Crossroads — A Time-Sensitive Autonomous Intersection Management Technique

by

Edward Paul Andert III

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved January 2017 by the
Graduate Supervisory Committee:

Aviral Shrivastava, Chair
Georgios Fainekos
Hani Ben Amor

ARIZONA STATE UNIVERSITY

May 2017

ABSTRACT

For autonomous vehicles, intelligent autonomous intersection management will be required for safe and efficient operation. In order to achieve safe operation despite uncertainties in vehicle trajectory, intersection management techniques must consider a safety buffer around the vehicles. For truly safe operation, an extra buffer space should be added to account for the network and computational delay caused by communication with the *Intersection Manager* (IM). However, modeling the worst-case computation and network delay as additional buffer around the vehicle degrades the throughput of the intersection. To avoid this problem, AIM Dresner and Stone (2004), a popular state-of-the-art IM, adopts a query-based approach in which the vehicle requests to enter at a certain arrival time dictated by its current velocity and distance to the intersection, and the IM replies yes/no. Although this solution does not degrade the position uncertainty, it ultimately results in poor intersection throughput. We present Crossroads, a time-sensitive programming method to program the interface of a vehicle and the IM. Without requiring additional buffer to account for the effect of network and computational delay, Crossroads enables efficient intersection management. Test results on a 1/10 scale model of intersection using TRAXXAS RC cars demonstrates that our Crossroads approach obviates the need for large buffers to accommodate for the network and computation delay, and can reduce the average wait time for the vehicles at a single-lane intersection by 24%. To compare Crossroads with previous approaches, we perform extensive Matlab simulations, and find that Crossroads achieves on average 1.62X higher throughput than a simple VT-IM with extra safety buffer, and 1.36X better than AIM.

Dedicated to my parents, for without whom I would never have made it.

ACKNOWLEDGMENTS

Although it is likely impossible to acknowledge all of the people and contributions that have resulted in this thesis and degree, I have make a valiant effort with the words that follow.

First and foremost I would like to thank my professor, Dr. Shrivastava for encouraging me to pursue a Masters degree when I was an Undergraduate working on my Capstone project under his guidance. Through your instruction over the years, I was able to take all the experiences I gained from the projects over the years and use them to create this thesis.

I would like to thank my fellow members of the Compiler Microarchitecture Lab at ASU for helping and encouraging me through both my studies, as well as the many project that eventually became this thesis. Your encouragement and questioning of my work helped shaped what it became. Also, thank you for putting up with all the noise that my experiments produced! I would like to give a special thanks to Dheeraj for providing me with a sense of guidance as well as the motivation to finally finish my degree. It is definitely uncertain I would have ever finished this degree without you!

Thank you to my family for all they have done to help me finish this degree. To my mom, Nancy, for always being there with encouragement even when I disappeared from communication because of an impending deadline and for making sure I always knew it didn't matter how long it took as long as I got there. To my dad, Ed, for somehow always finding a way to proofread my proposals and business schemes that I sent at the last minute and for willingly having conversations with me about whatever new thing I had learned or crazy idea I had come up with. To Elissa, for keeping my closet stocked so that I didn't show up to presentations looking like a bum. To Joe, for making sure my vehicles were in working order. To my grandma, Beth, for

keeping me encouraged while making sure I had all the necessities for a comfortable survival. And last but not least to my younger brother, Mitchell, for serving overseas so that I was able to get this degree in the safety of home. It goes without saying that your sacrifices deserve more thanks than a simple acknowledgement in this thesis document.

I would also like to thank my girlfriend, Erica, for putting up with my ridiculous and variable schedule. Thank you for keeping me sane when I inevitably overcommitted myself by taking on too many jobs while simultaneously dealing with schoolwork and my research. Without your encouragement and help, I may not have made it through this will all my sanity!

Finally, thanks to my friends for all the distractions they provided that kept me on track during this ordeal! As well as a thanks to all the others who helped me along the way that I could not fit in here due to the number of pages it would take!

TABLE OF CONTENTS

Page

LIST OF FIGURES vi

CHAPTER

1 INTRODUCTION 1

2 SETUP TO EVALUATE IM 4

3 SAFETY BUFFER CALCULATION 6

3.1 Estimating Sensing Error 6

3.2 Estimating Time Synchronization Error 7

3.3 Timing Problems in VT-IM 8

4 TIMING PROBLEMS IN VT-IM 11

5 RELATED WORKS 14

5.1 Velocity Transaction Based IMs 14

5.2 Query-Based IMs 15

6 OUR TIME-SENSITIVE TECHNIQUE 18

7 EMPIRICAL EVIDENCE 21

7.1 Time-sensitive Programming Enables Efficient IM 21

7.2 Crossroad Scales Well 22

8 CONCLUSION 25

REFERENCES 26

LIST OF FIGURES

Figure		Page
1.1	Five Vehicles in a Real Implementation. Top Speed 3.0 m/s. Intersection Lines Overlayed After the Test.	3
2.1	Vehicle Modeled With Lateral, Longitudinal Error Including Round-Trip Delay.	5
3.1	Expected Velocity Versus Actual Velocity Due to Control Algorithm Errors, and Sensor Errors.	7
3.2	RTD Causes Late Command Delivery.	10
4.1	RTD Causes Late Command Delivery.	13
6.1	Vehicles Receiving Command From IM with Different Round Trip Delays.	18
6.2	Example of a Vehicle Trajectory Based on Max Acceleration	20
7.1	Average Wait Time Comparison for Vehicles in Four Different Cases of Our Physical Implementation.	22
7.2	Throughput and Computation Time for Different Input Flow Rates	24

Chapter 1

INTRODUCTION

As vehicles become autonomous, intersections are no longer constrained by the humans that are currently driving and instead, automated *Intersection Managers* (IMs) can make intersections safer and more efficient. Vehicle to Infrastructure (V2I) IMs interact with the vehicles as they approach the intersection to find a safe and efficient way to operate the intersection.

There are two main ways to design the interface between the vehicles and the IM. The first is the most intuitive one, which we call *Velocity Transaction* IM (or VT-IM). In a VT-IM the approaching vehicle announces its arrival to the IM, and the IM responds with a velocity command to follow that will ensure safe and efficient operation of the intersection. To guarantee the safety of vehicles in any intersection, a safety buffer must be considered around the vehicle which accounts for the uncertainty in the position and velocity of the vehicle. The uncertainty in the position and speed of the vehicle stem from different sources such as errors in various sensors and actuators, inaccuracies in the data fusion algorithms, and even due to the clock synchronization drift between the vehicle and the IM. To ensure the safe operation of a VT-IM, the IM must also take into account the network and computation delay related to the interaction between the vehicle and the IM. Here, the network delay is the variable lag in delivering information to the IM and then back to the vehicle, and computational delay is the time it takes for the IM to compute the correct response to send back to the vehicle. Network and computation delay together, compose *Round Trip Delay* (RTD). Neglecting RTD makes VT-IM scheduling methods vulnerable to uncertainties and may lead to accidents. So, a time buffer should be considered to

account for the worst-case RTD.

One way to avoid having this large RTD buffer is to use a *Query-Based* IM (QB-IM) design. This approach is quite popular, and is used in the state-of-the-art work AIM Dresner and Stone (2004, 2008). In this, the vehicle approaches the intersection at a constant speed, and sends a speed query to the IM. The IM simulates vehicle trajectory and replies with a yes/no answer. If the answer is yes, the vehicle can continue moving with the speed, and if the answer is no, the vehicle slows down to a lower speed, and makes a request again. A QB-IM approach does not incur error in the position of the vehicle due to RTD, and therefore the RTD buffer is not required. However, in such an IM [1] design there is not much scope for the IM to optimize the traffic at the intersection. In particular, the QB-IM design cannot solve an optimization problem and send its result to the vehicles because it can only give a yes/no answer. Ultimately, a QB-IM results in less intersection throughput.

To solve this RTD buffer problem correctly, we present our approach Crossroads. Crossroads is a time-sensitive method to program a VT-IM, without requiring the addition of an RTD buffer. Crossroads solves the RTD buffer problem by fixing the action time of the target velocity received by the vehicle, so that the position of the vehicle becomes deterministic. We use 1/10 scale models and design several traffic scenarios to test the two IM techniques, the VT-IM which requires the RTD buffer, and Crossroads that does not. We also implemented all the three IMs (the simple VT-IM, Crossroads, and AIM) in Matlab to study the scalability of our approach. We run the intersection simulation on randomly generated vehicle input sets. We found that at low input rates, all the techniques perform almost the same, however, as input rate increases, the throughput of VT-IM drops sharply. QB-IM works better,

[1] In this paper we use the acronym IM for both intersection manager and intersection management. The correct word will be clear from the context.

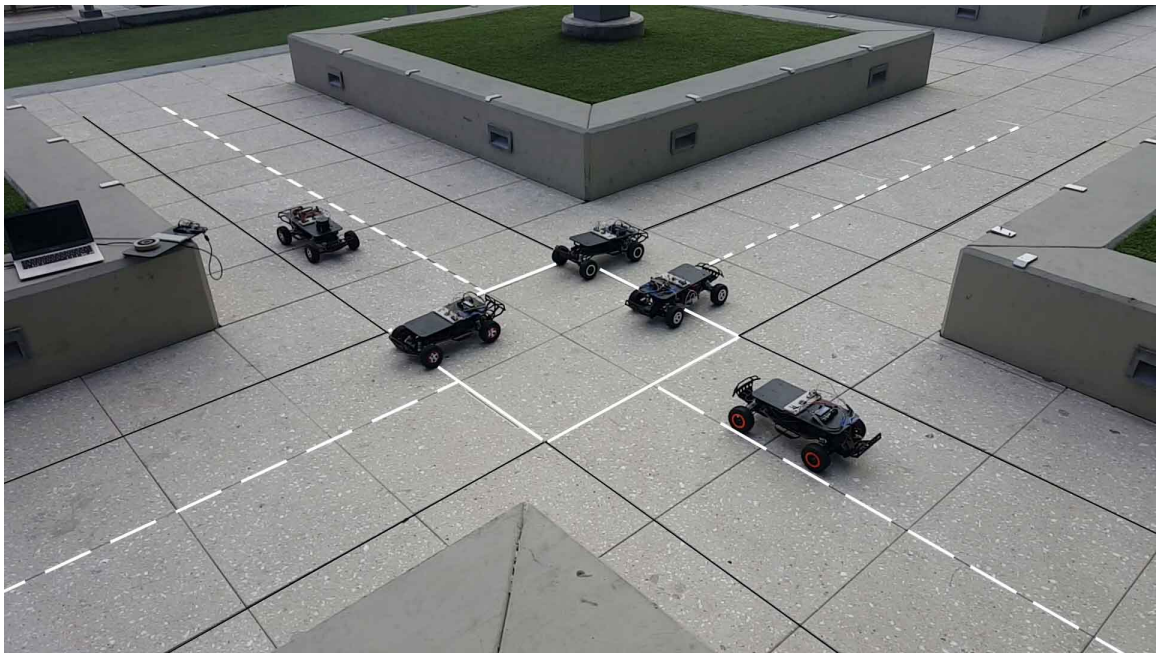

Figure 1.1: Five Vehicles in a Real Implementation. Top Speed 3.0 m/s. Intersection Lines Overlayed After the Test.

but Crossroads scales better. On average over all input flows, Crossroads has 1.62X better throughput ($\frac{number\ of\ vehicles\ per\ second}{average\ delay\ per\ vehicles}$) than VT-IM, and 1.36X better than AIM. The performance overhead and network traffic of Crossroads and VT-IM is up to 20X lower than AIM.

Chapter 2

SETUP TO EVALUATE IM

Most IM techniques have been evaluated on simulators, therefore researchers did not encounter timing and error-related problems. In order to test the effectiveness of IM policies on a physical intersection, we created a 1/10 scale model. As shown in Figure 1.1, we have designed a four-way intersection with one lane per road. The size of the intersection is $1.2 \times 1.2m^2$ and the vehicle length and width are $0.568m$ and $0.296m$ respectively. Each vehicle will start communicating with the intersection manager when crossing a designated transmission line. The distance between intersection and designated transmission line was chosen to be $3m$. The maximum velocity of the vehicles is limited to $3m/s$.

We used RWD (rear wheel drive) Traxxas Slash RC as the chassis. A quadrature encoder was installed on the motor to measure vehicle velocity. Arduino Mega 2560s were used as the central control unit of the vehicles. Bosch BNO055 9DOF sensor fusion IMUs were used for steering feedback. For wireless communication, NRF24L01+, 2.4GHz serial network adapters were utilized. Our setup is similar to Fok *et al.* (2012).

The vehicle interaction with the IM is implemented broadly as a state machine with four states: i) Arriving state: The vehicle is in this state before it reaches the transmission line. ii) Sync state: Once the vehicle reaches the transmission line, it registers with the IM, and sends a sync request to the IM. The IM sends back the time synchronization data (based on NTP) Mills (1991). iii) Request state: Once the time sync is achieved, the vehicle transmits a packet of data to the IM requesting to make an intersection crossing. After processing the requests ahead in a FIFO queue,

the IM computes the response for the vehicle and sends the response (e.g., proceed at a particular speed). vi) Follow state: Once the vehicle receives the plan from IM, the vehicle then follows the plan, and when it crosses the intersection, the vehicle sends an exit timestamp to notify the IM, and goes back to the Arriving state for the next intersection. The exit timestamp allows us to track wait time of each vehicle.

Different IMs are implemented and run on a laptop with 10 GB memory, Core i7 -3517u @1.9/2.4 GHz CPU and Windows 8.1 64-bit OS. The IMs are written in Matlab R2016.

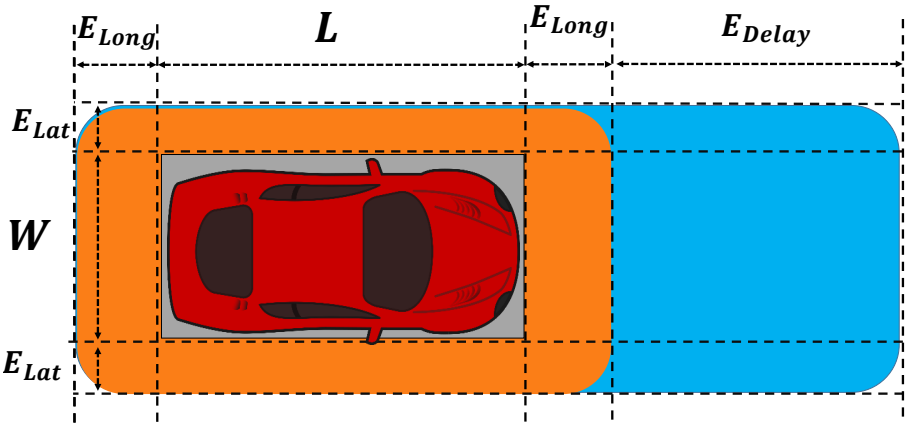

Figure 2.1: Vehicle Modeled With Lateral, Longitudinal Error Including Round-Trip Delay.

Chapter 3

SAFETY BUFFER CALCULATION

In any physical system (not simulation), there are always uncertainties/inaccuracies in identifying the exact state of the system. In our case, the uncertainty is in the position and velocity of the vehicles. This positioning uncertainty can be due to several reasons: sensor errors (in our case, position and speed sensors), state estimation algorithms used (for example, if a GPS and IMU are used to estimate the position and the velocity of the vehicle, the sensor fusion algorithm can affect the accuracy of the position and velocity), and even due to the difference in the clocks of the different components of the system (in our case, the synchronization error between the IM and the vehicle). In order to achieve safe operation of the intersection, a safety buffer must be modeled around the vehicles. The safety buffer essentially implies that the vehicle can be anywhere within the buffer and the movement of the vehicles must be planned/implemented such that the safety buffers do not overlap at any moment in time.

3.1 Estimating Sensing Error

Vehicle positioning is based on acquired measurements from sensors. An IM design must take into account the error propagated from GPS, encoder, etc. It should be noted that an encoder error would affect the vehicle longitudinally, whereas GPS error would affect a vehicle both laterally and longitudinally. Figure 2.1 depicts both cases. Although it may be possible to estimate the size of safety buffer around the vehicles using the error numbers from the data sheets of the sensors, it is still hard to estimate the effect of the data fusion and control algorithms on the buffer. Therefore,

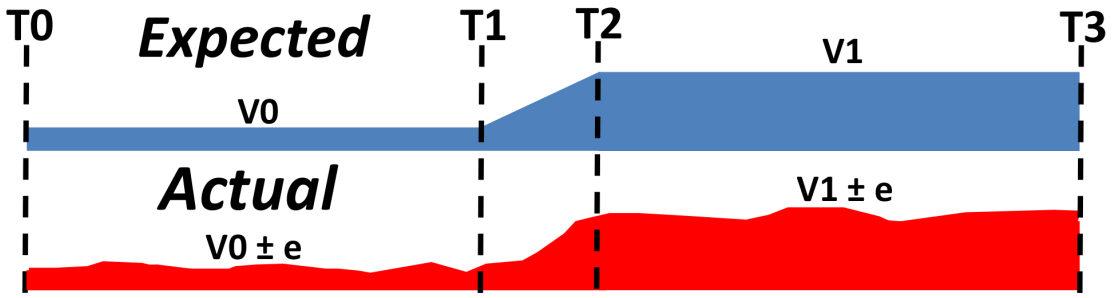

Figure 3.1: Expected Velocity Versus Actual Velocity Due to Control Algorithm Errors, and Sensor Errors.

we devise an experiment to estimate the error in the overall position and velocity of the vehicles, and use that to estimate the safety buffer size.

As shown in Figure 3.1, we start the experiment with the vehicle at position P_0, with velocity v_0, at start time T_0. The vehicle then attempts to hold velocity v_0 until it reaches time T_1. At T_1, the vehicle accelerates until it reaches velocity v_1 at time T_2. The vehicle then maintains the velocity v_1 until time T_3. Suppose that ideally the vehicle should have reached position P_3 at time T_3, then the error in the final position will be $E_{long} = P_3 - P_{actual}$. The worst-case positive control error will happen in our model when $v_0 = 0.1m/s$ and $v_1 = 3.0m/s$. And the worst-case negative error will happen when $v_0 = 3m/s$ and $v_1 = 0.1m/s$. Using the worst-case of these two test, we can determine the outer bound of our longitudinal error E_{long}, which will consequently become the safety buffer. We perform this experiment 20 times, and measure E_{long}. The maximum value of E_{long} was $\pm 75mm$ before adding Synchronization error.

3.2 Estimating Time Synchronization Error

Our physical implementation is a distributed system containing multiple nodes communicating with the central server. Without proper synchronization, commands given to nodes can be executed at different times, depending on when the command

is received. Synchronization is the solution to having the same understanding of time among the nodes. Different time synchronization methods like NTP, PTP, GNSS, etc. can be used in order to synchronize with the server. We utilize NTP (Network Time Protocol) for synchronization in our setup. Our time synchronization error with NTP is well defined and is 1 millisecond over the course of the test. Therefore, at maximum speed $3.0m/s$, the safety buffer due to time synchronization error is 3mm.

Overall in our system, $E_{long} = \pm 78mm$. E_{lat} is not the focus of this research and we make the assumption that all vehicles entering our intersection can maintain proper lateral position, therefore it can be disregarded.

3.3 Timing Problems in VT-IM

Velocity Transaction IMs or VT-IMs work in a manner in which when a vehicle makes an entrance request, the intersection manager calculates the optimal speed and sends it back to vehicle. Then, the vehicle executes the received command. Algorithms 1 and 2 show how the IM and vehicles collaborate with each other. Incoming vehicles send a request to the IM that includes two key pieces of information: the current velocity of the vehicle, V_C, and the distance to intersection, D_T. Another packet of information about the vehicle called $Vehicle_{Info}$ is sent that includes maximum acceleration, maximum deceleration, max speed, length, width, lane of entry, lane of exit, direction of entry, direction of exit, and safety buffer size. Additional information about the intersection, such as number of lanes in/out, directions of entry/exit, lane width, designated transmission line distance, and other intersection parameters are included in the packet IM_{Params} that is already known to the IM.

Although VT-IMs can provide the high throughput because of flexibility to adopt a variety of different scheduling algorithms, current VT-IM implementations do not consider computational delay caused by the IM and network delay imposed due to

if *a request is received* **then**

  V_T = calculateTargetVelocity(V_C, D_T, $Vehicle_{Info}$, IM_{Params});

  sendResponse(V_T);

end

Algorithm 1: Scheduling Algorithm - IM

if *designated line is crossed* **then**

  retransmit:

  sendRequest(V_C, D_T, $Vehicle_{Info}$) **while** *elapsed time ¡ timeout* **do**

    if *distance to intersection ¡= safe stop distance* **then**

      slow down to stop;

    end

    if *receive response* V_T **then**

      accelerate to V_T and maintain until exit;

      return;

    end

  end

  goto: *retransmit*;

end

Algorithm 2: Scheduling Algorithm - Vehicle

communication. Neglecting these delays affects the system correctness because the vehicle executes the received velocity command as soon as it is received. Figure 4.1 depicts how round trip delay (RTD) affects the position of the vehicle. In order to achieve safe operation, we must add extra RTD buffer around the vehicle to take into account the worst-case RTD.

The RTD consists of computation delay and transmission delay. Computational

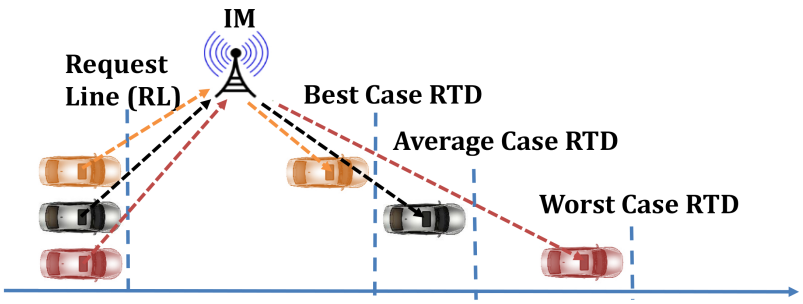

Figure 3.2: RTD Causes Late Command Delivery.

delay is the amount of time it takes for the intersection manager to compute the required information a vehicle needs. Compute time is longest when many vehicle requests are in the queue, therefore the worst-case for our four-way setup can be defined as four vehicle arrivals at the exact same time, one in each of the four directions. The resulting worst-case RTD from 10 tests with four vehicles arrivals was 135 milliseconds. The network delay is the time required to send the information back and forth between the vehicles and the IM, assuming the computation on IM is instant. In order to measure this delay, each request message can be followed by an acknowledge message from the receiver. Subtracting the time the message is sent, from the time the Ack is received, network delay for that message is accounted for. For our 2.4 GHz wireless devices, the worst measured network delay was 15 milliseconds. So, we have bounded RTD with 150 milliseconds for the sake of our experiments. At maximum speed, the $150ms$ delay would equate to an extra $0.45mm$ length being added to the vehicle. With the safety buffer and RTD buffer added in, our vehicles will be significantly longer longitudinally than they were originally.

In order to guarantee the safety of the vehicle, the *Worst-Case Computational Delay* (WC-CD) should be considered based on the worst-case scenario. However, because we cannot always bound the WC-RTD, vehicles are programmed with a retransmit clauses if no response is received from the IM within the WC-RTD timeout.

Chapter 4

TIMING PROBLEMS IN VT-IM

Velocity Transaction IMs or VT-IMs work in a manner in which when a vehicle makes an entrance request, the intersection manager calculates the optimal speed and sends it back to vehicle. Then, the vehicle executes the received command. Algorithms 1 and 2 show how the IM and vehicles collaborate with each other. Incoming vehicles send a request to the IM that includes two key pieces of information: the current velocity of the vehicle, V_C, and the distance to intersection, D_T. Another packet of information about the vehicle called $Vehicle_{Info}$ is sent that includes maximum acceleration, maximum deceleration, max speed, length, width, lane of entry, lane of exit, direction of entry, direction of exit, and safety buffer size. Additional information about the intersection, such as number of lanes in/out, directions of entry/exit, lane width, designated transmission line distance, and other intersection parameters are included in the packet IM_{Params} that is already known to the IM.

if *a request is received* **then**

    V_T = calculateTargetVelocity(V_C, D_T, $Vehicle_{Info}$, IM_{Params});

    sendResponse(V_T);

end

Algorithm 3: Scheduling Algorithm - IM

Although VT-IMs can provide the high throughput because of flexibility to adopt a variety of different scheduling algorithms, current VT-IM implementations do not consider computational delay caused by the IM and network delay imposed due to communication. Neglecting these delays affects the system correctness because the

```
if designated line is crossed then
    retransmit:
    sendRequest(V_C, D_T, Vehicle_Info) while elapsed time ¡ timeout do
        if distance to intersection ¡= safe stop distance then
            slow down to stop;
        end
        if receive response V_T then
            accelerate to V_T and maintain until exit;
            return;
        end
    end
    goto: retransmit;
end
```

Algorithm 4: Scheduling Algorithm — Vehicle

vehicle executes the received velocity command as soon as it is received. Figure 4.1 depicts how round trip delay (RTD) affects the position of the vehicle. In order to achieve safe operation, we must add extra RTD buffer around the vehicle to take into account the worst-case RTD.

The RTD consists of computation delay and transmission delay. Computational delay is the amount of time it takes for the intersection manager to compute the required information a vehicle needs. Compute time is longest when many vehicle requests are in the queue, therefore the worst-case for our four-way setup can be defined as four vehicle arrivals at the exact same time, one in each of the four directions. The resulting worst-case RTD from 10 tests with four vehicles arrivals was 135 milliseconds. The network delay is the time required to send the information back and

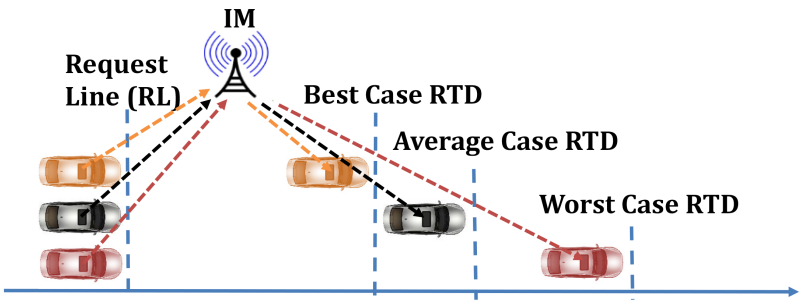

Figure 4.1: RTD Causes Late Command Delivery.

forth between the vehicles and the IM, assuming the computation on IM is instant. In order to measure this delay, each request message can be followed by an acknowledge message from the receiver. Subtracting the time the message is sent, from the time the Ack is received, network delay for that message is accounted for. For our 2.4 GHz wireless devices, the worst measured network delay was 15 milliseconds. So, we have bounded RTD with 150 milliseconds for the sake of our experiments. At maximum speed, the $150ms$ delay would equate to an extra $0.45mm$ length being added to the vehicle. With the safety buffer and RTD buffer added in, our vehicles will be significantly longer longitudinally than they were originally.

In order to guarantee the safety of the vehicle, the *Worst-Case Computational Delay* (WC-CD) should be considered based on the worst-case scenario. However, because we cannot always bound the WC-RTD, vehicles are programmed with a retransmit clauses if no response is received from the IM within the WC-RTD timeout.

Chapter 5

RELATED WORKS

5.1 Velocity Transaction Based IMs

Capitalizing on the optimization problem that intersection scheduling presents, there have been a number of works looking at the problem of how to schedule vehicles most efficiently using a velocity/acceleration profile based control methodology. In 2012, Lee and Park introduced a optimal methodology in order to schedule the incoming vehicles Lee and Park (2012). They constructed a conflict look-up table for vehicle entrance and exit lanes. The work is limited to simulation. Similarly, Zohdy, et al. solved an optimization problem to minimize the total delay. They proposed a tool which avoids collision based on characteristic of vehicles Zohdy *et al.* (2012). Unfortunately, neither of these methods consider the WC-RTD problem and its effect on the safety of their policies. In 2016, Tache et. al. proposed a batch scheduling technique that features a re-organization period where any vehicles that have reached the transmission line in a certain period of time can be shuffled around to find the most efficient order of entrance to the intersection Tachet *et al.* (2016). After the reshuffling period has elapsed for a given vehicle, the IM picks the vehicle velocity using a scheduling technique where the vehicle entrance time is set as the time the last vehicle occupying the designated lane has exited. The authors claim that the throughput can be doubled in comparison with fair scheduling. The authors implemented the technique for a two lane case in simulation. Computation time and network traffic overhead would be very high method because of the reordering, thereby increasing WC-RTD. However the authors do not model RTD, therefore the work could not be

applied to a physical system.

Some VT-IM methods implemented in a physical model, but at speeds too slow to experience timing and modeling issues. In Milanés *et al.* (2010), a fuzzy controller for a simple crossroad is presented. They evaluated their work by experiment on two mass-produced vehicles in Spain. Perronnet et all implemented a simple two way intersection utilizing Lego NXT robots and road marking to test IM protocols on a realistic model Ahmane *et al.* (2013).

5.2 Query-Based IMs

Dresner and Stone introduced AIM (Autonomous Intersection Management), a First Come First-Served (FCFS) IM policy that mitigates the effect of WC-RTD Dresner and Stone (2004), Dresner and Stone (2008), Dresner and Stone (2005), VanMiddlesworth *et al.* (2008). When an incoming vehicle reaches the designated line, it sends a request to the AIM IM indicating time of arrival T_{OA}, current velocity V_C, and $Vehicle_{Info}$ packet. The IM simulates the trajectory of the vehicle in the intersection and responds to the vehicle with an approval if the trajectory has no overlap with the trajectories of existing vehicles with reserved spots. If the request gets rejected, the vehicle will continue and re-request after a given interval.

AIM has a number of problems, the foremost being that AIM is limited to receiving vehicle requests to enter at a time determined by the requesters' current speed. This means that if the first request is denied, all subsequent requests will be as well until the vehicle slows down, and in many cases comes to a complete stop. Due to the tendency of the AIM IM to need to re-simulate the same vehicle trajectory multiple times before accepting, AIM has high compute and network load.

Dresner and Stone implemented an augmented reality simulation using their Java-based Autonomous Intersection Simulator, where virtual vehicles shared a four way

```
if request received then
    vehicleCollisions = simulateTrajectories(T_OA, V_C, Vehicle_Info);
    if vehicleCollisions == 0 then
        sendAccept();
    else
        sendReject();
    end
end
```

Algorithm 5: Query Algorithm - IM

stop intersection with an actual autonomous vehicles for the 2007 Darpa Challenge Dresner and Stone (2008). Fok, *et al.* built a scale model of autonomous vehicles Fok *et al.* (2012). Their vehicle systems prove it is possible to use the AIM autonomous intersection policy on a scale model. However, there are clear limitations: The use of only four vehicles (one vehicle per direction), and the slow speed of the vehicles ($0.5m/s$ @ 1/10 scale) in the intersection cause timing and modeling problems to be masked.

```
if designated line is crossed then
    retransmit:
    sendRequest(T_OA, V_C, Vehicle_Info);
    while time_elapsed ¡ timeout do
        if distance to intersection ¡= safe stop distance then
            brake to stop;
        end
        if response recieved then
            if response is accepted then
                enter at T_OA with velocity V_C;
                return;
            else
                break while loop;
            end
        end
    end
    reset(time_elapsed);
    goto: retransmit;
end
```

Algorithm 6: Query Algorithm — Vehicle

Chapter 6

OUR TIME-SENSITIVE TECHNIQUE

In order to cancel the effect of RTD in the real implementation, we treat WC-RTD as a delay in command execution. Figure 6.1 depicts three different scenarios where the vehicles start executing a velocity command at a designated execution time rather than the moment the command is received.

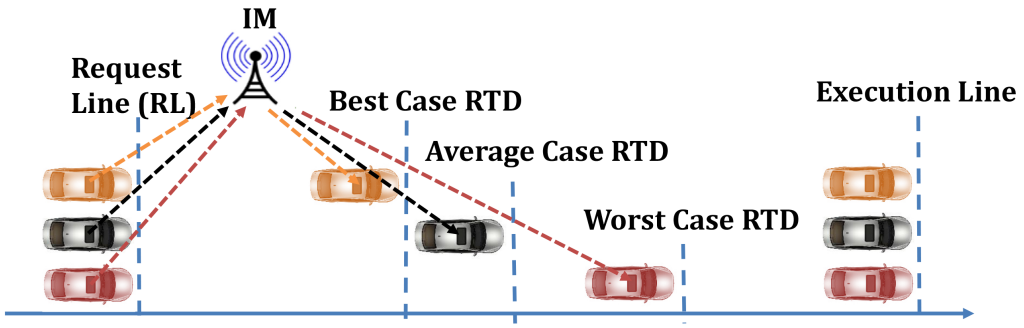

Figure 6.1: Vehicles Receiving Command From IM with Different Round Trip Delays.

Crossroads is based on a VT-IM. When a vehicle crosses the designated line, it sends a request message to IM containing the transmission timestamp, T_T, distance to intersection D_T, current velocity V_C, and the $Vehicle_{Info}$ packet. The IM calculates the desired Time of Arrival, T_{oA}, and, the execution time, T_E and sends it back to the vehicle. After receiving the message, the vehicle calculates a trajectory starting at time T_E and arriving at the intersection at time ToA with velocity V_T and follows that trajectory through the intersection. Pseudo-algorithms 5 and 6 depict how proposed technique works on the IM and vehicle, respectively.

Consider the example of a vehicle transmitting a request message at P_T and receiving the response at P_R (Figure 6.2). Then, the vehicle will start executing the

if *a request is received* **then**

T_E = calculateActuationTime(T_T,IM_{Params});

V_T, T_{oA} = calculateTargetArrivalTime(T_E, T_T, D_T, V_C, $Vehicle_{Info}$, IM_{Params});

sendResponse(T_E, T_{oA}, V_T);

end

Algorithm 7: Crossroads Algorithm — IM

if *designated line is crossed* **then**

T_T = getTime();

sendRequest(T_T, D_T, V_C, $Vehicle_{Info}$);

while *elapsed time ¡ timeout* **do**

if *distance to intersection ¡= safe stop distance* **then**

slow down to stop;

end

if *receive response* T_E, T_{oA}, *and* V_T **then**

actuate(T_E, T_{oA}, V_T);

return;

end

end

goto: *retransmit*;

end

Algorithm 8: Crossroads Algorithm — Vehicle

calculated trajectory at P_E.

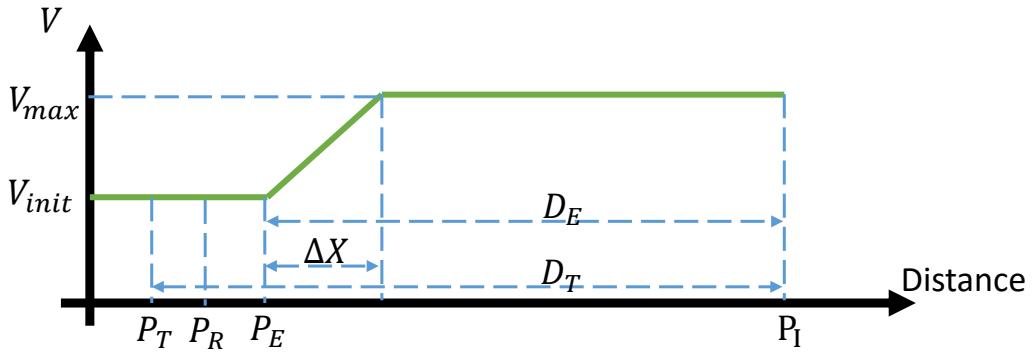

Figure 6.2: Example of a Vehicle Trajectory Based on Max Acceleration

IM computes the T_E as following: $T_E = T_T + WC - RTD$ where T_T is the time captured by the vehicle at transmit position. The vehicle should start executing the command exactly at time T_E. In our management technique, the IM checks the conflicts between current vehicle's trajectory and the trajectories of the existing ones. Then, a safe T_{oA} is calculated based on kinematic equation of vehicles and the earliest arrival time assigned to the last entered vehicle. The calculated T_{oA} may not be achievable for the vehicle depending of execution time, T_E, T_{oA}, maximum acceleration a_{max} and maximum deceleration d_{max}. Therefore, the IM checks the calculated T_{oA} based on the shortest acceleration time $T_{Acc} = \frac{V_{max} - V_{init}}{a_{max}}$ where V_{init} is initial speed of the vehicle. Then, earliest time of arrival can be calculated as $ET_{oA} = T_{Acc} + \frac{V_{max}}{D_E - \Delta X}$ where ΔX is the acceleration distance, $\Delta X = (0.5 a_{max} T_{Acc}^2 + V_{init} T_{Acc})$ and D_E is the the distance between intersection line and execution position $D_E = D_T - V_{init}(T_T - T_E)$ where D_T and T_T are distance to intersection and time respectively which are received from the vehicle.

Chapter 7

EMPIRICAL EVIDENCE

7.1 Time-sensitive Programming Enables Efficient IM

In order to evaluate the effect of the extra safety buffer, we designed 10 different traffic scenarios, and tried them with the two IMs, the VT-IM, which requires the extra safety buffers for safe operation, and Crossroads, which does not. Two of the cases, Scenario 1, and Scenario 10 are pre-designed, as the worst-case and best-case for VT-IM. In the best case, Scenario 10, the traffic is so sparse that the presence/absence of the safety buffer does not matter much. The cars can go cross the intersection with little conflict. On the contrary, in the worst-case, Scenario 1, all the cars arrive at the intersection at almost the same time, and the presence of extra safety buffers around the cars reduces the rate at which the cars can cross the intersection. In the rest of the cases, the vehicle orders and distances are randomly selected. We run all the traffic scenarios for each IM, and the delay is measured for all the cars. From here we compute the average delay of the cars. The experiment is repeated 10 times, and the average of that is plotted in figure 7.1.

The results show that for each scenario, Crossroads has lower average delay, ranging from 1.24X better for the worst-case, Scenario 1, to 1.08X better for Scenario 10. The slightly improved performance of Crossroads in Scenario 10 is because even in the case where vehicles are nicely spread out, there are still some Safety Buffer conflicts that cause the VT-IM policy to be slower.

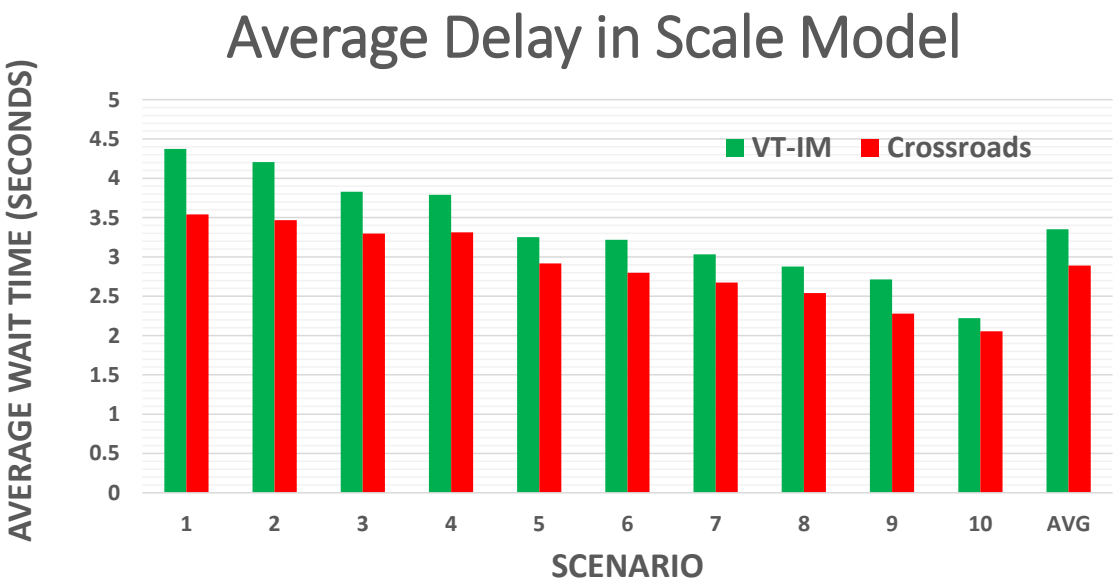

Figure 7.1: Average Wait Time Comparison for Vehicles in Four Different Cases of Our Physical Implementation.

7.2 Crossroad Scales Well

In order to show how our method scales, we implemented three simulators in Matlab for AIM, velocity-transaction IM with extra safety buffers and Crossroads. The IM code for TT-IM and Crossroads are exactly the same as those from our scale implementation. The major difference is in the modeling versus the physical scale model. In our Matlab simulators, the following differential equations are considered to model motion of the vehicles:

$$\begin{cases} \dot{x} = vcos(\phi) \\ \dot{y} = vsin(\phi) \\ \dot{\phi} = \frac{v}{l}tan(\psi) \end{cases} \tag{7.1}$$

where x, y represents the longitude and latitude of the car respectively in the Cartesian coordinates, ϕ is heading of the car from east, v is car velocity, l is car's

wheelbase and ψ is steering angle. The Matlab simulators are ran on an ordinary PC (Intel(R) Core(TM) i7-6700 @ 3.4GHz, 16 GB of memory and 64-bit Windows 10 Enterprise).

In our AIM simulator, we only considered sensor error buffer. In VT-IM, we considered both sensor error and WCRTD. Tn our time-sensitive approach, Crossroads, we only consider sensor error as WCRTD is already accounted for. We used the same input traffic flow and sequence of vehicle for all simulator to have a fair comparison. We considered the same velocity computational method for VT-IM and crossroads to emphasize on the effect of a larger buffer. Although the computation time of VT-IM and Crossroads is the same, AIM has up to 16x higher computation overhead. However, due to trial error scheme of AIM, it has 16x more computational delay than crossroads on average. Figure 7.2 also shows the throughput of the intersection for different flow rates routing 160 cars. In our computation, throughput is defined number of managed vehicles divided by total wait time.

Figure 7.2 reveals the throughput of Crossroads is 1.28x greater than AIM in worst case and 1.15x in average. Figure 7.2 also shows the throughput of Crossroad is 1.62x better than VT-IM in worst case and 1.36x in average. The results show that all three methods has the same throughput, however, AIM and VT-IM are saturated with increasing the input flow rate. VT-IM efficiency is better than AIM in low input flows (0.05-0.4 Car/Lane/Second) because at low flow rates, there are less conflict between the arriving vehicle. However, in higher flow rates (0.45 - 1.25 Car/Lane/Second), AIM can handle the traffic in a wise manner since the VT-IM has a larger buffer than AIM. The results from Matlab simulator show Crossroads has better throughput in comparison with a VT-IM policy because in higher input flow rates, Crossroads performs even better. This is mainly due to the effect of extra buffer which saturates the intersection earlier.

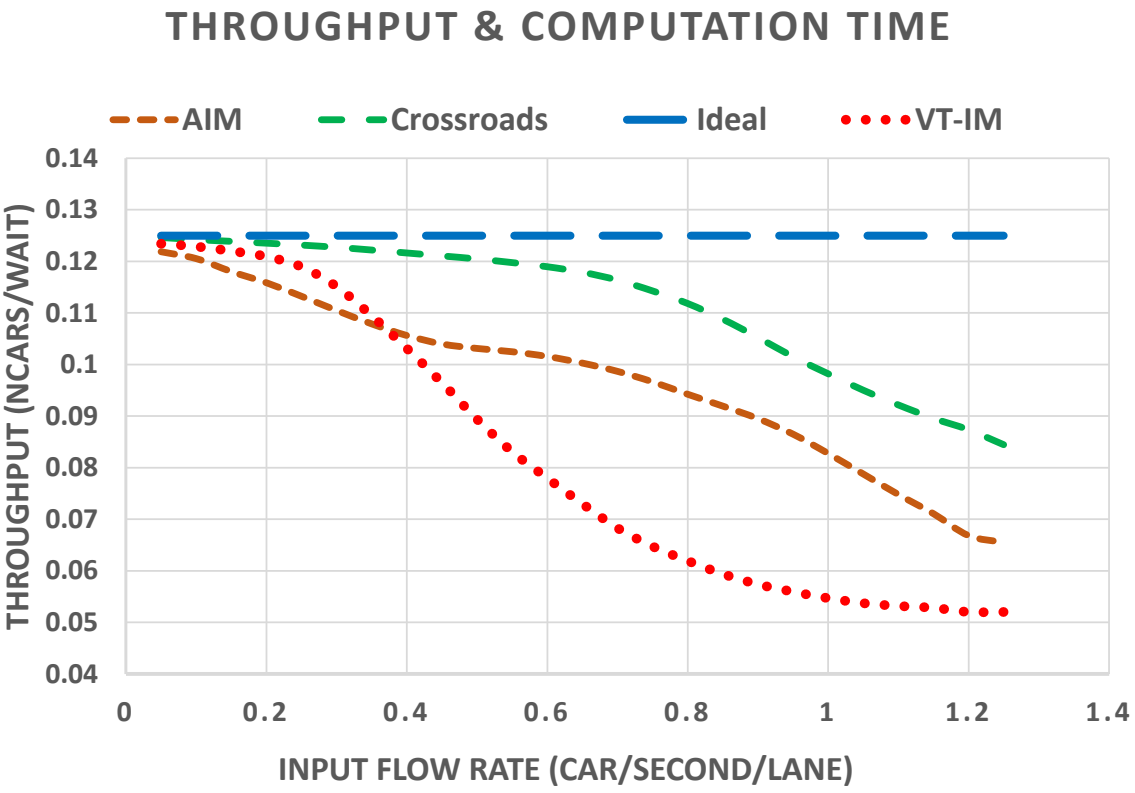

Figure 7.2: Throughput and Computation Time for Different Input Flow Rates

Chapter 8

CONCLUSION

In this paper, a time-sensitive technique, Crossroads, is proposed in order to eliminate the effect of network and computational delay in an automated intersection. The effectiveness of Crossroads is evaluated by conducting experiments on 1/10 scale autonomous vehicles as well as simulation. Crossroads improves the throughput and removes the safety-jeopardizing effects of both computation and network delay on vehicles in our simulated and modeled automated intersections.

REFERENCES

Ahmane, M., A. Abbas-Turki, F. Perronnet, J. Wu, A. El Moudni, J. Buisson and R. Zeo, "Modeling and controlling an isolated urban intersection based on cooperative vehicles", Transportation Research Part C: Emerging Technologies **28**, 44–62 (2013).

Dresner, K. and P. Stone, "Multiagent traffic management: A reservation-based intersection control mechanism", in "Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2", pp. 530–537 (IEEE Computer Society, 2004).

Dresner, K. and P. Stone, "Multiagent traffic management: An improved intersection control mechanism", in "Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems", pp. 471–477 (ACM, 2005).

Dresner, K. and P. Stone, "A multiagent approach to autonomous intersection management", Journal of artificial intelligence research **31**, 591–656 (2008).

Fok, C.-L., M. Hanna, S. Gee, T.-C. Au, P. Stone, C. Julien and S. Vishwanath, "A platform for evaluating autonomous intersection management policies", in "Cyber-Physical Systems (ICCPS), 2012 IEEE/ACM Third International Conference on", pp. 87–96 (IEEE, 2012).

Lee, J. and B. Park, "Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicles environment", IEEE Transactions on Intelligent Transportation Systems **13**, 1, 81–90 (2012).

Milanés, V., J. P. Rastelli, E. Onieva and C. González, "Controller for urban intersections based on wireless communications and fuzzy logic", IEEE Transactions on Intelligent Transportation Systems (2010).

Mills, D. L., "Internet time synchronization: the network time protocol", IEEE Transactions on Communications (1991).

Tachet, R., P. Santi, S. Sobolevsky, L. I. Reyes-Castro, E. Frazzoli, D. Helbing and C. Ratti, "Revisiting street intersections using slot-based systems", PloS one **11**, 3, e0149607 (2016).

VanMiddlesworth, M., K. Dresner and P. Stone, "Replacing the stop sign: Unmanaged intersection control for autonomous vehicles", in "Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3", pp. 1413–1416 (International Foundation for Autonomous Agents and Multiagent Systems, 2008).

Zohdy, I. H., R. K. Kamalanathsharma and H. Rakha, "Intersection management for autonomous vehicles using icacc", in "2012 15th International IEEE Conference on Intelligent Transportation Systems", pp. 1109–1114 (IEEE, 2012).