Delay-Aware Design, Analysis and Verification of Intelligent Intersection Management

(Invited Paper)

Bowen Zheng*, Chung-Wei Lin†, Hengyi Liang*, Shinichi Shiraishi‡, Wenchao Li‡, Qi Zhu*

*University of California, Riverside, CA 92521, Email: {bzheng, hliang, qzhu}@ece.ucr.edu
†TOYOTA InfoTechnology Center, Mountain View, CA 94043, Email: {cwlin, sshiraishi}@us.toyota-itc.com
‡Boston University, Boston, MA 02215, Email: wenchao@bu.edu

Abstract—With the rapid advancement of autonomous driving and vehicular communication technology, intelligent intersection management has shown great promise in improving transportation efficiency. In a typical intelligent intersection, an intersection manager communicates with autonomous vehicles wirelessly and schedules their crossing of the intersection. Previous system designs, however, do not address the possible communication delays due to network congestion or security attacks, and could lead to unsafe or deadlocked systems. In this work, we propose a delay-tolerant protocol for intelligent intersection management, and develop a modeling, simulation and verification framework for analyzing the protocol’s safety, liveness and performance. Experiments demonstrate the advantages of our proposed protocol over traditional traffic light control, and more importantly, demonstrate the importance and effectiveness of using this framework to address timing (delay) in vehicular network applications. This work is the first step towards a comprehensive delay-aware design and verification framework for practical vehicular network applications.

I. INTRODUCTION

In transportation systems, intersections are critical as they are associated with a significant percentage of traffic accidents and play an important role in traffic efficiency. According to the Fatality Analysis Reporting System (FARS) in the United States, 40% of crashes and 21.5% of fatal traffic accidents are associated with a significant percentage of traffic accidents [1], [2]. While traditional traffic lights have helped us manage intersections, they do not adapt well to real-time traffic and face challenges in efficiency [1].

With the rapid advancement of autonomous driving and vehicular communication technology, intelligent intersection management techniques have shown great promise in improving intersection safety and transportation efficiency. In an intelligent intersection, autonomous vehicles with vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication capabilities will exchange information of current driving states with each other or with roadside controllers for coordinated crossing of the intersection. In the United States, the advancement of vehicular communication technology has led to the development of Dedicated Short Range Communication (DSRC) standard [3], [4], and sets the foundation for V2V and V2I applications like intelligent intersection management.

In the literature, intelligent intersection management techniques can be classified into two major categories: centralized management and distributed management. Centralized intersection management utilizes V2I communications, where every vehicle communicates with a central intersection manager for permission to cross the intersection [5], [6], [7], [8], [9], [10]. These works typically divide an intersection into grids, and formulate the problem as assigning grids to different vehicles at each time step. In [5], the proposed protocol is extended to combine traffic lights and V2I communication for both autonomous and regular vehicles. The work in [6] studies fuel consumption and vehicle emission compared with traditional traffic lights. The work in [7] uses control theories to prove system safety and liveness through hybrid architectures. In [8], [9], the authors define a spatio-temporal protocol that can handled both four-way intersections and roundabouts by assigning grids to vehicles at each time step, and prove their protocol is deadlock free when there is no communication delay. In [10], the problem is abstracted as traffic flows with conflict points and formulated as a linear programing problem. Distributed intersection management requires the vehicles to negotiate the right-of-way among themselves before entering the intersection [11], [12], [13]. In [13], the system can be proven to be deadlock-free using Petri Net models. In [11], Timed Petri Nets models are again used to decide the sequence of vehicles entering intersection for traffic smoothness. In all these previous works, vehicular communications are assumed as instantaneous (or with a short constant delay) and always reliable.

However, the wireless nature of vehicular communications makes it susceptible to significant communication delays [14] and packet losses [15] in dense traffic scenarios or under security attacks [16]. Previous works on intelligent intersection management lack the consideration of communication message delays and losses, and consequently cannot ensure the proposed protocols to be safe, deadlock free and efficient in practical conditions.

In this work, we propose a delay-tolerant centralized intersection management protocol, which takes into account the possible communication delays and losses between vehicles and the central intersection manager. We develop a modeling, simulation and verification framework for analyzing the safety, liveness and performance of the proposed protocol, as shown in Fig. 1. To the best of our knowledge, both the delay-tolerant protocol and the analysis framework are the first to quantitatively and formally address the intersection management problem with delay consideration.
Overall, the main contributions of our work include:

- We develop a delay-tolerant protocol for intelligent intersection management. The protocol assures that as long as the communication delays are bounded, every vehicle will eventually cross the intersection and vehicles with conflicting routes will never enter the intersection at the same time.

- We model and implement our protocol in the SUMO traffic simulation suite [17], with the extension of modeling communication delays.

- We verify the safety and liveness properties of our protocol by building more abstract timed automata models and leveraging the UPPAAL environment [18].

The rest of the paper is organized as follows. Section II introduces the system model for intersection management. Section III presents the proposed delay-tolerant intersection management protocol. Section IV presents the building of more abstract timed automata models and the usage of UPPAAL for verifying the protocol safety and liveness. Section V presents the simulation with SUMO extension together with the verification and simulation results. Section VI concludes the paper.

II. Basic System Model

The basic intersection management system model is illustrated in Fig. 2. In this system, a central Intersection Manager communicates with every vehicle via V2I communication channels to schedule the traffic crossing the intersection. A basic version of the protocol is as follows (a more formal and detailed description with delay consideration is presented in Section III):

- Vehicle: 1) sends a Request message to the intersection manager, 2) enters the intersection only after it receives a Confirm message from the manager, otherwise stops before the intersection, and 3) resends a Request message when Confirm is not received within a pre-defined timeout bound.

- Intersection Manager: 1) receives Request messages from vehicles, and 2) schedules vehicles to enter the intersection based on a scheduling policy, e.g., first come, first served (FCFS).

As stated before, this work explicitly considers communication delays and losses between Intersection Manager and vehicles, as shown in Fig. 2. The goal of this work is to design a delay-tolerant protocol that can improve intersection performance/efficiency (measured by average traveling time for vehicles to cross the intersection) and satisfy the following properties:

- Safety: vehicles with conflicting routes (i.e., routes that may cross each other within the intersection) may never enter the intersection at the same time.²

- Liveness: every vehicle that sends request will eventually cross the intersection, as long as the communication delays are bounded by a timeout bound.

To guarantee the two properties and provide high performance, we assume that the Intersection Manager is capable of detecting whether the vehicles have entered or left the intersection, which can be provided through sensors such as cameras, traffic loop detectors, etc. We assume all vehicles are autonomous and can detect whether there is any vehicle between its current location and the intersection.

²It should be noted that the vehicles are assumed to have autonomous driving capabilities and may detect or even avoid incoming collisions in many cases. Nevertheless, conflicting routes could still lead to unsafe situations given the limitations of autonomous driving, and are likely to cause deadlocks even without accidents.
the question is how long the intersection manager should wait before it can safely confirm another vehicle V2.

The third issue is shown in Fig. 3 (c) where a vehicle sends Request to the intersection manager but gets no response from the intersection manager. The question is when the vehicle should resend the Request message to avoid possible deadlock.

B. Delay-tolerant Intersection Management Protocol

Timeouts: To address the issues caused by communication delays, we introduce three types of timeouts in our protocol: 1) timeout for each message transmission, denoted as $T^{\text{out}}$, 2) timeout for a vehicle to wait before resending the request, denoted as $T^{\text{out}}$, and 3) timeout for the Intersection Manager to wait for a vehicle to enter the intersection, denoted as $T^{\text{out}}$. More specifically, $T^{\text{out}}$ represents the living period of that message, i.e., the message becomes invalid and should not be used after the timeout. $T^{\text{out}}$ represents how long a vehicle should wait, when no Confirm is received, before resending the request. $T^{\text{out}}$ represents how long the Intersection Manager should wait for the currently scheduled vehicle to enter the intersection, before it schedules another vehicle.

Messages: Three types of messages are defined in our protocol for communication between the vehicle and the Intersection Manager, as shown below.

- **Request.** A request message is sent by a vehicle to acquire permission for entering the intersection. It contains requestID, sender, sending time, timeout ($T^{\text{out}}$), and estimated arriving time ($t_{\text{exp}}$). In particular, the estimated arriving time is used by the Intersection Manager to schedule the time for each vehicle to enter the intersection. As we assume the vehicles are autonomous, the estimated arriving time can be calculated using the location, speed and acceleration information collected from their sensors.
- **Confirm.** A confirm message is sent by the Intersection Manager to give permission to a vehicle for entering the intersection. It contains confirmID, sending time, timeout ($T^{\text{out}}$), and arriving time range ($[T^L, T^H]$). If the vehicle enters the intersection during the arriving time range, it is guaranteed to be safe according to our protocol. A vehicle cannot enter the intersection if no Confirm is received. If the vehicle cannot enter the intersection within the time range, it must not enter the intersection, either; instead, the vehicle can send a cancel message as discussed below.
- **Cancel.** A cancel message is sent by a vehicle to notify the Intersection Manager that a previous Confirm is “cancelled” by the vehicle and it will not enter the intersection. The Cancel message is used for improving the performance and is in fact optional. Once receiving the Cancel message, the Intersection Manager can schedule other vehicles immediately and does not need to wait for the vehicle to cross the intersection. Without receiving the Cancel message, the Intersection Manager will wait for the timeout $T^{\text{out}}$ before scheduling another vehicle (note that the Intersection Manager knows whether the vehicle enters the intersection through sensors). The fields in a Cancel message include

![Diagram](image-url)

(a) Message living period.

![Diagram](image-url)

(b) When to safely schedule another vehicle.

![Diagram](image-url)

(c) When to resend request.
cancelID, corresponding confirmID, sending time, and timeout $T_{out}$.

Based on the above definitions, our protocol is described by state machines in below.

**State Machine for Vehicles:** The state machine for a vehicle is shown in Fig. 4. In the state machine, there are two variables for denoting time. Variable $t_1$ denotes the local time for each state and variable $t_{exp}$ denotes the expected arriving time in global time. There are five states for the vehicle: approaching not confirmed, decelerating not confirmed, approaching confirmed, entering intersection and left intersection. The details of each state and the transitions are described below.

**Approaching not Confirmed:** This is the starting state for every vehicle approaching an intersection. In this state, once the vehicle becomes the front vehicle (i.e., there is no other vehicle between it and the intersection), it sends a Request message and waits for the corresponding Confirm message from the intersection manager. Inside the request message, the field $t_{exp}$ includes estimated arriving time to the intersection based on current vehicle location, speed and acceleration. The vehicle will resend the Request if no Confirm is received within the timeout bound $T_{out}$. As the vehicle is approaching the intersection, if no Confirm is receive, it may need to decelerate and stop before the intersection waiting line. In our case, if the distance to the intersection $d$ (or to the last vehicle waiting at the intersection) is less than a safe value $L$, the vehicle will enter the state Decelerating not Confirmed. If a Confirm is received within the timeout bound and before decelerating, the vehicle will directly enter the state of Approaching Confirmed.

**Decelerating not Confirmed:** In this state, the vehicle decelerates and ensures that it can fully stop before the waiting line of the intersection. The vehicle will send a Request if it becomes the front vehicle, and if no Confirm is received within the timeout bound $T_{out}$, it will resend the Request. The field $t_{exp}$ in the request message will be based on the new location, speed and deceleration information. If Confirm is received within $T_{out}$ at this state, the vehicle will enter the Approaching Confirmed state.

**Approaching Confirmed:** In this state, the vehicle has received Confirm from the Intersection Manager with a time range $[T_L, T_H]$ assigned for it to enter the intersection. The vehicle will continuously check whether it can arrive at the intersection within the assigned time range. If the vehicle finds it cannot enter the intersection in time, it will send a Cancel message to notify the Intersection Manager and switch back to the one of the states waiting for the Confirm message: If the distance to the intersection is still larger than the safe value ($d \geq L$), the vehicle will switch back to the Approaching not Confirmed state; otherwise to the Decelerating not Confirmed state. If the vehicle can arrive at the intersection within time range $[T_L, T_H]$, it will enter the intersection and switch to the state Entering Intersection. It should be noted that our protocol will still function safely (but less efficient) if there is no Cancel message (or it is lost or delayed too long), since the Intersection Manager can sense whether the vehicle has entered the intersection and will schedule another vehicle after timeout $T_{out}$.

**Entering Intersection:** In this state, the vehicle enters the intersection with a preset speed. Once the vehicle has left the intersection, it will enter the Left Intersection state. As we assume the Intersection Manager can sense whether the vehicle has entered or left the intersection, no action is needed for a vehicle in this state.

**Left Intersection:** In this state, the vehicle has left the intersection. No action is needed as explained above.

**State Machine for Intersection Manager:**

The state machine for the Intersection Manager is shown in Fig. 5. There are three states: Idle, confirm sent vehicle not cross and confirm sent vehicle cross. Before discussing the details of each state, we first introduce the routine that handles the messages received from the vehicles, i.e., the Request and Cancel messages. All the messages received will be put into a buffer, and the messages exceeding timeout will be deleted. The message handling routine is activated during all states. The Idle state is the one that the Intersection Manager schedules vehicles. In our current implementation, we adopt the First Come First Served (FCFS) scheduling policy, and this can be easily changed to other policies. In FCFS, the Intersection Manager will first schedule the request from the vehicle that 1) has no other vehicle between it and the intersection (this is in fact guaranteed as only the front vehicle can send request in current model), and 2) has an estimated arriving time $t_{exp}$ that is the smallest among all front vehicles.

**Idle:** In this state, the intersection manager checks whether the buffer storing messages from vehicles is empty. If it is not empty, it will select a Request (hence a vehicle) based on the scheduling policy. If the route of the selected vehicle conflicts with the routes of the vehicles currently inside the intersection, the Intersection Manager will wait for the current vehicles inside the intersection to finish crossing before it sends the Confirm message to the selected vehicle; otherwise it sends the Confirm message immediately. The Intersection Manager will also assign the time range for the selected vehicle to enter.
In our current implementation, the Intersection Manager will first compare current time with the expected arrival time \( t_{exp} \) from the Request. If \( currentTime \geq t_{exp} \), the upper bound for the time range \( T_H \) is set to \( T_H = currentTime + T_{out} \); otherwise it is set to \( T_H = t_{exp} + T_{out} \). The lower bound is set as \( currentTime \). After sending the Confirm message, the Intersection Manager will enter the state of Confirm Sent Vehicle not Cross.

**Confirm Sent Vehicle not Cross:** As stated before, we assume the Intersection Manager can sense whether the selected vehicle has entered the intersection. In this state, if a Cancel message is received, the Intersection Manager will enter the Idle state immediately. If the Intersection Manager senses the vehicle has entered the intersection within assigned time range, it will enter the Confirm Sent Vehicle Cross state; otherwise, it will enter the Idle state and schedule another vehicle.

**Confirm Sent Vehicle Cross:** In this state, the Intersection Manager sensed the current vehicle had entered the intersection and should switch to the Idle state immediately.

**Issues Revisit:** Given the protocol above, we revisit the three examples discussed at the beginning of this section. For the first problem, every message has a living period \( T_{out} \), so an outdated message will never be used. For the second issue, once a confirmation is sent to the vehicle, the intersection is currently reserved for it. The intersection manager will not schedule another vehicle until \( T_{out}' \). The upper bound for the time range \( T_H \) is set to \( T_H = currentTime + T_{out} \); otherwise it is set to \( T_H = t_{exp} + T_{out} \). The lower bound is set as \( currentTime \). After sending the Confirm message, the Intersection Manager will enter the state of Confirm Sent Vehicle not Cross.

**Confirm Sent Vehicle not Cross:** As stated before, we assume the Intersection Manager can sense whether the selected vehicle has entered the intersection. In this state, if a Cancel message is received, the Intersection Manager will enter the Idle state immediately. If the Intersection Manager senses the vehicle has entered the intersection within assigned time range, it will enter the Confirm Sent Vehicle Cross state; otherwise, it will enter the Idle state and schedule another vehicle.

**Confirm Sent Vehicle Cross:** In this state, the Intersection Manager sensed the current vehicle had entered the intersection and should switch to the Idle state immediately.

**Issues Revisit:** Given the protocol above, we revisit the three examples discussed at the beginning of this section. For the first problem, every message has a living period \( T_{out} \), so an outdated message will never be used. For the second issue, once a confirmation is sent to the vehicle, the intersection is currently reserved for it. The intersection manager will not schedule another vehicle until \( T_{out}' \). The upper bound for the time range \( T_H \) is set to \( T_H = currentTime + T_{out} \); otherwise it is set to \( T_H = t_{exp} + T_{out} \). The lower bound is set as \( currentTime \). After sending the Confirm message, the Intersection Manager will enter the state of Confirm Sent Vehicle not Cross.

**Confirm Sent Vehicle not Cross:** As stated before, we assume the Intersection Manager can sense whether the selected vehicle has entered the intersection. In this state, if a Cancel message is received, the Intersection Manager will enter the Idle state immediately. If the Intersection Manager senses the vehicle has entered the intersection within assigned time range, it will enter the Confirm Sent Vehicle Cross state; otherwise, it will enter the Idle state and schedule another vehicle.

**Confirm Sent Vehicle Cross:** In this state, the Intersection Manager sensed the current vehicle had entered the intersection and should switch to the Idle state immediately.

**Issues Revisit:** Given the protocol above, we revisit the three examples discussed at the beginning of this section. For the first problem, every message has a living period \( T_{out} \), so an outdated message will never be used. For the second issue, once a confirmation is sent to the vehicle, the intersection is currently reserved for it. The intersection manager will not schedule another vehicle until \( T_{out}' \). The upper bound for the time range \( T_H \) is set to \( T_H = currentTime + T_{out} \); otherwise it is set to \( T_H = t_{exp} + T_{out} \). The lower bound is set as \( currentTime \). After sending the Confirm message, the Intersection Manager will enter the state of Confirm Sent Vehicle not Cross.

**Confirm Sent Vehicle not Cross:** As stated before, we assume the Intersection Manager can sense whether the selected vehicle has entered the intersection. In this state, if a Cancel message is received, the Intersection Manager will enter the Idle state immediately. If the Intersection Manager senses the vehicle has entered the intersection within assigned time range, it will enter the Confirm Sent Vehicle Cross state; otherwise, it will enter the Idle state and schedule another vehicle.

**Confirm Sent Vehicle Cross:** In this state, the Intersection Manager sensed the current vehicle had entered the intersection and should switch to the Idle state immediately.

**Issues Revisit:** Given the protocol above, we revisit the three examples discussed at the beginning of this section. For the first problem, every message has a living period \( T_{out} \), so an outdated message will never be used. For the second issue, once a confirmation is sent to the vehicle, the intersection is currently reserved for it. The intersection manager will not schedule another vehicle until \( T_{out}' \). The upper bound for the time range \( T_H \) is set to \( T_H = currentTime + T_{out} \); otherwise it is set to \( T_H = t_{exp} + T_{out} \). The lower bound is set as \( currentTime \). After sending the Confirm message, the Intersection Manager will enter the state of Confirm Sent Vehicle not Cross.

**Confirm Sent Vehicle not Cross:** As stated before, we assume the Intersection Manager can sense whether the selected vehicle has entered the intersection. In this state, if a Cancel message is received, the Intersection Manager will enter the Idle state immediately. If the Intersection Manager senses the vehicle has entered the intersection within assigned time range, it will enter the Confirm Sent Vehicle Cross state; otherwise, it will enter the Idle state and schedule another vehicle.

**Confirm Sent Vehicle Cross:** In this state, the Intersection Manager sensed the current vehicle had entered the intersection and should switch to the Idle state immediately.

**Issues Revisit:** Given the protocol above, we revisit the three examples discussed at the beginning of this section. For the first problem, every message has a living period \( T_{out} \), so an outdated message will never be used. For the second issue, once a confirmation is sent to the vehicle, the intersection is currently reserved for it. The intersection manager will not schedule another vehicle until \( T_{out}' \). The upper bound for the time range \( T_H \) is set to \( T_H = currentTime + T_{out} \); otherwise it is set to \( T_H = t_{exp} + T_{out} \). The lower bound is set as \( currentTime \). After sending the Confirm message, the Intersection Manager will enter the state of Confirm Sent Vehicle not Cross.

**Confirm Sent Vehicle not Cross:** As stated before, we assume the Intersection Manager can sense whether the selected vehicle has entered the intersection. In this state, if a Cancel message is received, the Intersection Manager will enter the Idle state immediately. If the Intersection Manager senses the vehicle has entered the intersection within assigned time range, it will enter the Confirm Sent Vehicle Cross state; otherwise, it will enter the Idle state and schedule another vehicle.

**Confirm Sent Vehicle Cross:** In this state, the Intersection Manager sensed the current vehicle had entered the intersection and should switch to the Idle state immediately.

**Issues Revisit:** Given the protocol above, we revisit the three examples discussed at the beginning of this section. For the first problem, every message has a living period \( T_{out} \), so an outdated message will never be used. For the second issue, once a confirmation is sent to the vehicle, the intersection is currently reserved for it. The intersection manager will not schedule another vehicle until \( T_{out}' \). The upper bound for the time range \( T_H \) is set to \( T_H = currentTime + T_{out} \); otherwise it is set to \( T_H = t_{exp} + T_{out} \). The lower bound is set as \( currentTime \). After sending the Confirm message, the Intersection Manager will enter the state of Confirm Sent Vehicle not Cross.

**Confirm Sent Vehicle not Cross:** As stated before, we assume the Intersection Manager can sense whether the selected vehicle has entered the intersection. In this state, if a Cancel message is received, the Intersection Manager will enter the Idle state immediately. If the Intersection Manager senses the vehicle has entered the intersection within assigned time range, it will enter the Confirm Sent Vehicle Cross state; otherwise, it will enter the Idle state and schedule another vehicle.

**Confirm Sent Vehicle Cross:** In this state, the Intersection Manager sensed the current vehicle had entered the intersection and should switch to the Idle state immediately.

**Issues Revisit:** Given the protocol above, we revisit the three examples discussed at the beginning of this section. For the first problem, every message has a living period \( T_{out} \), so an outdated message will never be used. For the second issue, once a confirmation is sent to the vehicle, the intersection is currently reserved for it. The intersection manager will not schedule another vehicle until \( T_{out}' \). The upper bound for the time range \( T_H \) is set to \( T_H = currentTime + T_{out} \); otherwise it is set to \( T_H = t_{exp} + T_{out} \). The lower bound is set as \( currentTime \). After sending the Confirm message, the Intersection Manager will enter the state of Confirm Sent Vehicle not Cross.

**Confirm Sent Vehicle not Cross:** As stated before, we assume the Intersection Manager can sense whether the selected vehicle has entered the intersection. In this state, if a Cancel message is received, the Intersection Manager will enter the Idle state immediately. If the Intersection Manager senses the vehicle has entered the intersection within assigned time range, it will enter the Confirm Sent Vehicle Cross state; otherwise, it will enter the Idle state and schedule another vehicle.

**Confirm Sent Vehicle Cross:** In this state, the Intersection Manager sensed the current vehicle had entered the intersection and should switch to the Idle state immediately.

**Issues Revisit:** Given the protocol above, we revisit the three examples discussed at the beginning of this section. For the first problem, every message has a living period \( T_{out} \), so an outdated message will never be used. For the second issue, once a confirmation is sent to the vehicle, the intersection is currently reserved for it. The intersection manager will not schedule another vehicle until \( T_{out}' \). The upper bound for the time range \( T_H \) is set to \( T_H = currentTime + T_{out} \); otherwise it is set to \( T_H = t_{exp} + T_{out} \). The lower bound is set as \( currentTime \). After sending the Confirm message, the Intersection Manager will enter the state of Confirm Sent Vehicle not Cross.
where all the Cancel messages are lost. The In-Channel automata is associated with the corresponding “id”s of corresponding directions. The automata can sense the trigger of the synchronizer “request[id]” and move to the GetRequest state, which represents the sending of the message from the vehicle. The automata will then wait for the trigger of another synchronizer “request2[id]”, which represents the receiving of the message at the Intersection Manager. Such transition is bounded by a timeout. The Out-Channel is similarly modeled.

Finally, the automata representing the Intersection Manager is shown in Fig. 6 (b). We first implement a queue to store the request from the vehicles, with functions as enqueue() and dequeue(). The queue is first-in-first-out, and the new request will overwrite the old request from the same direction. The enqueue() routine runs on all states. Once the queue becomes non-empty, the Intersection Manager will select a request from the buffer with an “id” number. The following scheduling is similar to the state machine case.

V. Experiment Results

A. Verification Results

Using the timed automata models from Section IV, we have successfully proved the following properties in UPPAAL:

- **A[] not deadlock imply delay <= T_m**. The message delay must be smaller than the message timeout T_m to ensure that the system does not deadlock. We have observed counter examples where delays longer than T_m caused deadlocks (similarly for the next two properties).

- **A[] not deadlock imply T_out >= 2 * T_m**. The timeout for resending the request must be at least two times larger than the timeout of the message to ensure the system does not deadlock.

- **T_out >= T_H and Vehicle(i).requestSent → Vehicle(i).EnteringIntersection**. When the first two properties are guaranteed by setting the proper timeout bounds, and the time the Intersection Manager should wait for the currently scheduled vehicle to enter the intersection T_out is greater than the upper bound of the time range assigned to the corresponding vehicle T_H, this liveness property is proved. That is, once the vehicle sends a request, it will eventually cross the intersection.

- **A[] IntersectionV(0).InIntersection + IntersectionV(1).Intersection + IntersectionV(2).InIntersection + IntersectionV(3).InIntersection <= 1**. When the first three properties are guaranteed, this safety property is proved. That is, no vehicles from different directions can enter the intersection at the same time (note that this is a stronger condition than the safety properties discussed in Section II).

B. Simulation Results

Simulator Implementation: We implement our simulation environment based on the widely-used traffic simulator SUMO [17]. Specifically, we implement the state machines for the vehicles and the Intersection Manager, following the state machines defined in Section III. We control the movement of the vehicles by leveraging the TraCI API provided by the SUMO simulation suite. Most importantly, we added the explicit modeling of communication delays in SUMO. During
simulation, at each time step, we halt the SUMO engine and obtain the location, speed and acceleration information of vehicles for facilitating our protocol simulation. In this experiment we model a four-way single-lane intersection and vehicles are arriving based on Poisson distributions. The screenshot for simulation in SUMO and our visualization tool Unity is shown in Fig. 7.

**Delay-Tolerant Protocol vs. Traffic Lights:** We first compare the performance of our protocol with traditional traffic lights. The performance is evaluated as the average traveling time of each vehicle, i.e., the time difference of entering the intersection range and leaving the intersection range. The range is a radius of 50 meters from the intersection center. The arriving rate of the Poisson distribution is within a range of $[0, 0.5]$ (the unit is vehicles/second). We also introduce a factor $K$ to denote the ratio of traffic arriving rates from different directions. $K = 1$ represents that the north-south directions have the same traffic arriving rate as the east-west directions. $K = 2$ and $K = 3$ represent that the north-south direction has twice and three times of traffic arriving rate than the east-west direction, respectively. The traditional traffic light is set with a red light phase of 36 seconds, a green light phase of 31 seconds, and a yellow light phase of 5 seconds, which are the default values in SUMO. The timeout values in our protocol are set as $T^p_{out} = 4$, $T^w_{out} = 8$ and $T^w_{out} \geq T_H$ (all units in seconds).

The simulation results are shown in Fig. 8, where the x-axis denotes the total traffic arriving rate from all directions, and the y-axis denotes the ratio of the average traveling time between our protocol and the traffic lights (i.e., setting the traffic lights performance as baseline). Each data point is the average of 6 randomly generated traffic patterns following Poisson distribu-
bution. We can find out that our proposed protocol provides significantly better performance than the traditional traffic lights when the traffic is not too heavy or when the traffic arriving rates from different directions are asymmetric. When the traffic is heavy and symmetric from different directions, the traditional traffic lights achieve their best performance and can be better than our solution (although our solution can be further improved with more finer-granularity control as planned in the future work).

Impact of Communication Delays on Performance: We further evaluate the performance (average traveling time of each vehicle) under different communication delays, as shown in Fig. 9. We can see that the performance significantly decreases (longer traveling time) with the increase of communication delays, in particular when the traffic is heavy. This again demonstrates the importance of modeling and analyzing the impact of delays in intersection management, not only for the safety and liveness properties, but also for the system performance. It should be noted that in normal traffic conditions, the communication delays are typically under one second (in the range of dozens of milliseconds and can reach hundreds of milliseconds when considering end-to-end delays [14]). Under security attacks such as jamming, the delays can be much longer. Note that if we remove the delay consideration in the protocol, deadlocks are observed during simulation.

VI. CONCLUSION

This paper addresses the intelligent intersection management problem with quantitative analysis of communication delays. It presents a delay-tolerant intersection management protocol, and a framework for modeling, simulating and verifying the safety, liveness and performance of the proposed protocol. Experiments demonstrate the effectiveness of both the proposed protocol and the framework. Future work includes the consideration of more complex intersection models where the intersection can be divided into grids and scheduled in finer granularity. It will also include the extension of the framework to address other vehicular network applications.

ACKNOWLEDGEMENT

This work is supported in part by the Office of Naval Research grants N00014-14-1-0815 and N00014-14-1-0816, and the National Science Foundation grants CCF-1553757, CNS-1646641 and CCF-1646381.

REFERENCES