Validating The Maneuver Coordination Protocol With UPPAAL

Rens Leendertz
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
r.r.leendertz@student.utwente.nl

ABSTRACT

In the future, autonomous vehicles will be able to communicate with each other on an ad hoc basis. Using this infrastructure, they will be able to cooperate to improve traffic flow. One recently proposed method to do that is the Maneuver Transport Protocol. In this paper, this protocol is tested for effectiveness through simulation. For this, the modeling tool Uppaal SMC is used. The model abstracts away from right-of-way rules and complex vehicle control. In the end, it is shown that the protocol is effective. Comparison to related work shows that the abstracted model still behaves like actual traffic. The main contribution of this paper is the validation of the Maneuver Transport Protocol with on-the-fly predictions. Other efforts have simulated the protocol but made use of precalculated trajectories.

Keywords

Autonomous Vehicles, Maneuver Coordination, V2V, Uppaal.

1. INTRODUCTION

In the future, autonomous vehicles (AV's) will become a part of normal traffic. They will be able to sense their environment, act to prevent crashes, and get their passengers to their destination. However, these individual decisions do not always result in the best traffic flow. For example, other cars need to make place for you to merge onto a highway. Human drivers can anticipate and do this, but autonomous vehicles do not have this built-in. For a better result, multiple autonomous vehicles can work together to create the best outcome possible. This cooperation can be achieved by the Maneuver Coordination Protocol[7], which allows cars to inform each other of their planned routes and negotiate a better one. Besides the single example present in the paper that proposes this protocol, there has not yet been any research to check the efficiency of this protocol. This paper aims to fill that gap. To answer this, the following research questions are used.

1. What abstractions are needed when modeling (autonomous) traffic from scratch given limited time?
2. Are there cases in which the traffic rules do not result in the greatest traffic efficiency?
3. Does the use of the maneuver coordination protocol yield greater traffic efficiency?

33rd Twente Student Conference on IT July. 3rd, 2020, Enschede, The Netherlands.

To answer question 2 and 3, a model is created in Uppaal. Time is limited, requiring a number of abstractions to create a working model within the time. With this model, a 2-vehicle scenario is explored in which the Maneuver transport protocol yields better throughput than standard right-of-way rules.

In section 2 the Maneuver Coordination Service is explained. After this, the tool Uppaal is discussed. Section 4 contains a description of the design choices and the model implemented, as well as some earlier attempts that failed. Then the scenario that is tested is discussed in section 5. Section 6 contains the results of the simulations after which section 7 discusses the results and makes a comparison to related work. Finally, section 8 contains conclusions and recommendations for future work.

2. MANEUVER COORDINATION SERVICE

The Maneuver Coordination Service allows autonomous vehicles to exchange trajectories. Each car sends a message 10 times per second using the vehicles' ad hoc communication capabilities. The Maneuver coordination Protocol makes use of this service to send two types of trajectories:

A) A *planned* trajectory is always sent, describing the actions a vehicle is going to make. The plan can of course change if, for example, a new car is detected that has right of way over this vehicle.

B) A *desired* trajectory can also be sent, describing a route that the vehicle would rather take than its planned trajectory. This route is currently not feasible because of some other vehicles that are blocking the path. The vehicle sending this trajectory is called the *requesting vehicle*. All vehicles blocking the path are called the *accepting vehicles*.

Cars can find conflicts between their planned path and other vehicles by looking at all received planned paths. If they find out that they do not have right-of-way over the other vehicle, they have to select a new trajectory. They can then create a desired trajectory as well, describing a path they like better.

When receiving a message with a desired trajectory, each car will determine if their path intersects with the desired path they received. If that is the case, the car is an accepting vehicle. Each accepting vehicle decides for themselves if they want to accept this desired trajectory by evaluating new trajectories that do not intersect with the received

desired trajectory. A car accepts the desired trajectory by adapting a new trajectory that does not intersect with the desired trajectory. The requesting vehicle can determine if his request has been granted by checking its desired trajectory against new trajectories received from the accepting vehicles. The request will expire after a timeout if it has not been fulfilled.

The effect of this protocol is two-fold: It increases the predictability of other vehicles because all vehicles tell each other what trajectory they plan to take, and IT allows vehicles to cooperate and find better solutions to their situation, by sending and accepting desired trajectories.

2.1 Communication

To send these messages, vehicles will make use of inter-vehicular communication using the IEEE 802.11p standard.[5] Recently, Standards have already reserved a dedicated for this service to be run without any other services interfering. [2] Inter-vehicular communication has its own challenges such as congestion and packet loss. Since these challenges are not dependent on the message type, a lot of research has already happened in the context of other safety-critical messages. [4] In the end it is reasonable that the standard and technology will be good enough to support safety-critical messages such as this one, though it should be noted that autonomous vehicles should be able to drive safely without any communication. The Maneuver Coordination Service should only enhance the possibilities of the vehicle.

Since the trajectories sent in the messages cover a longer time (though not yet set in stone, 10 seconds would not be unreasonable), losing a small number of packets should not negatively impact the ability to predict the paths of other vehicles; there is 9.9 seconds worth of trajectory already available. For the desired trajectories it can cause more trouble, as some vehicles might not receive the request on time. The timeout set for each request can give a bit of leeway, such that the desired trajectory is received in a later packet, and still have time to accept the request. At the same time, if the network would be completely overloaded (e.g. a cyberattack), the timeout ensures that vehicles will not keep on waiting for their request to be fulfilled.

2.2 Extensions

Though the protocol examined in this paper will be the original proposed by Lehmann [6], another paper by Xu et al. proposed an extension on the maneuver transport protocol [7]. This paper is mentioned because it is the only other work related to this protocol. It also performs some simulations which are used for comparison of results. To simulate the Maneuver Transport Protocol they created a set of trajectories before their tests, and the vehicles used those to come to an agreement. This paper is different because it generates the trajectories during run time and tests the original protocol instead of the extension.

3. UPPAAL

To model these cars and their communication, UPPAAL SMC is used. UPPAAL [1] allows for modeling and verification of real-time systems, modeled as a set of timed automata with some extensions. Standard UPPAAL does an exhaustive exploration of the state space, meaning it can determine with certainty if some property holds. To model autonomous vehicles however, A more powerful model is needed: UPPAAL SMC [3]. The SMC tool allows for stochastic variables and custom clock rates, at the cost of using statistical model checking instead of formal validation. This means that instead of searching the entire state space, it runs a simulation over and over again, creating a statistically significant value in the process.

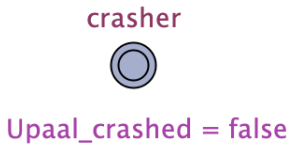

Figure 1. The automaton to crash Uppaal SMC

3.1 Queries

Retrieving statistically significant values is done by performing queries. A query asks Uppaal to measure the value of an expression over multiple runs until some statistical boundary is met (e.g. the value is determined to be in some range with a 95% certainty).

3.2 Tricks

Uppaal SMC has some particularities that were found during the making of the model. They are described here for future researchers working with Uppaal.

3.2.1 Clocks within functions

When using a clock inside the coding language, Uppaal treats them differently from doubles. This results in simple inequality checks not producing the correct types. To get around this, a simple function can be written like this:

```
double getClockName(){
  return copysign(clockName, clockName);
}
```

The copysign copies the sign of the first variable to the second, resulting in the same value as output. This function is provided by Uppaal. This is a side-effect-free function, meaning it can be used inside guard functions as well.

3.2.2 Debugging complex states

When using Uppaal's query language, it can be hard to determine that your model is correct. There are 2 ways to retrieve the actual values of variables. First, the simulate query allows for graphing of variables over time, which is useful for clocks etc. Second, you can force Uppaal to crash by violating the model sanity. This only works with Uppaal SMC, as it only runs a simulation, instead of searching the entire state space. Uppaal then spits out a giant error message containing the value of all variables, thus allowing you to examine the complete state. To do this, one needs an extra automaton as shown in figure 1. The Boolean variable 'Uppaal_crashed' can be stored in global memory, to allow access from any other automaton. Then you can set this variable to true at any point in the code, allowing you to crash Uppaal at the exact moment you suspect something to be wrong. Though all the data is outputted, the way in which this is done is not very usable, especially when using arrays or other structures. It is doable to make this data a bit more readable using some regex and programming especially because Uppaal comes with a command-line tool to run queries yourself. A robust tool that does this for you could be a nice addition to help all researchers working with Uppaal.

4. MODEL

Using Uppaal a model has been created. First, the design choices used to make the model are described, after that the implementation.

4.1 Design Choices

The complexity of real-world traffic and right-of-way rules, as well as the planning needed for autonomous vehicles to create a safe path, required more abstractions to be made than originally was intended.

Roads

Roads are described on a 2d plane, stretching infinitely in the X-direction. Each integer Y describes a separate lane, with 0 being the first. This allows vehicles to have a simple goal (get as far on the X-axis as possible, which is still realistic (e.g. driving down a highway).

Obstacles

To create the need for lane changes, obstacles are introduced. They are characterized by a coordinate. All vehicles must avoid the obstacles by changing lanes. By using obstacles, different phenomena can be created. A single obstacle could be a broken-down car, a series of obstacles close together form a closed down road, etc.

Vehicles

All vehicles have an (x,y) coordinate describing their location on one of the lanes. Lane changes and speed changes are instant, to allow for simplification of predicted paths. They can only adapt 4 distinct speeds (1, 2, 3, or 4 per unit of time) to reduce computation time. Vehicles have to keep one second worth of traveling distance to each other (so 4) to ensure safety. This is similar to the real world, where cars need to keep their distance to the preceding vehicle as well.

Trajectories

Instead of using Frenét frames to describe trajectories, as suggested in the original paper [6], a list of coordinates combined with a starting time is used to describe a trajectory. The math behind Frenét frames was too complex to be used in Uppaal. Furthermore, coordinates allow for easy comparison of different trajectories for clashes, because the exact locations can be compared.

Communication

The Maneuver Transport Message is stored in shared memory, only the latest being saved. Uppaal does not have the means to allow automata to communicate directly. All vehicles communicate at the maximum rate allowed (10x per second).

Right-of-way

Instead of implementing the correct right-of-way rules, each car is assigned 2 priorities. The model treats higher priority cars as having right of way. One priority defines normal behavior, when no requests are being sent, the second priority is used when that car sends a request. This way requesting cars can have their requests be accepted, even though they did not have a higher standard priority.

4.2 Implementation

In the end, the complete model is described in code, with a single state automaton for each car, as shown in figure 2. The smallest time units are hundredths of seconds, as the smallest distance units are hundredth of a meter. This allows for the use of integers in all of the calculations done in Uppaal. Below is a subset of defined functions are listed, to describe how the vehicles work.

Create_path

This function creates a valid trajectory, taking into account the obstacles described above. It takes as input a specific speed, creating a path with that constant speed for the entire duration. Because it does not take other vehicles into account, this function alone is not enough to create a safe trajectory.

Simulate

The *simulate* function takes a set of trajectories as created by the *create_path*, and tries to determine their safety. If this set of paths leads to a crash, the function returns the time at which the simulation failed.

Combine_paths

Combine_paths takes a set of trajectories of other vehicles and tries to find a safe path for the vehicle itself. It does this by running *Simulate* with all paths generated by *create_path*. It returns the fastest safe path.

Create_requested_path

This function acts much like *create_path*. It generates requested trajectories. One interesting thing to note here, is that *create_requested_path* uses another paths' starting phase as its own. If a request would be send out to drive on maximum speed only, but the vehicle itself had to slow down because there was no other option that was safe, the request becomes unfeasible immediately. By going with the flow of the new planned trajectory, it is possible to continue with the requested trajectory after the other vehicles accepted and some time has passed.

Controller

The *controller* is in charge of communication and path generation. It retrieves the messages sent by other vehicles, runs *Combine_paths* with other planned trajectories to get a safe trajectory, and does everything related to requests. It checks if a current request has been granted, creates a new request if no request is active and the car is not driving at maximum speed and discards an open request if the time limit is reached. It also tries to honor other vehicles' requests if they are found in the messages. Last, the controller sends a new message to the other vehicle with updated information.

Movement_update

This function performs the act of moving the vehicle forward. It uses the path planned by the *Controller* to move.

Update

The *update* function is run every tick of the simulation, as can be seen in figure 2. it always runs *Movement_update*, and it runs *Controller* 10 times per second.

4.3 Attempts

The model underwent a lot of design iterations. Not all design choices described above were used from the start. Attempts were made to create a more complex system, but these failed. Here a few of these are described along with the reasons they did not result in a working system.

At first, vehicle movement was more complex. They had a clock governing their X-coordinate and speed, allowing for more realistic simulation of vehicle movement. This turned out to be a bad idea because the vehicle needed a way to determine its future locations. By making use of simulation-specific features (e.g. the passing of time of clocks) to simulate the X coordinate, The same system would be needed to allow vehicles to predict their own paths. This essentially means that for every prediction a

car wants to make, they need to run an Uppaal simulation. For this reason, the concept of gradual speed changes was removed, using a separate function update the location of the vehicle every tick instead.

Another choice made later to avoid over-complication was to separate the creation of a path and the feasibility checking of that path. Having these two things together would allow for better paths because cars could create paths that fit exactly. This turned out to be a problem too hard to solve in the given time frame, requiring backtracking and complex calculations to try and find a path that worked. To prevent this, the car would have needed human-like anticipation, seeing that they could not drive on until they were close to an obstacle and had to move to another lane, because another vehicle was blocking their path. By separating the path creation and car, no backtracking is needed. Furthermore, to reduce the number of paths created every time a car tried to make a prediction, the vehicles are limited to only 4 distinct speeds, instead of trying out every possible speed. This combination of restrictions allows predictions to happen within a reasonable time. The issue of paths not being perfectly aligned is mostly mitigated by the recalculation of the path every 10th of a second, thus allowing to adjust the path regularly when a better path becomes feasible.

As a result of the choice to split path creation and checking against other vehicles, a split was also made between obstacles and cars.

At first, obstacles were cars that did not communicate nor moved, thus behaving as a barrier. Because paths did no longer take other cars into account, The obstacle had to be a separate entity to be able to consider it when creating the path. If one would create the path without taking obstacles into account, there would be no way to change lanes and avoid them. Last, the switch from right-of-way implementation to a set of priorities was made. It turns out right-of-way is a tricky thing to program, requiring a lot of context. When two cars get too close to each other, there are a number of possible causes, without a set car that is in the right or not. For example: if you find out you are too close to a car in front of you, you probably have driven too fast and you need to slow down. However, it could also be that you are actually in the right, and the other car wanted to perform a lane swap and didn't see you. In the real world, you have to break nonetheless, because there is a car too close in front of you. However, if this occurs during a prediction, you should continue whilst the other vehicle has to adapt accordingly. This complicates things a lot. Not only is it tricky to get right, but you also have to make sure that the predictions of all vehicles yield equal results. If both cars think they are in the right, a crash is imminent. If both think the other has right-of-way they cause unnecessary congestion. Besides, it had to be possible to turn this feature off. When trying to meet a desired trajectory, the accepting vehicle needs to essentially yield their right-of-way to the requesting vehicle. It is not good enough if they aren't in the wrong, they also need to ensure that there is no crash. Combine this with the time lost trying to implement other items on this list, and there was no time left to ensure this worked properly. Set priorities would suffice.

5. SCENARIO

To test the Maneuver coordination protocol, a scenario as pictured in figure 3 is created in the model described in the previous section. The Scenario is characterized by the following parameters:

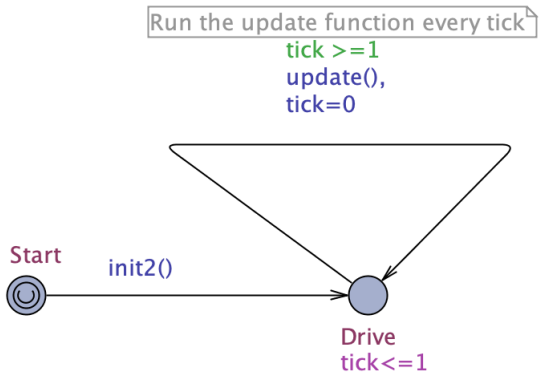

Figure 2. Autonomous vehicle model

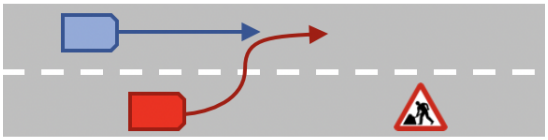

Figure 3. Lane merging scenario

- Duration: 5
- Number of lanes: 2
- Car 1:
  - Start: x = 3, lane = 0
  - Normal priority: 1
  - Request priority: 3
- Car 2:
  - Start location: x = 0, lane = 1
  - Normal priority: 2
  - Request priority: 0
- Obstacles: 1
  - location: x = 15, lane = 0

Car 1 (red in figure 3) is ahead of car 2 but has not got enough space to merge in front. It does not have right of way (characterized by its normal priority being lower).

To measure the effectiveness of the protocol, the scenario is simulated for 5 seconds. The value awarded to that specific run will be the distance 'lost' because cars were not driving at their maximum speed. Running this 100 times, an average is created.

The exact query to determine the value is as follows:

```
E [current_time <= simulation_time; 100] (
    min:
    simulation_time * max_speed * car_count -
    sum(car_id : Car_id) (
        (Car2_2(car_id).x-starts[car_id].x)
            / distance_units
    )
)
```

Simulation time will be set to 5. The query tries to minimize the total distance lost. The start X coordinate is subtracted as only the distance traveled matters. *distance_units* is a constant required by the simulation to keep values inside integer boundaries.

Table 1. Results

	protocol status	Average (100 runs)
1	Perfect run	1.1
2	Protocol enabled	1.6
3	Protocol disabled	7.1

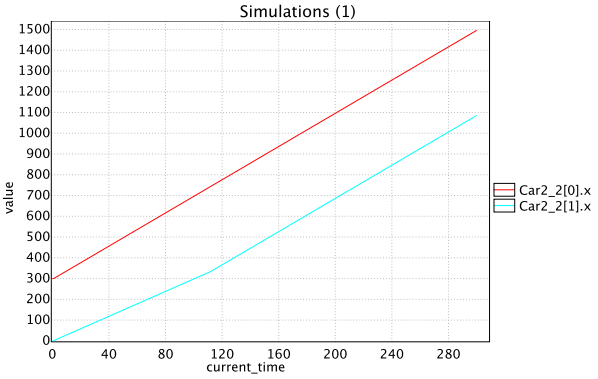

Figure 4. The perfect run

6. RESULTS

The scenario is put through the test 3 times with different priorities to simulate different behaviors. The only difference will be the (request) priorities awarded to each vehicle. The 'perfect' run gives right-of-way to Car 1 (red), allowing it to merge in front. The 'enabled' run uses the priorities as specified in the previous listing, allowing Car 1 to negotiate the path it wants to take. The 'disabled' run uses standard right-of-way rules, and disables the protocol, thus simulating what would happen without the communication in place.

The results of the tests are displayed in table 1. Besides the average score, of each test, graphs 4, 5, 6 contain The X coordinates of the two cars (their progress on the road) against time in a typical run. Both distance and time units are multiplied by 100 in the graphs, as those are the smallest units used in the simulation. Not all graphs have the same time scale, as their purpose is to show the characteristics of the run. Extending the duration would show longer straight lines as the cars have settled on their respective trajectories, and makes the characteristics harder to see. The 'perfect' run is very close to the maximum possible value; only 1.1 distance units lost. This difference can be lead back to the fact the two vehicles need to keep a safe distance from each other. They can start closer to each other because they start on different lanes. Graph 4 contains such a run for the first 3 seconds. Tough it is hard to see, the blue line makes a bend at around 110 time units, where its speed increases from 3 to 4. At that point in time, there is enough distance between the two cars for the front car to insert and the blue car to continue at maximum speed. Second-best is the run with the protocol enabled. It loses a little bit more time with a distance loss of 1.6. The extra 0.5 distance it differs from the perfect run can be lead back to the startup lag caused by the protocol. Before the cars have agreed on the better route, Car 1 has to slow down in case they cannot come to such agreement. This, in turn, requires car 2 to slow down more as well to allow car 1 in front. In graph 5, running until only 1 second, The startup lag can be seen that is caused by the protocol and the nature of beaconing messages. In last place, the run without the maneuver coordination protocol stands with 7.1 distance units lost. It

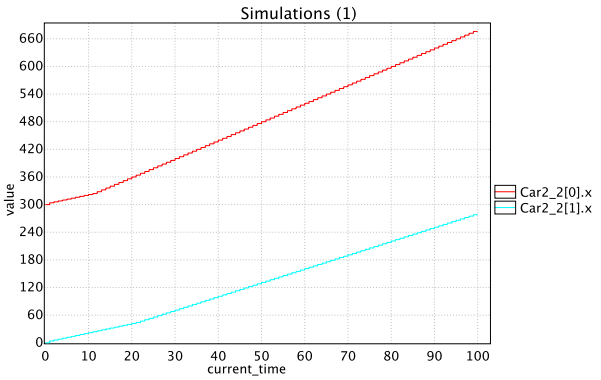

Figure 5. Typical run with protocol

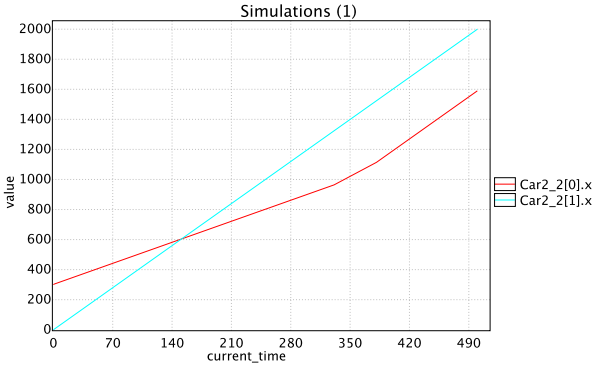

Figure 6. Typical run without protocol

has a significant loss due to car 1 having to slow down and move behind car 2. This can be seen in graph 6, where the red crosses the blue line and has to get to a safe position behind the blue line before inserting into the free lane and speeding up, continuing in sync. This graph shows the full 5 seconds used to test the different scenarios.

From the graphs, it is clear that the tests are of sufficient duration for capturing the characteristics of the different scenarios. Extending their duration would not affect the simulations, as in all scenarios, all vehicles are driving at their maximum speed, and there is no reason to slow down at a later time.

7. DISCUSSION

Creating a model of real-world traffic & autonomous vehicles proved difficult. The first iteration made use of Uppaal clocks to model speed and the right of way rules were being implemented inside functions. This proved to be too complex to get working, especially because of the complexity of predicting your own future trajectory. The small time frame given proved insufficient to set up a model close to reality. Though in the end, this scenario showed that the Maneuver Coordination Protocol performed better than standard right-of-way rules, A better environment in which to perform measurements is needed to perform rigorous testing. Uppaal does have some nice features to use when creating more complex models, but this model was too simple to use most of them.

The simplified control also showed in the 'hiccup' at the start of graph 5. Because the only variable for the trajectory is the speed, the vehicle cannot drive at a high speed but planning to slow down if things do not change. A real vehicle will probably not decide to immediately take drastic action if it detects an issue 30 seconds ahead in

time. Instead it would scope out its options, trying to resolve the conflict during those 30 seconds, only deciding that slowing down is its only option after some time has passed. Before that point the vehicle could send out its *desired* trajectory, with it being accepted before it has to slow down. This means that in reality, reaching an agreement is even closer to a perfect run, maybe even equal in time lost.

7.1 Comparison to other work

When looking at the maneuver transport protocol, there has only been one paper [7] that measured the effectiveness of the protocol. There are some caveats when comparing the two: First, that paper main goal is to propose an extension to the maneuver transport protocol which is not covered in this work. Second, they use the speed of each vehicle as a heuristic instead of total distance traveled. Third, the scenario in that paper is not completely equal to the scenario described in this paper. The concepts are the same, but in that scenario there is a third car trailing car 2 (blue in figure 3). Despite these issues, some comparisons can still be made.

First of all, they too found that the requesting vehicle has to slow down massively to be able to enter the lane behind the other cars. Second, when comparing the scenarios with the protocol activated, they found the accepting vehicle (blue in figure 3) slowing down a little bit, whilst the requesting vehicle kept driving at maximum speed. This reinforces the claim that this model, though abstracted heavily, still shows the characteristics of actual traffic.

8. CONCLUSION

In this paper, an environment was created to model the Maneuver Coordination Protocol in autonomous vehicles. Large abstractions were made, though the model still showed the same behavior as related work. In the end it is shown that the protocol increases the traffic throughput.

8.1 Abstractions

As seen in the section containing Design Choices, hefty abstractions need to be made for a functioning system to be created in the approximately 7 weeks available for this paper. If one wants to perform research involving modeled autonomous vehicles, it is strongly advised to try and find an existing tool that allows you to create scenarios, as building one from scratch proved too complex.

8.2 Traffic Efficiency

There are examples to be found where normal right-of-way rules do not yield the best throughput, such as the scenario used in this paper. Traffic rules exist to prevent accidents with human drivers, not efficiency first. This means that there are valid reasons to develop methods such as the Maneuver Transport Protocol to improve traffic efficiency.

8.3 Protocol Effectiveness

Though substantial abstractions were made, comparison to related work showed that the model retained the expected characteristics of real-world traffic. Furthermore, the performed tests showed that scenario's with the Maneuver Coordination Protocol activated, resulted in vehicles without right-of-way to negotiate outcomes with an overall higher throughput, though perfect throughput was not reached. All in all the protocol proved effective in increasing traffic efficiency.

8.4 Future Work

There are three directions for which future research can be done involving the protocol. Besides the protocol Some future work is also proposed to extend Uppaal.

First of all, effort should be put into developing a modelling tool capable of performing predictions and communicating those with other vehicles. Without such tool it would be almost impossible to perform decisive measurements. No such tool exists yet. This model restricted the number of possible paths and made some abstractions to get around this, though that would not work for a more realistic simulation. The paper referenced in the discussion tried to get around this issue by running a large amount of separate simulations to determine possible trajectories, storing those, and having them available for the vehicles to send during the real deal. This gives a huge overhead when creating a scenario and requires some level of human involvement, as they have to think about the possible scenarios that could occur. A tool that performs accurate predictions on the fly, using trajectories of other vehicles as input, allows for much more realistic simulations.

Second, research into the effect of packet loss on performance should be an interesting area to explore. It is nice that the perfect scenario is XX% more efficient, but packet loss impacts this value. If a packet is lost containing a desired trajectory, it takes longer for the vehicles to reach an agreement. This impacts the performance, as the requesting vehicle has to wait longer before making its move. By looking at packet loss rates found in other research involving the IEEE 802.11p protocol, an estimate can be made to determine the actual performance of the protocol.

Lastly, a comparison should be made between the original protocol proposed by Lehmann et al.[6] and the extension proposed by Xu et al. [7]. There are some important differences that can lead to different results in different scenarios. For example, the ability to specify more than one desired trajectory should in theory result in a larger number of agreements being met, thus increasing the extensions' throughput. However, since the extension requires three separate messages to arrive before an agreement is met, each agreement takes longer and is more susceptible to packet loss. These are interesting trade-offs and it would be interesting to see a side-by-side comparison.

8.4.1 Uppaal

For Uppaal, a tool allowing you to easily view the state of the model would be very useful. Not only for autonomous vehicles, but for Uppaal models in general. It could be very similar to the Simulator tab already present, but made to work with Uppaal SMC simulate queries.

9. REFERENCES

[1] Uppaal, www.UPPAAL.org. Accessed: 10 June 2020.

[2] C2C-CC. Guidance for day 2 and beyond roadmap. Technical report, 2019.

[3] A. David, K. G. Larsen, A. Legay, M. Mikučionis, and D. B. Poulsen. Uppaal SMC Tutorial. Technical report, Aalborg University, Denmark & INRIA/IRISA Rennes, France, 2018.

[4] G. Heijenk, M. Van Eenennaam, and A. Remke. Performance comparison of IEEE 802.11 DCF and EDCA for beaconing in vehicular networks. In G. Norman and W. Sanders, editors, *Quantitative Evaluation of Systems. QEST 2014. Lecture Notes in Computer Science)*, volume 8657, pages 154–169, Cham, 2014. Springer International Publishing.

[5] D. Jiang and L. Delgrossi. IEEE 802.11p: Towards an international standard for wireless access in vehicular environments. In *IEEE Vehicular Technology Conference*, pages 2036–2040, 2008.

[6] B. Lehmann, H. J. Günther, and L. Wolf. A Generic Approach towards Maneuver Coordination for Automated Vehicles. In *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, volume 2018-Novem, pages 3333–3339. Institute of Electrical and Electronics Engineers Inc., 2018.

[7] W. Xu, A. Willecke, M. Wegner, L. Wolf, and R. Kapitza. Autonomous maneuver coordination via vehicular communication. *Proceedings - 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop, DSN-W 2019*, pages 70–77, 2019.