending on the precision and the confidence level, which are 2 user defined parameters, UPPAAL SMC calculates the *number of runs* which are required for the query, and here it was 738 runs. The query took around 4 days to execute. Due to the long execution time of the query, we did not perform longer queries, specially that we wanted to allocate more time for modeling and verifying(through SMC) the AIM protocol in UPPAAL, which is of more interest to us than the traffic light.

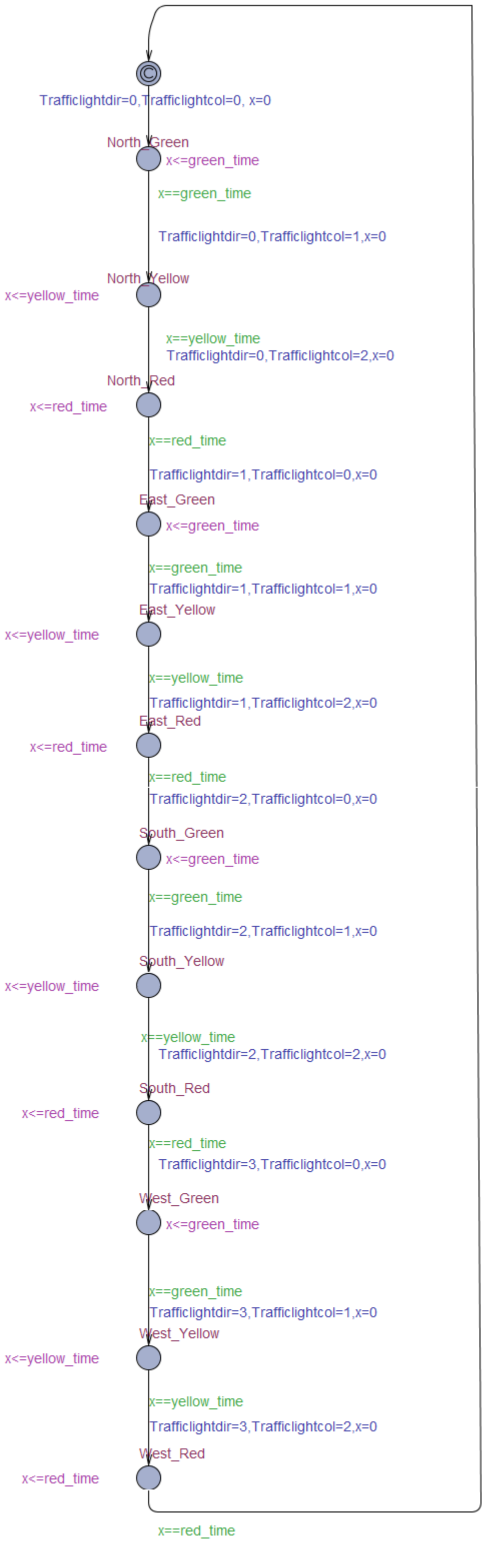

Trafficlightdir=0,Trafficlightcol=0, x=0
North_Green
x<=green_time
x==green_time
Trafficlightdir=0,Trafficlightcol=1,x=0
North_Yellow
x<=yellow_time
x==yellow_time
Trafficlightdir=0,Trafficlightcol=2,x=0
North_Red
x<=red_time
x==red_time
Trafficlightdir=1,Trafficlightcol=0,x=0
East_Green
x<=green_time
x==green_time
Trafficlightdir=1,Trafficlightcol=1,x=0
East_Yellow
x<=yellow_time
x==yellow_time
Trafficlightdir=1,Trafficlightcol=2,x=0
East_Red
x<=red_time
x==red_time
Trafficlightdir=2,Trafficlightcol=0,x=0
South_Green
x<=green_time
x==green_time
Trafficlightdir=2,Trafficlightcol=1,x=0
South_Yellow
x<=yellow_time
x==yellow_time
Trafficlightdir=2,Trafficlightcol=2,x=0
South_Red
x<=red_time
x==red_time
Trafficlightdir=3,Trafficlightcol=0,x=0
West_Green
x<=green_time
x==green_time
Trafficlightdir=3,Trafficlightcol=1,x=0
West_Yellow
x<=yellow_time
x==yellow_time
Trafficlightdir=3,Trafficlightcol=2,x=0
West_Red
x<=red_time
x==red_time

Figure 12: Traffic Light Controller Automaton

6 Modeling AIM in UPPAAL

In the previous section we described the traffic light model in UPPAAL SMC , now we turn to our model of the AIM protocol in UPPAAL SMC.

Following is a list containing all automata in the AIM model with their respective (main) functions:

- **Spawning Traffic Automaton**
    - *Spawn Vehicle*: Create vehicles dynamically and spawn them on the start of their spawn roads.
- **Vehicle Automaton**
    - *chooseSpawnLane*: Choose the spawn lane based on the direction the vehicle wants to take, and set the destination road and lane.
    - *setVehInitialState*: set vehicle's initial state.
    - *move*: Move the vehicle according to *Newtonian kinematics for uniform acceleration.*
    - *IntersectionApproach*: Checks if the vehicle is approaching the intersection.
    - *driverAgent_sending*: Implements the driver agent sending side by creating the appropriate messages to be sent to the Intersection Manager according to the vehicle's state.
    - *driverAgent_receiving*: Implements the driver agent receiving side by looking at the content of the received messages (received from the Intersection Manager) and allowing the vehicle to take the appropriate action accordingly.
    - *endOfJourney*: Check if the vehicle has reached the end of its trip.
    - *terminatevehicle*: Reset the vehicle's global variables before termination.
    - *exit*: Terminate the dynamically created vehicle.
- **Collision Detection Automaton**
    - *checkCollisions*: Count the number of collisions on the intersection and on the road.
- **AIM Intersection Manager Automaton**
    - *AIMpolicy_IM*: Look at the content of a message received from a vehicle, implement the AIM policy by running an internal simulation of the requesting vehicle's trajectory through the intersection using the vehicle's reservation request parameters and finally create a message which will be sent back to the requesting vehicle.

UPPAAL XML file of the AIM model can be downloaded from this link: `https://github.com/aymannedaa/UPPAAL`

The Spawning Traffic automaton and the Collision Detection automaton are also included in the AIM model and are identical to the ones described in the traffic light model, in the previous section. The Vehicle Automaton has undergone some changes for communication purposes, and the Traffic Light Controller is replaced with an Intersection Manager automaton.

6.1 Vehicle Automaton

The Vehicle Automaton for AIM model is shown in figure 13. The Vehicle Automaton in AIM model is similar to the one in the traffic light model except for the addition of some states for communication between vehicles and the Intersection manager. Before the *IntersectionApproach* state, the model is identical to the one in the traffic light except for the removal of the 'runPolicy' function which was related to the traffic light policy.

Vehicles reach the intersection approach state when their distance to the intersection is less or equal to a predefined value. At that point the vehicle checks if the communication channel is being used by another vehicle or not, and if the channel was busy, the vehicle waits until the channel becomes available (states are 'committed locations', so even though vehicles 'wait' for each other to use the channel the clock does not proceed).

When the channel becomes available the vehicle sends a message through the V2I channel to the intersection manager through calling the 'driverAgent_sending' function and then receives a message from the intersection manager through the I2V channel and processes it by calling the 'driverAgent_receiving' function.

In our model we have only two channels, one is being used by vehicles to send messages to the intersection manager (V2I) and the other to receive messages from the intersection manager(I2V). One vehicle can only be using the channel at a given time. Channels in UPPAAL are only used to synchronize, meaning that there is no data being sent on the channel. Global shared variables are set from the sender and checked by the receiver to model data exchange. After receiving messages from the intersection manager, vehicles update their position based on the messages received, by calling the 'move' function after one time unit. Afterwards, the cycle for checking if the channel is busy or not, then sending and receiving messages and finally updating the position is repeated until the vehicle reaches the end of its journey where it is terminated.

As in the traffic light model, vehicles keep a safety distance between them and the vehicles in front, and decelerate accordingly to keep this safety distance. However, in the AIM model, this is limited to vehicles on the road(i.e. vehicles not crossing the intersection), as the AIM policy is responsible for planning how vehicles cross the intersection. So while crossing the intersection, vehicles only follow the AIM policy.

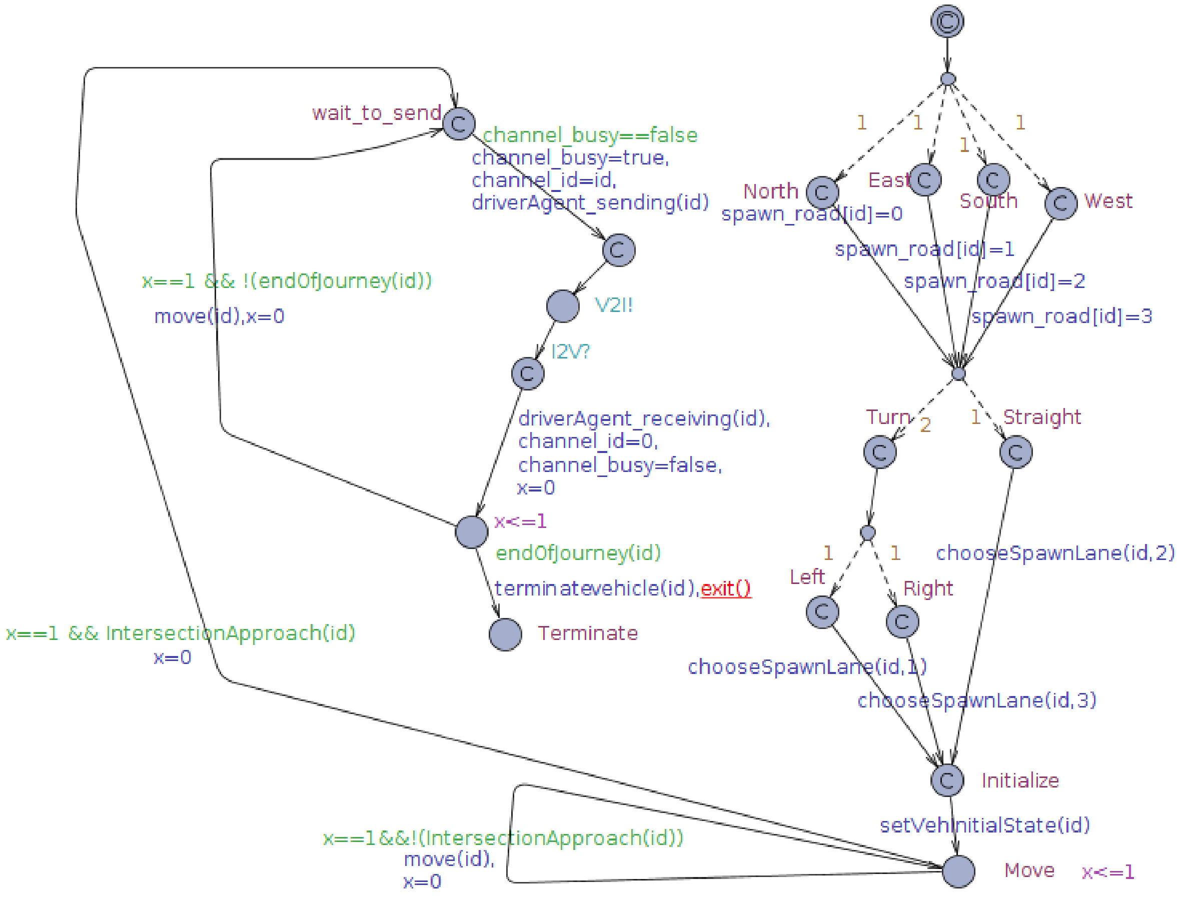


Figure 13: Vehicle Automaton

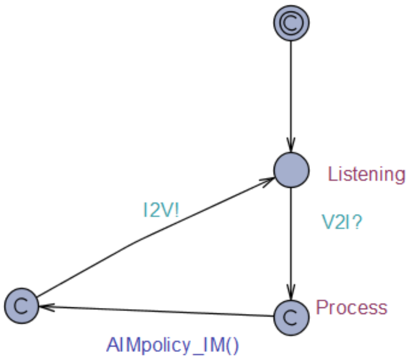


Figure 14: Intersection Manager Automaton

6.2 Intersection Manager Automaton

The Intersection Manager Automaton is shown in figure 14. The intersection manager constantly listens to messages from vehicles. When it receives a message it runs the AIM policy to process the message and based on the policy's decision a reply in the form of a message is sent back to the vehicle.

The AIM policy runs an internal simulation of the requesting vehicle's trajectory along the intersection based on the parameters provided in the request message. The internal simulation finds the tiles that will be occupied by the requesting vehicle at each time step of the simulation, in order to decide if the requesting vehicle should be granted a reservation or not. We check if a vehicle will occupy a tile or not through testing if any of the vehicle's edge points are found to be inside the polygon of the tile. If that was false, we do another test this time by testing if any of the tile edge points are found to be inside the polygon of the vehicle or not. If that was also false, then the tile is not occupied by the vehicle at this time step, otherwise it is.

The *point-in-polygon* test is done using an implementation of the *Jordan Curve Theorem* [33], which was described in the previous section.

6.3 Results

Due to the complexity of the AIM model over the traffic light model, when we tried to run the same query as we did in the traffic light model with the same configuration parameters, UPPAAL SMC results in a *memory exhaustion* error. This was mostly due to the number of vehicles being spawned (a total of 107 vehicles spawned in the case of traffic light query). We first thought of decreasing the spawn rate; however, having lower spawn rate would lead to less conflicting trajectories which in turn will lead to biased results, as many vehicles would not collide simply as they do not have conflicting trajectories along the intersection and not due to following the AIM protocol.

In order to maintain the same spawn rate(1.785 vehicles per second) but lower the number of vehicles per run, we lowered the number of time steps to 1000 instead of 3000. In order to have more conflicting trajectories in this duration we reduced the number of lanes per road to 2, having 1 lane in each direction, which gives a traffic volume of 1607 veh/hr/lane. In addition each lane is made shorter, so that more vehicles reach the intersection in the duration of the run.

The buffer values chosen for our model are similar to the values used in the AIM-Simulator.

The query (shown below) computes the probability of the *no_ of_ collisions* counter being zero for all 1000 time units of running time.

Query: `Pr ([][0,1000] no_of_collisions == 0)`

The results(shown below), as we described in the previous section, is given as an interval that shows an estimate plus or minus a precision value that is believed with some level of confidence to contain the true value of the population parameter.

Results: `[0.95,1]`

The results are the highest interval values for such a precision, so these results are considered satisfactory.

The precision, confidence level, and the number of runs is the same as in the traffic light query described in the previous section.

7 Conclusions and Perspectives

7.1 Conclusions

As autonomous vehicles will become a reality in the near future, our traditional traffic light intersections will become outdated and developing more efficient automated intersections will be possible. Several autonomous intersection management protocols have been proposed in the academia for this purpose. Among these, two protocols have been studied and evaluated in this thesis.

In this context, we have modeled the AMP-IP and developed a simulation environment for it, based on the previously developed AIM-simulator, in order to compare the performance of AMP-IP to that of the AIM protocol and of traffic lights with different green time intervals.

In our simulation tests for AIM protocol and AMP-IP we observed no collisions between vehicles crossing a 4-way cross intersection assuming an ideal communication channel, when no messages are lost between the communicating parties, and when vehicles follow the protocols precisely. In addition, our performance analysis results show that both protocols lead to a reduction in traffic delay compared to the traffic light model, with AIM having the lowest traffic delay.

After simulations, we turned to statistical model checking, to get more credible results on whether these protocols, studied earlier, are free from collisions or not, having the same assumptions mentioned in the previous paragraph, while avoiding exhaustive exploration of the state-space. We have used a well-known model checking tool for this purpose called UPPAAL. We modeled AIM in UPPAAL, and in the process we developed a traffic light model as well. The statistical model checking results for AIM have backed-up the simulation results, showing no collisions between vehicles while crossing a 4-way cross intersection.

7.2 Future Work

We consider our work as a starting point, and many challenges remain. A main challenge is to be able to run longer tests for the AIM model in UPPAAL. Once we overcome this challenge it will open up the door for modeling and verifying(through SMC) other recent intersection management protocols like the AMP-IP and others in UPPAAL. Other challenges that lie ahead are those challenges facing formal verification of complex systems. Overcoming those challenges would lead to formally proving or disproving safety properties in autonomous intersection management protocols, increasing the chances of deploying reliable automated intersections on our roads in the near future.

References

[1] Kurt Dresner and Peter Stone. 2008. A multiagent approach to autonomous intersection management. J. Artif. Int. Res. 31, 1 (March 2008), 591-656.

[2] Azimi, S., Bhatia, G., Rajkumar, R. and Mudalige, P. (2013). Reliable intersection protocols using vehicular networks. pp.1–10.

[3] World Health Organization. Global Status Report On Road Safety 2013: Supporting A Decade Of Action. WHO Press, 2013.

[4] Europa.eu, 'European Commission - PRESS RELEASES - Press release - Road Safety Programme 2011-2020: detailed measures', 2010. [Online]. Available: http://europa.eu/rapid/press-release_MEMO-10-343_en.htm. [Accessed: 12 April 2015].

[5] Seriousaccidents.com, (n.d.). Top 25 Causes of Car Accidents . [online] Available at: http://seriousaccidents.com/legal-advice/top-causes-of-car-accidents/ [Accessed 12 April 2015].

[6] Niccolai, 'Self-driving cars a reality for 'ordinary people' within 5 years, says Google's Sergey Brin', Computerworld, 2012. [Online]. Available: http://www.computerworld.com/article/2491635/vertical-it/self-driving-cars-a-reality-for–ordinary-people–within-5-years–says-google-s-sergey-b.html. [Accessed: 13 April 2015].

[7] Kurt Dresner and Peter Stone. Multiagent Traffic Management: A Reservation-Based Intersection Control Mechanism. In The Third International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 530–537, July 2004.

[8] Kurt Dresner and Peter Stone. Human-Usable and Emergency Vehicle-Aware Control Policies for Autonomous Intersection Management. In AAMAS 2006 Workshop on Agents in Traffic and Transportation, May 2006.

[9] Minjie Zhu; Xu Li; Hongyu Huang; Linghe Kong; Minglu Li; Min-You Wu, "LICP: A look-ahead intersection control policy with intelligent vehicles," Mobile Adhoc and Sensor Systems, 2009. MASS '09. IEEE 6th International Conference on , vol., no., pp.633,638, 12-15 Oct. 2009

[10] R.Azimi, G. Bhatia, R. Rajkumar, P. Mudalige “STIP: Spatio-Temporal Intersection Protocols for Autonomous Vehicles” ACM/IEEE 5th International Conference on Cyber-Physical Systems (ICCPS), 2014.

[11] R.Azimi, G. Bhatia, R. Rajkumar, P. Mudalige “Intersection Management using Vehicular Networks” Society for Automotive Engineers (SAE) World Congress,April 2012, Detroit, MI, USA.

[12] Mladenovic, M., Abbas, M., Priority-Based Intersection Control Framework for Self-Driving Vehicles: Agent-based Model Development and Evaluation, 3rd International Conference on Connected Vehicles, 2014

[13] Standards.ieee.org, (2014). IEEE SA - 802.11p-2010 - IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments. [online] Available at: http://standards.ieee.org/findstds/standard/802.11p-2010.html [Accessed 16 April 2015].

[14] Standards.sae.org, (n.d.). J2735: Dedicated Short Range Communications (DSRC) Message Set Dictionary - SAE International. [online] Available at: http://standards.sae.org/j2735_200911/ [Accessed 16 April 2015]

[15] Matthew Hausknecht, Tsz-Chiu Au, and Peter Stone. Autonomous Intersection Management: Multi-Intersection Optimization. In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), September 2011.

[16] Bento, L., Parafita, R. and Nunes, U. (2012). Intelligent traffic management at intersections supported by V2V and V2I communications. pp.1495—1502.

[17] Paramics-online.com, 'Quadstone Paramics | Traffic and Pedestrian Simulation, Analysis and Design Software', 2015. [Online]. Available: http://www.paramics-online.com/. [Accessed: 20 April 2015].

[18] AIMSUN Version 4.1 User's Manual, TSS-Transport Simulation Systems, 2002.

[19] Vissim.com, 'The smarter, faster way for model-based system development | VisSim', 2015. [Online]. Available: http://www.vissim.com/. [Accessed: 20 April 2015].

[20] Mctrans.ce.ufl.edu, 'TSIS-CORSIMTM', 2015. [Online]. Available: http://mctrans.ce.ufl.edu/mct/?page_id=63. [Accessed: 20- April- 2015].

[21] F. Karnadi, Z. Mo, K.-C. Lan, Rapid Generation of Realistic Mobility Models for VANET, in Proc. of the IEEE Wireless Communication and Networking Conference (WCNC07), March 2007.

[22] M. Piorkowski, M. Raya, A. L. Lugo, P. Papadimitratos, M. Grossglauser, and J.-P. Hubaux, "Trans: realistic joint traffic and network simulator for vanets" SIGMOBILE Mob. Comput. Commun. Rev., vol. 12, no. 1, pp. 3133, 2008.

[23] J. Härri, M. Fiore, F. Fethi, and C. Bonnet, VanetMobiSim: generating realistic mobility patterns for VANETs, in Proc. of the 3rd ACM International Workshop on Vehicular Ad Hoc Networks (VANET'06), September 29, 2006, Los Angeles, USA.

[24] C.L. Chou, Y.H. Chiu, Y.S. Tzeng, M.S. Hsu, Y.W. Cheng, W.L. Liu, T.W. Ho, NCTUns 4.0: An Integrated Simulation Platform for Vehicular Traffic, Communication, and Network Researches, Vehicular Technology Conference, Baltimore, Maryland USA, 2007.

[25] Cs.utexas.edu, (n.d.). AIM4 1.0-SNAPSHOT API. [online] Available at: `http://www.cs.utexas.edu/~aim/aim4sim/aim4-release-1.0.3/aim4-root/target/site/apidocs/index.html` [Accessed 20 April 2014].

[26] Georgoulas, S.; Moessner, K.; Mcaleer, B.; Tafazolli, R., "Using formal verification methods and tools for protocol profiling and performance assessment in mobile and wireless environments," Personal Indoor and Mobile Radio Communications (PIMRC), 2010 IEEE 21st International Symposium on , vol., no., pp.2471,2476, 26-30 Sept. 2010

[27] Antti Valmari, The State Explosion Problem, Lectures on Petri nets: advances in Petri nets Springer-Verlang, Berlin-Heidelberg, 1998, 429–473

[28] Legay A, Delahaye B, Bensalem S (2010) Statistical model checking: an overview. In: Runtime verification (RV). LNCS, vol 6418. Springer, Berlin, pp 122–135

[29] Younes, H.L.S., Musliner, D.J.: Probabilistic plan verification through acceptance sampling. In: AIPS Workshop on Planning via Model Checking, pp. 81–88 (2002)

[30] UPPAAL.org, 'UPPAAL', 2015. [Online]. Available: http://www.uppaal.org/. [Accessed: 20 April 2015].

[31] A. David, K. Larsen, A. Legay, M. Mikučionis and D. Poulsen, 'Uppaal SMC tutorial', International Journal on Software Tools for Technology Transfer, 2015.

[32] State of Florida, Department of Transportation: Traffic Engineering Manual. [Online] Available at: `http://www.dot.state.fl.us/trafficoperations/Operations/Studies/TEM/FDOT_Traffic_Engineering_Manual_revised_February_2015.pdf` [Accessed 27 April 2014].

[33] Thomas C. Hales: The Jordan Curve Theorem, Formally and Informally. The American Mathematical Monthly 114(10): 882-894 (2007)

[34] Fairfield, N.; Urmson, C., "Traffic light mapping and detection," Robotics and Automation (ICRA), 2011 IEEE International Conference on, vol., no., pp.5421,5426, 9-13 May 2011

[35] Alonzo Kelly , "A Vector Algebra Formulation of Kinematics of Wheeled Mobile Robots," tech. report CMU-RI-TR-10-33, Robotics Institute, Carnegie Mellon University, August, 2010

[36] R. Alur and D.L. Dill, “A theory of timed automata,” Theoretical Computer Science, vol. 126, pp. 183–235, 1994.

[37] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, Quentin Menet, Christel Baier, Marcus Grosser and Marcin Jurdzinski, "Stochastic Timed Automata," Logical Methods in Computer Science, vol. 10, no. 4, 2014.

[38] A. Fehnker, R. van Glabbeek, P. Hofner, A. McIver, M. Portmann, and W. L. Tan, “Automated analysis of AODV using UPPAAL,” in 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2012). Tallinn, Estonia: Springer, March 2012, pp. 173–187.

[39] A Tutorial on Uppaal, Gerd Behrmann, Alexandre David, and Kim G. Larsen. In proceedings of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT’04). LNCS 3185.

[40] B. Bonakdarpour, ’Model Checking Timed Automata, material from "Principles of Model Checking" by C. Baier and J-.P Katoen’, School of Computer Science, University of Waterloo, 2013.

[41] Seniorsecondary.tki.org.nz, ’Understanding true probability / AO S8-4 / AOs by level / Achievement objectives / Mathematics and statistics / Home - Senior Secondary’, 2013. [Online]. Available: http://seniorsecondary.tki.org.nz/Mathematics-and-statistics/Achievement-objectives/AOs-by-level/AO-S8-4/Understanding-true-probability. [Accessed: 13- May- 2015].

8 APPENDIX A

8.1 APPENDIX A.1 - AMP-IP Safety Message Fields

8.1.1 ENTER and CROSS messages

-Vehicle ID: Vehicle unique Identifier.
-Current Road Segment: Departure road of the vehicle.
-Current Lane: Departure lane of the vehicle.
-Next Road Segment: Arrival road of the vehicle.
-Next Vertex: Arrival lane of the vehicle.
-Arrival-Time: Time at which the vehicle wants to arrive at the intersection.
-Exit-Time: Time at which the vehicle plans to exit the intersection.
-Trajectory Cells List: Ordered list of cell numbers which the vehicle will occupy along its trajectory.
- Trajectory Cells Arrival Time List: Estimated arrival times in each cell found in the Trajectory cells list.
-Message Sequence Number: Sequence number of message type (ENTER or CROSS).
-Message Type : Message Type (ENTER or CROSS).

8.1.2 EXIT message

-Vehicle ID: Vehicle unique Identifier.
-Message Sequence Number: Sequence number of EXIT messages.
-Message Type: Type of message is EXIT.

8.2 APPENDIX A.2 - AIM Message Fields

8.2.1 REQUEST message

-Vehicle ID : Vehicle unique identifier.
-Arrival Time : Time at which the vehicle wants to arrive at the intersection.
-Arrival Lane : Lane identifier of the lane at which the vehicle arrives at the intersection.
-Turn : The way the vehicle will turn after reaching the intersection.
-Arrival Velocity : Velocity at which the vehicle arrives with at the intersection.
-Maximum Velocity : Vehicle dynamics does not allow the vehicle's velocity to exceed this value.
-Maximum Acceleration : Vehicle dynamics does not allow the vehicle's acceleration to exceed this value.
-Minimum Acceleration : Vehicle dynamics does not allow the vehicle's deceleration to exceed this value(in negative).
-Vehicle Length : Length of the vehicle.
-Vehicle Width : Width of the vehicle.
-Front Wheel Displacement : Perpendicular distance between the farthest point at

a vehicle's front body to the front axle.
-Rear Wheel Displacement : Perpendicular distance between the farthest point at a vehicle's front body to the rear axle.
-Max Steering Angle : Maximum angle at which the front wheels can be steered.
-Max Turn Per Second : Maximum rate at which the vehicle is able to turn its wheels.
-Emergency : An indicator informing the intersection manager that this vehicle is an emergency vehicle.

8.2.2 CHANGE-REQUEST message

Same fields as in the Request message, in addition to one more field:
-Reservation ID : Reservation identifier for the reservation that needs to be changed.

8.2.3 CANCEL message

-Vehicle ID : Vehicle unique Identifier.
-Reservation ID : Reservation identifier for the reservation that needs to be canceled.

8.2.4 DONE message

-Vehicle ID : Vehicle unique identifier.
-Reservation ID : Reservation identifier for the completed reservation.

8.2.5 CONFIRM message

-Reservation ID : Unique reservation identifier for the reservation that this confirmation corresponds to.
-Arrival Time : Time at which the vehicle is expected to arrive at the intersection.
-Early Error : The vehicle receiving this confirmation is allowed to arrive at the intersection prior to the Arrival Time by this amount.
-Late Error : The vehicle receiving this confirmation is allowed to arrive at the intersection after the Arrival Time by this amount.
-Arrival Lane : Lane identifier of the lane at which the vehicle arrives at the intersection.
-Arrival Velocity : The Velocity at which the vehicle receiving this confirmation is expected to be traveling with when it arrives at the intersection.
-Accelerations : Acceleration values and corresponding durations that the vehicle receiving this confirmation should follow while crossing the intersection.

8.2.6 REJECT message

-Stop Required : Boolean field that informs the vehicle receiving this message about whether it needs to come to a complete stop before entering the intersection or no.

8.2.7 ACKNOWLEDGE message

-Reservation ID : Reservation identifier of the completed or canceled reservation.