IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 3, 2020, accepted July 16, 2020, date of publication July 22, 2020, date of current version August 3, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3011195

Real-Time Counting of Vehicles Stopped at a Traffic Light Using Vehicular Network Technology

MANUEL CONTRERAS[1] AND ERIC GAMESS[2]
[1]School of Computer Science, Central University of Venezuela, Caracas 1040, Venezuela
[2]MCIS Department, Jacksonville State University, Jacksonville, AL 36265, USA

Corresponding author: Manuel Contreras (mcontre@ula.ve)

This work was supported in part by the Consejo de Desarrollo Científico y Humanístico-Central University of Venezuela (CDCH-UCV) under Grant PG 03-8066-2011/1.

ABSTRACT Vehicular Adhoc NETworks (VANETs) is a new and emerging technology for wireless communications that has attracted considerable attention in the last years from the academic, scientific, industrial and governmental communities, due to the improvements and the new features that it brings to the Intelligent Transportation Systems (ITSs). In this paper, we present an algorithm that uses VANET technology to determine the total number of vehicles that are stopped at a traffic light at an intersection. To facilitate an efficient counting, we divide the road into fixed-size road regions and through a leader election mechanism, we designate one vehicle in each region as the ''Region Leader'' that is in charge of computing and propagating the total number of nodes in its region. Additionally, the region leader will act as a router to retransmit counting information from other region leaders that are further away, so it can eventually reach the central processing point where it is processed. We have carried out extensive experiments in various scenarios to validate and analyze the behavior and precision of our new proposed algorithm. Also, we compare our algorithm with another strategy proposed by a research team led by Alok Rajan. Our simulations were done using Veins (Vehicle in Network Simulation), an open-source simulation framework that ties together two simulators: OMNeT++ for the network simulation and SUMO for the microscopic road traffic simulation. Both simulators run in parallel and communicate with each other through a TCP connection using a protocol called TraCI (Traffic Control Interface). Our simulation results show that the algorithm performs an effective counting of vehicles, with a reduced response time, and a small total number of control messages sent by the vehicles to accomplish the counting task, under different conditions of vehicular traffic loads.

INDEX TERMS Intelligent transportation systems, OMNeT++, SUMO, TraCI, VANET, vehicle counting, vehicular networks, Veins.

I. INTRODUCTION

Using the road transportation is almost inevitable in modern-day life. However, in recent years, we have seen the apparition of new problems due to the growth of vehicular traffic that include: traffic accidents, traffic congestion, air pollution, noise contamination, and others [1]. For these reasons, funding and research that aim to mitigate all these increasing phenomena are becoming more noticeable.

One important step to improve the ITSs (Intelligent Transportation Systems) of today is to make the vehicles and road side infrastructure more ''intelligent'', by allowing communications between each other. This new ability will help in finding new solutions to current problems [2]. Advances in this direction are aimed to reduce the number of accidents, facilitate the task of driving, optimize the traffic flow, minimize the contamination, and provide greater comfort and entertainment to passengers.

To meet these needs, information and communication technologies are experiencing strong development in the context of the vehicular environment. In recent times, WAVE (Wireless Access in Vehicular Environment) has been added to the possibilities of connectivity in VANETs (Vehicular Adhoc NETworks) and is getting growing attention from both the academic and the industrial communities [3]. The goal of a vehicular network is to allow communications between vehicles and between vehicles and the network infrastructures that are deployed on the side of roads. Thanks to these new possibilities of inter-vehicular communications, many novel applications have been proposed that we can classify is three

The associate editor coordinating the review of this manuscript and approving it for publication was Yan Huo.

groups: (1) safety, (2) traffic optimization, and (3) comfort and entertainment. To implement these applications, new basic algorithms are needed, for example, the counting of vehicles in different scenarios. In fact, counting vehicles is required in most of the vehicular applications, such as the monitoring, control, and optimization of traffic, the detection of traffic congestion, the computation of routes for vehicles, the development of Adaptive Traffic-light Control Systems (ATCSs), and the automation of parking systems [4], [5]. The authors of [5] explained that the arrival of WAVE to the ITSs has been promising and will improve the safety of drivers and passengers, enhance the efficiency of road transportation, and make the transportation experience more enjoyable with several innovative services such as collision warning, lane change assistance, speed limit notification, intelligent navigation, road traffic control, and multimedia content delivery. According to [6], the goal of the ITSs is to contribute to a smarter use of the transportation networks, allowing a better management of time and resources.

The counting of vehicles at a signalized intersection can be used by local entities to determine the phase lengths and cycle times for the traffic lights [7], to improve the traffic flow and to reduce oil consumption. At the present time, most of the techniques or methods existing in the ITSs for vehicle number estimation requires a dedicated expensive infrastructure, such as pressure pads, inductive loops under the road surface with magnetic field detectors, digital/video/ thermal cameras, pneumatic road tubes, radar counters, infrared beams, and piezoelectric sensors [8]. These techniques require the detection devices to be pre-installed [5] and suffer from low reliability and limited coverage, high likeliness to be damaged, as well as high deployment and maintenance costs [8]. For example, video-based solutions are easily influenced by rains, fog, illumination variations, and moving shadows during the day [9]. Likewise, inductive loop detectors have a high failure rate, the wires often break due to dilatation as a result of the changing weather conditions, and they obstruct traffic during installations and maintenances. Similarly, infrared sensors are affected to a large extent by fog [10].

Counting vehicles with WAVE technologies has several strengths [11], [12]. For example, illumination and visibility will not affect the counting process. The weather conditions will barely affect the operation of WAVE. Also, the cost of installation and maintenance of WAVE will be shared between vehicle owners that will have to install an OBU (On-Board Unit) in their vehicles, and the road administration (town hall, city hall, county administration, state government, highway administration, etc.) that will have to put RSUs (Road Side Units) at key locations on roads. The ever-increasing number of vehicles equipped with WAVE capabilities provides new solutions to estimate the number of vehicles more accurately and in a faster way, than the traditional techniques and methods previously described [5].

In this paper, we present an algorithm to count vehicles on roads, specifically those vehicles that are stopped at traffic lights at an intersection with two-way multi-lane roads, using vehicular wireless communications. To study the behavior and performance of our algorithm, we use Veins (Vehicles in Network Simulation) [13], an open-source simulation framework that ties together two simulation tools: OMNeT++ for the network simulation and SUMO for the microscopic road traffic simulation. The two simulators run in parallel and communicate with each other through a TCP connection using a protocol called TraCI (Traffic Control Interface). The results of our experiments seem to indicate that the algorithm performs an effective vehicle counting, with a reduced response time, using a small number of control messages sent by the nodes during the counting process, under different scenarios of vehicular traffic density (light, medium, and heavy).

The rest of the paper is organized as follows. Section II discusses the related work. In Section III, we introduce our WAVE-based algorithm to count vehicles that are stopped at a traffic light in intersections, and we explain its characteristics in details. A brief description of the simulation tools and scenarios that we use to validate the proposed algorithm is made in Section IV. The evaluation and analysis of the performance results of our experiments are presented in Section V. Finally, Section VI concludes the paper with outlooks on future research directions.

II. RELATED WORK

The number of vehicles on roads is one of the key parameters used to monitor road traffic conditions and to provide useful predictions for traffic congestion [8]. Currently, most of the work done in the specialized literature to estimate the number of vehicles on roads is based on techniques or methods that use specialized infrastructure (e.g., pressure pads, inductive loops under the road surface with magnetic field detectors, digital/video/thermal cameras, pneumatic road tubes, radar counters, infrared beams, and piezoelectric sensors) [14]–[16]. For example, images from road side cameras are used for traffic monitoring and density estimation in [17]. To achieve their goal, the authors employ a Kalman filter-based background estimation. The difference between the incoming image and the calculated background is used to mark vehicles and to estimate the density of vehicles on the road. Anand *et al.* [18] propose a similar approach using data fusion in which the flow measured from video cameras on the road and the travel time measured from GPSs are used to estimate the density of vehicles. The authors of [19] apply a neural network technique on the data collected from a video monitoring system to determine the density of vehicles. In [20], the authors present a density analyzer scheme based on counting the number of vehicles in live images. To do so, the authors obtain images of a section of the road, crop them to filter out unnecessary information, make the difference of them with a reference image, and finally count the vehicles in the resulting images. In [21], a vehicle counting system is proposed based on three vision sensors. In [22], cumulative road acoustics are used in estimating road traffic density. The authors also analyze the impact of noise on the estimation

of the density. In [23], the authors present a new model-based state estimator supported by the Extended Kalman Filter technique (EKF), in which the discretized Lagrangian Lighthill-Whitham-Richards (LWR) model is used as the process equation, and in which observation models for both Eulerian and Lagrangian sensor data (from loop detectors and vehicle trajectories, respectively) are incorporated to estimate vehicle density on a road segment.

On the other hand, up to now, there have been very few studies that use vehicular wireless communications to estimate the number of vehicles on roads. For example, in [24], with the assumption that each vehicle knows its own location and the location of all other vehicles, a density estimation algorithm was designed based on information of direct neighbors (1-hop neighbors) and 2-hop neighbors. A fully distributed grouping approach is used for density estimation in [25], where group leaders compute vehicle density and disseminate this information among other members of the group. In [5], the authors introduced a density estimation algorithm that uses a simple count of neighbors in each hop. The proposed algorithm does not rely on the location of the nodes and can utilize information of neighbors of an arbitrary number of hops. In [26], the density is calculated based on the number of local neighbors. Then, the global density of the road is estimated by supposing that the inter-vehicle spacing is exponentially distributed, which is not the case for all possible traffic scenarios. Muhammad *et al.* [27] presented a distributed method for road traffic estimation named ''Road Oriented Traffic Information System (ROTIS)''. It is exclusively designed for a city environment and limited to two-way multi-lane roads, to precisely estimate the road traffic density. The authors of [11] proposed an approach similar to [26] that does not require fixed-sized cells and lets the probe vehicle to take samples of the density anywhere within the area of interest.

The authors of [28] introduce a novel algorithm for counting vehicles stopped at a traffic light using VANET technology. The algorithm is based on the idea of the propagation of a count request message from the RSU (originating node) toward the vehicles that are at the end of the waiting line, and the propagation of a response message (with the number of vehicles counted) in the opposite direction, that is, from the vehicles at the end of the line toward the RSU. In [29], the authors propose an approach that uses a combination of vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications. The V2V communication scheme incorporates the procedure of density calculation of vehicles in clusters, and the V2I communication is employed to transfer the computed density information and prioritized movement information to the road side traffic controller. The main traffic input for applying traffic assessment in this approach is the queue length of vehicle clusters at the intersections.

The authors of [30] suggested a ''four way road intersection traffic light control model'' which obtains traffic density information in different lanes at the intersection by taking advantage of V2V and V2I communication for transferring the acquired data to a traffic controller. The process of registration and vehicle count is carried out using Clustering. The algorithm employs six different types of packets named 1) Initiation Packet, 2) Cluster Header Packet, 3) RSU Acknowledgment Packet, 4) Cluster Packet, 5) Reply Packet, and 6) Density Packet, for selecting the cluster head, becoming a cluster's member, and transferring the density estimation data to the traffic controller. The first vehicle in each lane is selected as the head of the cluster. Unlike our proposal, there is no multi-hop count. That is, each cluster head will only count the vehicles that are within a 1-hop distance (1-hop neighbors).

Unlike the previous work, we propose the strategy of dividing the road into regions of a fixed size where a region leader is elected and in charge of estimating the local number of vehicles, and its diffusion. Additionally, each region of the road is split into three equal zones with the objective of selecting the Region Leader as one of the vehicles that are positioned in the closest zone to the region center. Our strategy allows the counting of all the vehicles that are stopped at a particular traffic light. That is, some of the vehicles can be 2-hop, 3-hop, 4-hop away from the RSU (even farther away), and the RSU will still have them in the counting.

It is worth remembering that clustering vehicles has been done in many research proposals, and the methodology used mainly depends on the applications that the authors plan to solve, or the period of time for data recollection. For example, Chen *et al.* [31] used temporal and spatial data of vehicles, collected in different points of a city, to classify them for a better support of possible services. Once the data were collected, they used the k-mean clustering algorithm, and the identification of vehicles was done by license plate recognition. Praveen *et al.* [32] compared two clustering algorithms (an enhanced version of k-mean proposed by them, and the Clarke and Wright's saving method) to group vehicles in an attempt to reduce the total distance and the total number of vehicles which is used, to deliver goods to customers. However, as stated previously, there are just a few works in the area of vehicle counting, which the creation of regions and the counting of vehicles that are beyond the signal transmission range (multi-hop counting).

III. PROPOSED ALGORITHM FOR COUNTING VEHICLES STOPPED AT A TRAFFIC LIGHT AT AN INTERSECTION

In this section, we describe the characteristics and the details of the design and implementation of our novel algorithm to count vehicles stopped at an intersection, waiting for the traffic light to change from red to green.

A. PRELIMINARIES

Our algorithm is based on a penetration rate of 100% for WAVE, i.e., it requires all the vehicles to be equipped with an OBU for communications. Vehicles that do not have a WAVE device will not be counted, since there is no way to detect them. However, we only require the usage of a unique channel, of the seven channels available in the Designated Short

Range Communication (DSRC) band (5.850–5.925 GHz) [33]. The algorithm also assumes that vehicles are equipped with a Global Positioning System (GPS) receptor, so that they can position themselves on digital maps [34]. It is also expected that the communication between neighboring vehicles is bidirectional, i.e., if vehicle V_1 can communicate with vehicle V_2, then a transmission from V_2 will also reach V_1.

In our algorithm, each vehicle is required to maintain a 1-hop neighbors list in which geographic position and speed of each neighbor vehicle are recorded.

B. STRUCTURE OF A MsgCount MESSAGE

In the counting process, vehicles that act as the leader of a region of the road are responsible for performing a local counting of vehicles in their respective region, by consulting their neighbors list. The *Region Leader* of the last segment of the road is in charge of initiating the process of sending a *MsgCount* message. For that, this *Region Leader* will set, in a field called *Number Vehicles Road* of the Protocol Data Unit (PDU), the number of valid neighbors of its region (denoted as *Number Vehicles Region* in the algorithm) obtained from its neighbors list, before sending this PDU (a unicast *MsgCount* message) toward the RSU. The PDU of the *MsgCount* message has eight fields as shown in Fig. 1.

Each node has a unique identifier and must place it in the *Sender ID* field before sending a *MsgCount* message. *Receiver ID* must be filled with the receiver node identifier of the *MsgCount.* This receiver is the leader node of the next adjacent region toward the RSU. *Timestamp* is set by the last *Region Leader* in the line of vehicles, when it sends the initial *MsgCount* message. In other words, it is the timestamp of the creation of the initial *MsgCount* message and is aimed to control out-of-date messages and replay attacks. The synchronization of time between the different nodes is solved with the time received from the GPS satellites. *Message Type* is a field to identify the type of message. In the case of a *MsgCount*, it must be set to 0. *Message Direction* indicates in which direction the message must be transmitted. For this, the lowest bits of the *Message Direction* field are used to indicate one of possible directions (North, South, East, West, Northeast, Northwest, Southeast, and Southwest). The value of the *Message Direction* field is set by the leader of the last region of each road when it sends the initial *MsgCount* message, and not changed during the journey of the PDU toward the RSU. *Road ID* is a field to identify the portion of the road on which the counting is done. *Region ID* is the identifier of the starting region, so that the RSU can infer an approximation of the length of the line. It is set by the last leader before sending the *MsgCount* message, and corresponds to the ID of the region of the last leader. *Number Vehicles Road* is set with the total number of vehicles counted up to now during the transmission of the *MsgCount* message by leader nodes. In other words, before the retransmission of a *MsgCount*, a leader node must update the value of this field by adding to it the number of valid neighbors (*Number Vehicles Region*) stored in its neighbors list.

C. STRUCTURE OF A BEACON MESSAGE

The vehicle information is exchanged between vehicles (within their propagation range) to maintain a list of 1-hop neighbors that will keep their position and speed [27]. This list is updated periodically through Beacon messages, which have a PDU composed of seven fields, as depicted in Fig. 2.

Sender ID must be filled with the sender node identifier. *Receiver ID* represents the identifier of the receiver nodes, in this case 0xFFFF to specify a broadcast. *Timestamp* is the actual time set by the node when it sends a Beacon message and is aimed to control out-of-date messages and replay attacks. *Message Type* is a field to identify the type of a message and must be set to 1 for *Beacon* messages. *Status ID* refers to the different states that a node can take during the region leader election. As described in Section III.E.4, the possible values of *Status ID* are: TRAVELING, LISTENING, COMMONLEADER, LASTLEADER, and NONLEADER. *Sender Position* and *Sender Speed* represent the actual position (latitude and longitude) and speed of the node when it sends the Beacon message, respectively.

D. ALGORITHM TO DISCOVER 1-HOP NEIGHBORING VEHICLES

As mentioned previously, we propose an algorithm to discover 1-hop neighbors that is aimed for helping in: 1) the election of the region leaders, 2) the counting of vehicles in regions, and 3) the selection of the next-hop in the transmission of a *MsgCount* message toward the RSU. The algorithm uses Beacon messages to periodically exchange the necessary information between any two in-range neighbors to maintain a list of 1-hop neighbors, in which the position and speed of each neighboring vehicle are recorded. The number of valid neighbor vehicles around one node can be easily obtained from its neighbors list. Every node periodically broadcasts Beacon messages (see Fig. 2), so that 1-hop neighbors are aware of its presence, position, and speed. When a vehicle receives a Beacon message, it first checks the *Timestamp* field (see Fig. 3) to validate that the Beacon message is current and not a copy of a previous message injected by a replay attack. If the *Timestamp* is valid, then the node checks whether or not the *Sender ID* exists in its list of 1-hop neighbors. If the *Sender ID* does not exist, a new entry is created to register the position and speed information of this neighbor, and a timer is initialized to control the aging of the entry. Otherwise, the position and speed information of the sending vehicle is just updated, and the associated timer is reset. With this information, the node can interpolate the actual position of its 1-hop neighbors at any time. Moreover, entries in the neighbors list that are not updated during a certain period of time will be considered stale and then removed. The flow diagram for creating/updating a 1-hop neighbors list using Beacon messages is given in Fig. 3. The Beacon interval is set to 0.5s to ensure that the information in the neighbors list is always up-to-date.

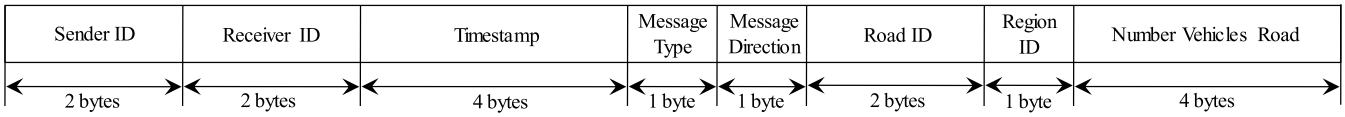

FIGURE 1. Structure of a MsgCount message.

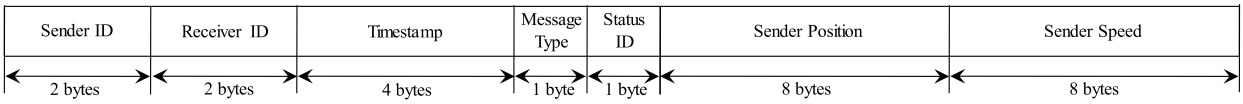

FIGURE 2. Structure of a Beacon message.

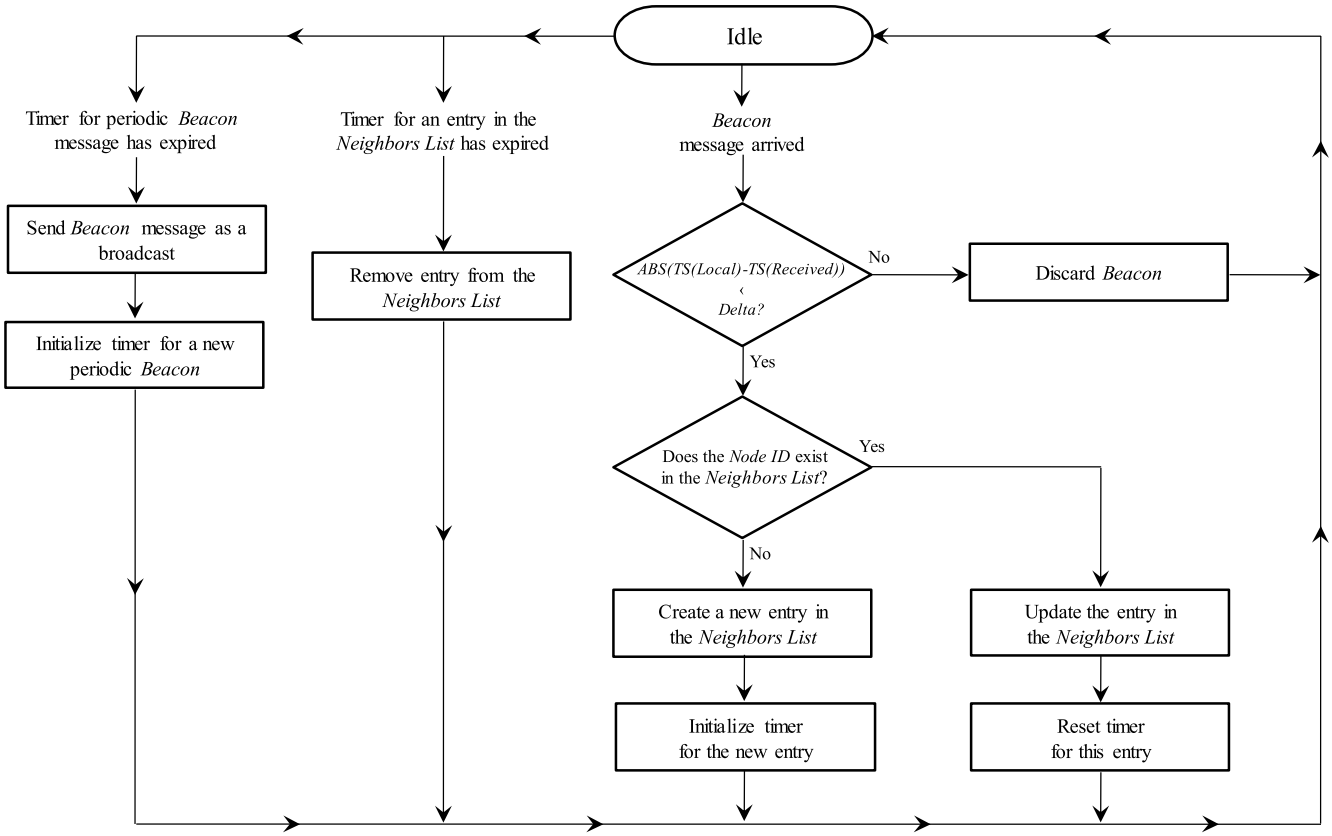

FIGURE 3. Flow diagram for updating the neighbors list.

E. PROPOSED MECHANISM TO DIVIDE THE ROAD INTO ROAD REGIONS

In order to achieve an effective vehicle counting, a reduced response time and total number of control messages sent by the nodes, we propose to divide the road into regions or segments of a fixed size (called in the algorithm as *Region Size*). The region size varies from road to road, but remains the same on a single portion of road. In the proposed algorithm, *Region Size* is selected as a fixed value smaller than the signal propagation range. Equation (1) can be used to determine the number of regions (called *Number Regions* in the algorithm) for a particular portion of road of length *Road Length* (obtained through the digital map):

$$Number\ Regions = \left\lceil \frac{Road\ Length}{Region\ Size} \right\rceil \tag{1}$$

In each region, a vehicle is designated as the *Region Leader*. We assume that each vehicle knows its own position (called *Node Position* in the algorithm) and the position of a reference point or collection point (called *Reference Position* in the algorithm) on the road. This *Reference Position*, for example, can be obtained from the digital map and can correspond to the geographical location of an RSU that receives the counting results, before its process. The *Region ID* of each node can be calculated using by (2):

$$Region\ ID = \left\lfloor \frac{abs\ (Node\ Position - Reference\ Position)}{Region\ Size} \right\rfloor \tag{2}$$

On the other hand, we also make a zonification in each region of the road as shown in Fig. 8. The area from the beginning of the region to its center is split into three equal

zones called Z_0, Z_1, and Z_2, where Z_0 and Z_2 are adjacent with the beginning and the middle of the region, respectively. The idea of these three zones is to select the *Region Leader* as one of the vehicles that are positioned in the closest zone to the region center.

Also, by dividing a road segment into regions, we can better control the transmission storm problem. If this problem is recurrent in a segment of road, the region size can be reduced to keep the network traffic inside a smaller geographical area. As a last solution, several channels can also be used to minimize possible transmission storm problems.

1) MESSAGES SENT BY THE NODES DURING THE REGION LEADER ELECTION

In this section, we describe the messages that are used during the process of the election and maintenance of a region leader.

(a). *MsgLdrDiscover*: This message is broadcasted by a node every time it enters and stops in a new region, with the purpose to rapidly discover the region leader of the same.
(b). *MsgLdrReaffirm*: This message is sent by a region leader for two reasons: First, it is sent as a unicast message to respond to a region leader discovery message (*MsgLdrDiscover*), and second it is broadcasted periodically by a region leader to reaffirm its leadership in its region.
(c). *MsgLdrLeave*: When a leader leaves its region, it broadcasts this message to inform the other nodes of its region so that they can compete for the leadership.
(d). *MsgLdrChange*: This message is sent by the region leader to a non-leader node in a better zone (zone closer to the region center) to yield the role of region leader.

A *MsgLdrChange* is sent as a unicast message, while *MsgLdrDiscover* and *MsgLdrLeave* are propagated as broadcast (the *Receiver ID* must be 0xFFFF). However, a *MsgLdrReaffirm* can be sent as a unicast or a broadcast by a region leader.

2) STRUCTURE OF MESSAGES SENT BY THE NODES DURING THE REGION LEADER ELECTION

The PDU of the messages of Section III.E.1, which are sent by the vehicles during the election and maintenance of the leader of a region, is composed of seven fields, as illustrated in Fig. 4.

The *Sender ID* field denotes the identifier of the vehicle sending the message. The *Receiver ID* field must be filled with the identifier of the receiver node (0xFFFF for broadcast). *Timestamp* is the actual time set by the node at the time of sending the message, and is aimed to control out-of-date messages and replay attacks. *Message Type* is used to identify the type of message sent, and has a value of 2, 3, 4, and 5 for *MsgLdrDiscover*, *MsgLdrReaffirm*, *MsgLdrLeave*, and *MsgLdrChange*, respectively. The *Road ID* field identifies the portion of road in which the node is traveling. *Region ID* corresponds to the identifier of the region of the road in which the node is currently located. The *Leader ID* field contains the identifier of the actual region leader. That is, when a node receives a *MsgLdrReaffirm*, *MsgLdrLeave*, or *MsgLdrChange* message, the *Leader ID* field represents the identifier of the actual region leader. It is an unused field for a *MsgLdrDiscover* message.

3) TIMERS USED IN THE REGION LEADER ELECTION AND IN THE COUNTING OF VEHICLES

In our algorithm, the following timers are used in the processes of region leader election and maintenance, as well as in the vehicle counting:

(a). *TimerCheckStopped*: When this timer expires, the node will check its speed in order to verify if it has stopped in a new region. If so, it will send a leader discovery message (*MsgLdrDiscover*).
(b). *TimerCheckLdr*: It is the maximum time that a node will wait (after sending a leader discovery message *MsgLdrDiscover*), either for the arrival of a response message (unicast *MsgLdrReaffirm*) or for the arrival of a periodic leadership confirmation message (broadcast *MsgLdrReaffirm*) from the region leader. If this timer expires without receiving a leadership reaffirmation on time, then the actual node becomes the leader of its region.
(c). *TimerLdrReaffirm*: It allows a region leader to send message *MsgLdrReaffirm* periodically.
(d). *TimerLdrChange*: This timer is used by a region leader to periodically check whether there is a node (in state LISTENING or NONLEADER) that is in a better zone than it, to yield the leadership.
(e). *TimerLdrAbsence*: This timer allows a non-leader node to decide when it should consider that a new region leader election must take place in the absence of *MsgLdrReaffirm* from the region leader.
(f). *TimerCheckRegion*: This timer is used by a node to check periodically if it has entered a new region.
(g). *TimerCheckLastLdr*: This timer is used by a region leader to periodically check if it is the last leader in the line of vehicles.
(h). *TimerSendMsgCount*: This timer is used by the last region leader (i.e., the region leader farthest away from the RSU) to initiate the process of vehicle counting periodically.

4) STATES OF A NODE DURING THE REGION LEADER ELECTION AND MAINTENANCE

It refers to the different states that a node can take during the region leader election. The possible values are:

(a). TRAVELING: It is the initial state of a node before it stops in a new region of the road.
(b). LISTENING: Nodes in this state are competing to be elected as the region leader. For that, each node initializes the *TimerCheckLdr* timer which represents the maximum amount of time a node will wait before

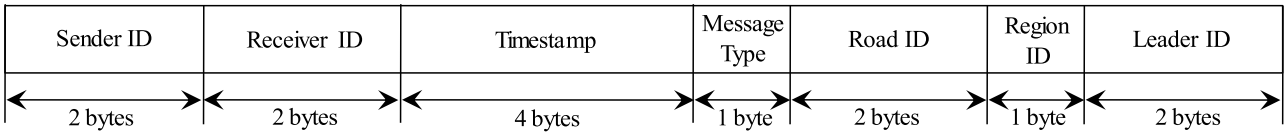

FIGURE 4. Structure of the messages sent by the nodes during the region leader election.

becoming the region leader. If no other node claims to be the new region leader during that time, the node will become the region leader and start to send periodic *MsgLdrReaffirm*.

(c). COMMONLEADER: A node in this state is the actual leader in its region. It is not the last leader in the line, hence, it is responsible for forwarding the *MsgCount* messages toward the RSU.

(d). LASTLEADER: A node in this state is the actual leader in its region and, unlike the COMMONLEADER state, it is the last leader located in the queue of vehicles. Therefore, it is in charge of initiating the counting process by propagating *MsgCount* messages.

(e). NONLEADER: The node is a non-leader in its actual region.

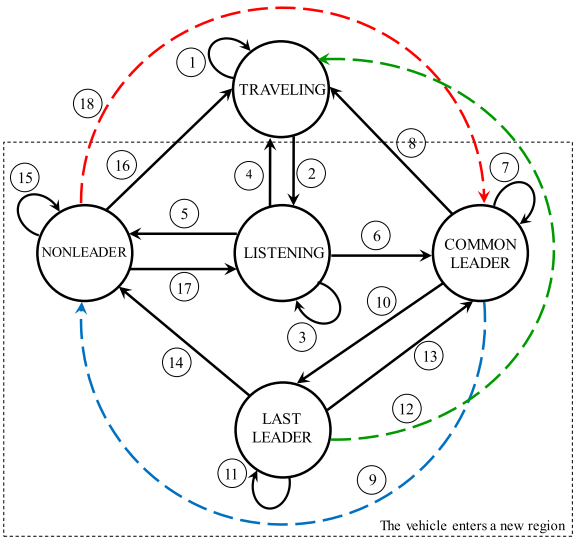

FIGURE 5. State transition diagram of a node.

5) TRANSITIONS OF A NODE DURING THE REGION LEADER ELECTION AND MAINTENANCE

Fig. 5 illustrates the state transition diagram followed by a node during the region leader election and maintenance. These transitions occur for reasons such as: the expirations of the timers described in Section III.E.3, or incoming messages (*MsgLdrDiscover*, *MsgLdrReaffirm*, *MsgLdrLeave,* or *MsgLdrChange*). Fig. 5 omits some reactions to incoming messages that do not result in a state transition. For example, when a region leader node receives a *MsgLdrDiscover* message, it sends back a unicast *MsgLdrReaffirm* message. Bellow, we describe each of the transitions shown in Fig. 5:

1) TRAVELING-TRAVELING. This transition occurs in a node when its *TimerCheckStopped* timer expires and it is still traveling. As a result, the node initializes its *TimerCheckStopped* timer.
2) TRAVELING-LISTENING: This transition occurs when the *TimerCheckStopped* timer expires and the node just entered and stopped in a new region. At this point, the node broadcasts a *MsgLdrDiscover* message and initializes its *TimerCheckLdr* and *TimerCheckRegion* timers.
3) LISTENING-LISTENING: This transition occurs when the *TimerCheckRegion* timer expires and the node is still in the same region. As a result, the node initializes its *TimerCheckRegion* timer.
4) LISTENING-TRAVELING: A node does this transition when its *TimerCheckRegion* timer expires and the node entered a new region. As a consequence, the node initializes its *TimerCheckStopped* timer.
5) LISTENING-NONLEADER: A node changes its state from LISTENING to NONLEADER when it receives a *MsgLdrReaffirm* from the region leader. This reaffirmation message can be either, the response to a previously sent *MsgLdrDiscover*, or a periodic *MsgLdrReaffirm* broadcasted by the region leader. As a result, the node initializes its *TimerLdrAbsence* timer.
6) LISTENING-COMMONLEADER: This change of state occurs when the *TimerCheckLdr* timer expires. As a consequence, the node becomes the region leader and starts to broadcast periodic *MsgLdrReaffirm* message (controlled by its *TimerLdrReaffirm* timer) to its neighboring nodes to announce and maintain its leadership. Additionally, the node initializes its *TimerLdrChange* and *TimerCheckLastLdr* timers.
7) COMMONLEADER-COMMONLEADER: This transition can occur for several reasons. First, it happens when the *TimerLdrReaffirm* timer expires. As a consequence, the node broadcasts a *MsgLdrReaffirm* message to its neighboring nodes to announce and maintain its leadership. Additionally, the region leader initializes its *TimerLdrReaffirm* timer. Second, it happens when the *TimerCheckRegion* timer expires and the node is still in the same region. As a result, the node initializes its *TimerCheckRegion* timer. Third, it happens when the *TimerLdrChange* timer expires and the node does not have to yield its leadership. As a consequence,

the node initializes its *TimerLdrChange* timer. Fourth, it happens when the *TimerCheckLastLdr* timer expires and the node is not the last leader in the vehicles line. As a result, the node initializes its *TimerCheckLastLdr* timer. Fifth, it happens when a *MsgCount* is received. In this case, the node has to update the *Number Vehicles Road* field, and forward the counting message to the next leader toward the RSU. And finally, this transition can happen when there are several leaders in the region and the node receives a *MsgLdrReaffirm* from another leader. In this case, the node discards the reaffirmation message since it was sent by a leader in inferior conditions (in an inferior zone, or in the same zone but with a lower identifier), i.e., the node has to maintain its leadership.

8) COMMONLEADER-TRAVELING: This transition occurs in a common-leader node when its *TimerCheckRegion* timer expires and it entered a new road region. As a result, the node broadcasts a *MsgLdrLeave* message to notify its neighboring nodes that it has left its region and initializes its *TimerCheckStopped* timer.
9) COMMONLEADER-NONLEADER: A node in the COMMONLEADER state becomes NONLEADER for two reasons. On the one hand, when it receives a *MsgLdrReaffirm* message from another leader node of the region in a better zone or in the same zone but with a higher identifier (i.e., several leaders in the region). As a consequence, the node initializes its *TimerLdrAbsence* timer. On the other hand, when its *TimerLdrChange* timer expires and the node determines that there is another node in a better zone (closer to the region center) that should become the region leader. As a consequence, the node sends a *MsgLdrChange* and initializes its *TimerLdrAbsence* timer.
10) COMMONLEADER-LASTLEADER: This change of state occurs when the *TimerCheckLastLdr* timer expires and the node determines that it is the last leader located in the vehicles line. As a result, the node initializes its *TimerCheckLastLdr* and *TimerSendMsgCount* timers.
11) LASTLEADER-LASTLEADER: This change of state can occur for several reasons. First, it happens when the *TimerLdrReaffirm* expires. As a consequence, the node broadcasts a *MsgLdrReaffirm* message to its neighboring nodes to announce and maintain its leadership. Additionally, the region leader initializes its *TimerLdrReaffirm* timer. Second, it happens when the *TimerCheckRegion* timer expires and the node is still in the same region. As a result, the node initializes its *TimerCheckRegion* timer. Third, it happens when the *TimerLdrChange* timer expires and the node does not have to yield its leadership role. As a consequence, the node initializes its *TimerLdrChange* timer. Fourth, it happens when the *TimerCheckLastLdr* timer expires and the node is still the last leader in the vehicles line. As a result, the node initializes its *TimerCheckLastLdr* timer. Fifth, it happens when the *TimerSendMsgCount* timer expires. In this case, the node has to send a *MsgCount* to the next leader toward the RSU and initializes its *TimerSendMsgCount* timer. And finally, this transition can happen when there are several leaders in the region and the node receives a *MsgLdrReaffirm* from another leader. In this case, the node discards the reaffirmation message since it was sent by a leader in inferior conditions (in an inferior zone, or in the same zone but with a lower identifier), i.e., the node has to maintain its leadership.
12) LASTLEADER-TRAVELING: This transition occurs in a node in state LASTLEADER when its *TimerCheckRegion* timer expires and it entered a new road region. As a result, the node broadcasts a *MsgLdrLeave* message to notify its neighboring nodes that it has left its region, and initializes its *TimerCheckStopped* timer.
13) LASTLEADER-COMMONLEADER: This change of state can occur for two reasons. On the one hand, the last leader in the line realizes that it is no longer the last leader because it receives a *MsgLdrReaffirm* message from the leader node of the next region. On the other hand, it realizes that it is not the last leader anymore when it discovers other nodes in the next region when its *TimerCheckLastLdr* timer expires. As a consequence, the node initializes its *TimerCheckLastLdr* timer.
14) LASTLEADER-NONLEADER: The transition from the LASTLEADER state to the NONLEADER state can occur for two reasons. On the one hand, when the node receives a *MsgLdrReaffirm* message from another leader node of the region in superior conditions (in a better zone or in the same zone but with a higher identifier). As a consequence, the node initializes its *TimerLdrAbsence* timer. On the other hand, when its *TimerLdrChange* timer expires and it determines that there is another node in a better zone (closer to the region center) that should become the region leader. As a consequence, the node sends a *MsgLdrChange* message and initializes its *TimerLdrAbsence* timer.
15) NONLEADER-NONLEADER: This transition occurs for two reasons. On the one hand, it happens when a node receives a *MsgLearderReaffirm* from the region leader. As a result, the node resets its *TimerLdrAbsence* timer. On the other hand, it happens when its *TimerCheckRegion* timer expires and the node is still in the same region. As a result, the node initializes its *TimerCheckRegion* timer.
16) NONLEADER-TRAVELING: A node does this transition when its *TimerCheckRegion* timer expires and it entered a new road region. As a consequence, the node initializes its *TimerCheckStopped* timer.
17) NONLEADER-LISTENING: This transition occurs for two reasons. Firstly, when the node receives a *MsgLdrLeave* message from the actual leader node of its region, notifying that it has left the region. As a

consequence, the node initializes its *TimerCheckLdr* timer. Secondly, it happens when the *TimerLdrAbsence* expires due to the absence of leadership reaffirmation messages by the actual region leader. As a result, the node initializes its *TimerCheckLdr* timer.

18) NONLEADER-COMMONLEADER: A node does this transition when it receives a *MsgLdrChange* message from the actual leader node of its region that has determined that it is in a better zone (zone closest to the region center). As a result, the node sends a *MsgLdrReaffirm* and initializes its *TimerLdrReaffirm, TimerLdrChange*, and *TimerCheckLastLdr* timers.

6) PROPOSED APPROACH TO THE REGION LEADER ELECTION

Fig. 6 and Fig. 7 depict simplified flow diagrams based on expired timers and messages received by the nodes in the procedure followed by vehicles in the election and maintenance of the region leaders, and the vehicle counting. In Fig. 6, we can see that the algorithm starts with the nodes with an initial state of TRAVELING. During the time that a node remains in the TRAVELING state, it periodically checks whether it has stopped in a new region. The period is controlled by its *TimerCheckStopped* timer. That is, each time its *TimerCheckStopped* timer expires, the node inspects if it has stopped in a new region. If so, the node changes its state to LISTENING and initializes its *TimerCheckRegion* timer. Additionally, the node sends a *MsgLdrDiscover* message and initializes the *TimerCheckLdr* timer, with the purpose to rapidly discover the region leader. If the node does not receive a leader reaffirmation message before the *TimerCheckLdr* timer expires, then the node changes its state to COMMONLEADER, i.e., this node becomes the new leader of the region and whereas it is in this state, it will broadcast a *MsgLdrReaffirm* message every *TimerLdrReaffirm* seconds. Furthermore, this new region leader will check every *TimerCheckLastLdr* seconds if it is the last leader node on the road. If so, it will change its state to LASTLEADER, where it will start the counting process every *TimerSendMsgCount* seconds. In parallel, this new region leader will initialize the *TimerLdrChange* timer, to check whether there is a better leader. If this is the case, it does the following: 1) it becomes a non-leader (changes its state to NONLEADER), 2) it sends a *MsgLdrChange* message to this better node to notify it that it must assume the leadership of the region, and lastly 3) it initializes the *TimerLdrAbsence* timer in order to monitor the correct operation of the new region leader. If *TimerLdrAbsence* expires, then the node changes its state to LISTENING and initializes its *TimerCheckLdr* timer with the intention of becoming the leader of the region.

When a *MsgLdrDiscover* message is received by the region leader, it responds with a unicast *MsgLdrReaffirm* message to the sending node of the message that will become a non-leader node (will change its state to NONLEADER) and initialize a *TimerLdrAbsence* timer. Similarly, a node transits from state LISTENING to NONLEADER upon receiving a periodic leadership reaffirmation message (*MsgLdrReaffirm*) that is broadcasted by the region leader.

When a region leader receives a *MsgLdrReaffirm* from its region (leadership duplication), it checks if the sender leader node is in a better zone according to the center of the region. If the above is true, then it becomes a non-leader (changes its state to NONLEADER) and initializes a *TimerLdrAbsence* timer. In addition, it cancels any *MsgCount* message that was previously programmed. Similarly, if both leaders are in the same zone, then the leader with the smallest value of *Node ID* will give up his leadership role (it will change its state to NONLEADER), whereas the other will remain as the region leader.

When a non-leader node receives a *MsgLdrChange* message, it becomes the new region leader (it changes its state to COMMONLEADER) and initializes its *TimerLdrChange* and *TimerCheckLastLdr* timers. Additionally, it will send a *MsgLdrReaffirm* every *TimerLdrReaffirm* seconds to reaffirm its leadership.

When the region leader has moved to a new region (i.e., generated for instance by a change of light from red to green), it changes its state to TRAVELING, initializes its check stopped timer (*TimerCheckStopped* timer), and sends out a leader abandon message called *MsgLdrLeave* to ask NONLEADER nodes in the region to start a re-election for leadership immediately. Upon receiving the *MsgLdrLeave*, the nodes in the same region change their state to LISTENING and initialize their *TimerCheckLdr* timer to compete for the leadership of the region.

A node in the state LISTENING, COMMONLEADER, LASTLEADER, or NONLEADER will change its state to TRAVELING, after detecting a change of region, as seen in Fig. 6.

In the implementation of the region leader election and vehicle counting, we select the size of the region (*Region Size*) as a fixed size value smaller than the message propagation range, to guarantee that a message can reach its intended destination. Additionally, when sending messages, we use a dynamic propagation range that ensures their delivery (minimum propagation range), while minimizing the unwanted collisions.

The following observations can be made on our algorithms:

(a). In a road region, the first vehicle to arrive will be elected as the region leader. In subsequent elections, the region leader will be a vehicle that is in the best zone relative to the center of the region, and it will be determined based on the *MsgLdrChange*, *MsgLdrReaffirm*, and *MsgLdrLeave* messages. The region leader can be in any lane of a road region. It is responsible for counting the total number of vehicles in its region and sending/ relaying this information toward the RSU.

(b). When a node receives any of the messages used in the region leader election and vehicle counting (*Beacon*, *MsgLdrDiscover*, *MsgLdrReaffirm*, *MsgLdrChange, MsgLdrLeave*, and *MsgCount*), it will first

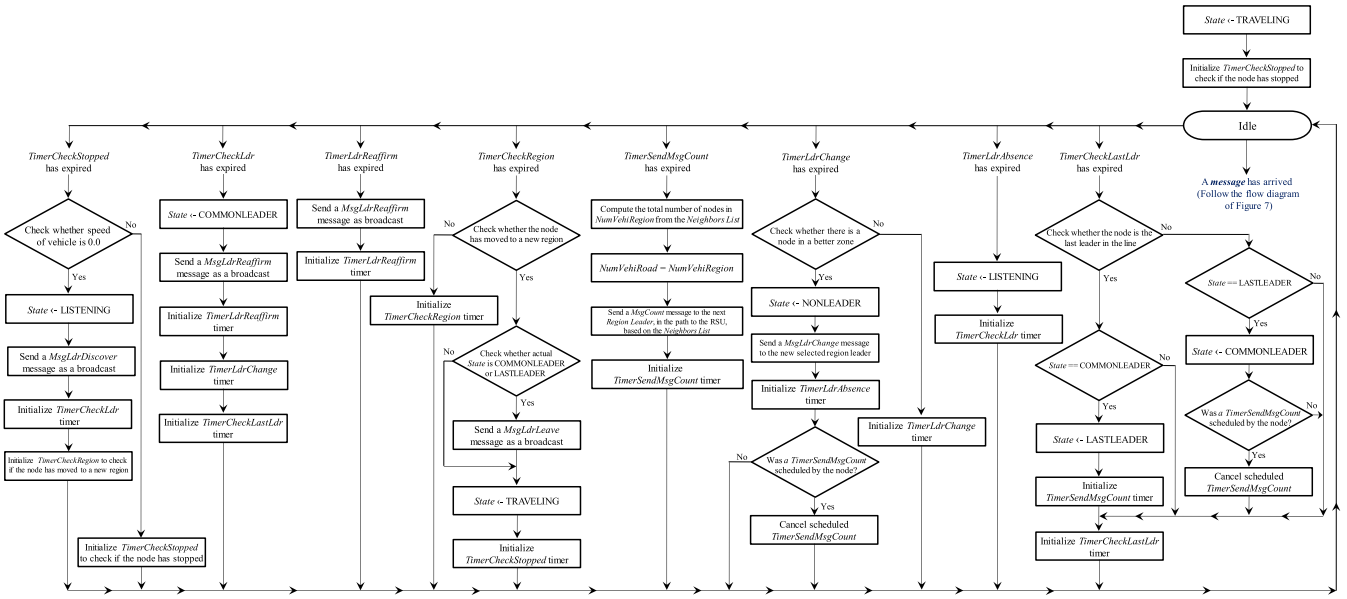

FIGURE 6. Flow diagram of timers used by the nodes in the region leader election and the vehicle counting.

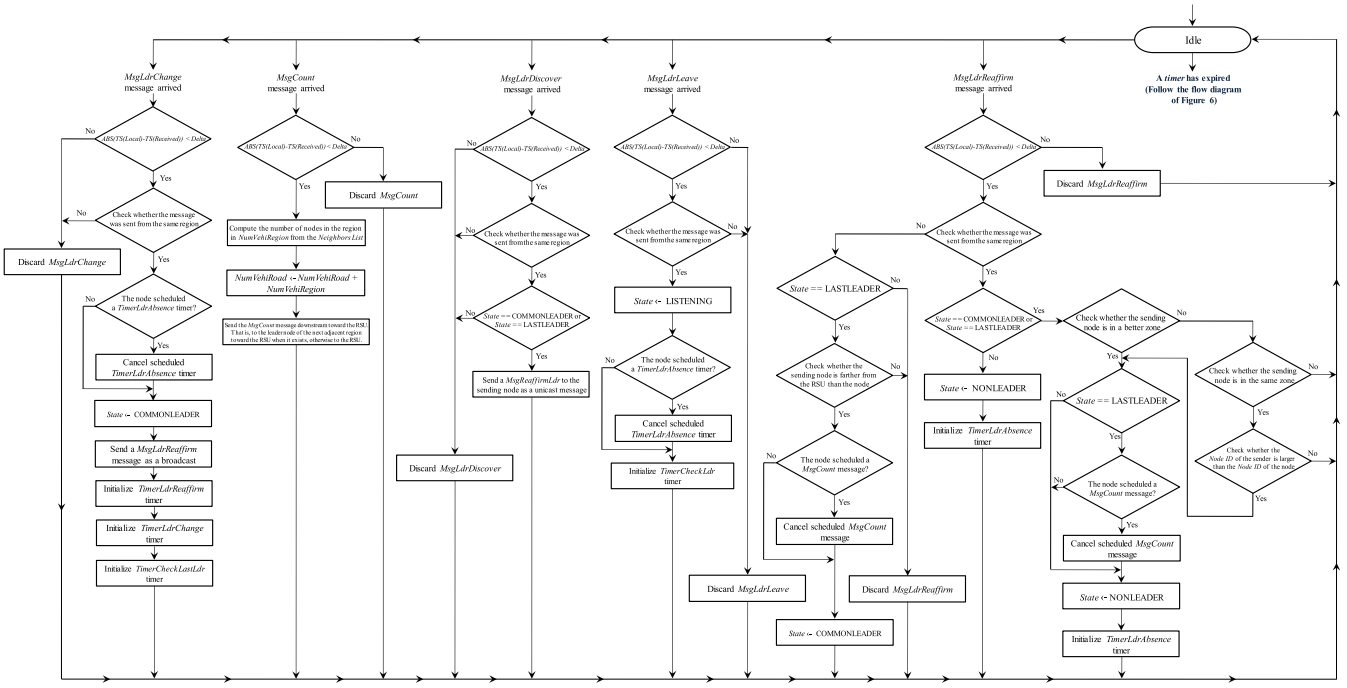

FIGURE 7. Flow diagram of the messages used by the nodes in the region leader election and the vehicle counting.

check if the *Timestamp* field of the received message is valid. If the *Timestamp* is not within the expected interval of time, the node will discard the message.

(c). In order to optimize the process of electing a new leader in a region when the actual leader node leaves the region, we give a higher priority to the vehicles that are in a better zone according to the center of the region. To do this, each node setups its *TimerCheckLdr* timer according to the zone of the region where it stops. For example, the *TimerCheckLdr* timer will be 3 seconds for vehicles that stop in Z_0, 2 seconds for vehicles that stop in Z_1, and 1 second for vehicles that stop in Z_2. As a result, those vehicles that are located in Z_2 will have a *TimerCheckLdr* timer that will expire earlier, and thus will have a higher possibility to become the region leader quickly.

(d). Leadership *duplication* occurs when messages are lost due to collisions. That is, when a node enters and stops in a region, it sends an *MsgLdrDiscover* to discover the leader of the same. If the *MsgLdrDiscover* message is not received by the region leader or its answer (*MsgLdrReaffirm* message) does not reach the

querying node, then this node may promote itself as the region leader. However, steps can be taken to minimize the possibility of duplicate leadership and eliminate it when it occurs. In our algorithm, when leader duplication occurs, the leader in the best zone will keep the leadership, while the other ones will lose it. Now, if both leaders are in the same zone of the region, then the leader with the highest *Node ID* value will remain as the region leader, while the other will abandon its leadership (it will change its state to NONLEADER). This strategy ensures the uniqueness of the region leader.

F. PROPOSED APPROACH FOR COUNTING VEHICLES STOPPED AT A TRAFFIC LIGHT AT AN INTERSECTION

In this section, we present our approach to count vehicles stopped at a traffic light in an intersection with bi-directional roads with multiple lanes on both sides, using vehicular communications. To do this, each road is divided into regions of road of fixed size. In each road region, a vehicle is elected as the region leader based on the proposed approach described in Section III.E.6. The region leader of the last segment of the road is responsible for starting the counting process periodically through the propagation of a *MsgCount* message toward the RSU, every *TimerSendMsgCount* seconds.

For every *MsgCount* sent, the last region leader sets the *Number Vehicles Road* field with the number of valid neighbors of its region, obtained from its neighbors list. It then propagates the *MsgCount* message to the region leader of the region that is adjacent to its region, in the direction of the RSU. That is, the *MsgCount* message is propagated from region leader to region leader. When an intermediate region leader receives a *MsgCount* message, it first validates the *Timestamp* field. If the *Timestamp* is not within the expected interval of time, then the *MsgCount* is discarded (see Fig. 7). Otherwise, the intermediate leader updates the *Number Vehicles Road* field of the *MsgCount* message by adding the number of valid vehicles in its region, and sends it to the next leader node. Eventually, the *MsgCount* message will be given to the RSU, with the *Number Vehicles Road* field containing the total number of vehicles on the road segment, waiting for the traffic light to change.

It is important to emphasize that not all the entries that are in the neighbors list of the vehicles are valid for the counting. That is, the neighbors list also includes vehicles that are moving in the opposite direction. However, these vehicles can be easily discarded based on their position and speed.

G. EXAMPLE OF REGION LEADER SELECTION AND COUNTING OF VEHICLES

In this section, we present an example of the processes associated with the election of region leaders and the counting of vehicles stopped at a traffic light, in the scenario shown in Fig. 8 (a road with a length of 300 m and two lanes in each direction). As shown in Fig. 8, the road was divided into two fixed-size road regions, labeled as R_1 and R_2. The vehicles named as V_{14} and V_{47}, of orange color, represent the region leaders of regions R_1 and R_2, respectively. The labels Z_0, Z_1, and Z_2 correspond to the three zones, from the beginning to the center of a region, for the selection of a better region leader. The red and green dashed lines indicate the maximum propagation range of the messages for leader vehicles V_{14} and V_{47}. To facilitate the explanation of the example, we will assume that the size of each region is equal to 150 m (so that its center is at 75 m from the beginning of the region). Hence according to (1), *Number Regions* is equal to 2. The size of each zone is equal to 25 m, and vehicles have a maximum propagation range of 225 m. Z_0 goes from 0 m to 25 m, Z_1 goes from 25 m to 50 m, and Z_2 goes from 50 m to 75 m. Before the initiation of the counting procedure, vehicles will listen to Beacon messages in order to discover other vehicles that are in their region. When vehicle V_1 stops at the traffic light, it sends a *MsgLdrDiscover* message to all the possible nodes that are present in R_1 and it initializes its *TimerCheckLdr* timer. Since V_1 is the first vehicle to arrive and stop in R_1 where there is no leader at this point, its *TimerCheckLdr* timer will expire without receiving a *MsgLdrReaffirm*, resulting in V_1 declaring itself as the new region leader in R_1 (V_1 changes its state to COMMONLEADER). At this point, V_1 initializes some timers: *TimerLdrReaffirm, TimerLdrChange*, and *TimerCheckLastLdr*. During the time that V_1 remains as region leader, it periodically checks whether a new vehicle has entered the region and has stopped in a better zone. The period is controlled by its *TimerLdrChange* timer. That is, when its *TimerLdrChange* timer expires, the node checks if a new vehicle has stopped in a better zone. Now, when V_2, V_3, V_4, V_5, and V_6 arrive and stop in R_1, they will send a *MsgLdrDiscover* that will be answered by V_1 with a unicast *MsgLdrReaffirm* message, and no change of leadership will be made. However, a leadership change will occur with the arrivals of new vehicles, as they will now stop in a better zone. For example, when V_7 arrives and stops in Z_1, it will periodically send Beacon messages that will be received by V_1. When the *TimerLdrChange* timer of V_1 expires, it determines that V_7 is in a better zone (Z_1). Hence, V_1 will send a *MsgLdrChange* message to V_7, to notify it that it has become the new region leader of R_1. Similarly, a change of leadership will occur with the arrival of vehicle V_{13}, V_{14}, V_{15}, V_{16}, V_{17}, and V_{18} in Z_2. According to the proposed strategy, a change of leadership can occur up to two times in a region. Once the region leader is in zone Z_2, a change of leader should not happen unless it moves to Z_1, Z_0, or out of the region.

The leader selection process described above is repeated in region R_2, where vehicle V_{47} might be elected as the region leader and considered as the last leader of the road.

The next step after choosing vehicles V_{14} and V_{47} as leaders of regions R_1 and R_2, respectively, is to determine the total number of vehicles stopped at the traffic light. The vehicle count is periodically initiated by the last region leader elected (V_{47}) in the line of vehicles. To do so, V_{47} relies on its *TimerSendMsgCount* timer. When its *TimerSendMsgCount* timer expires, using the information from its neighbors list,

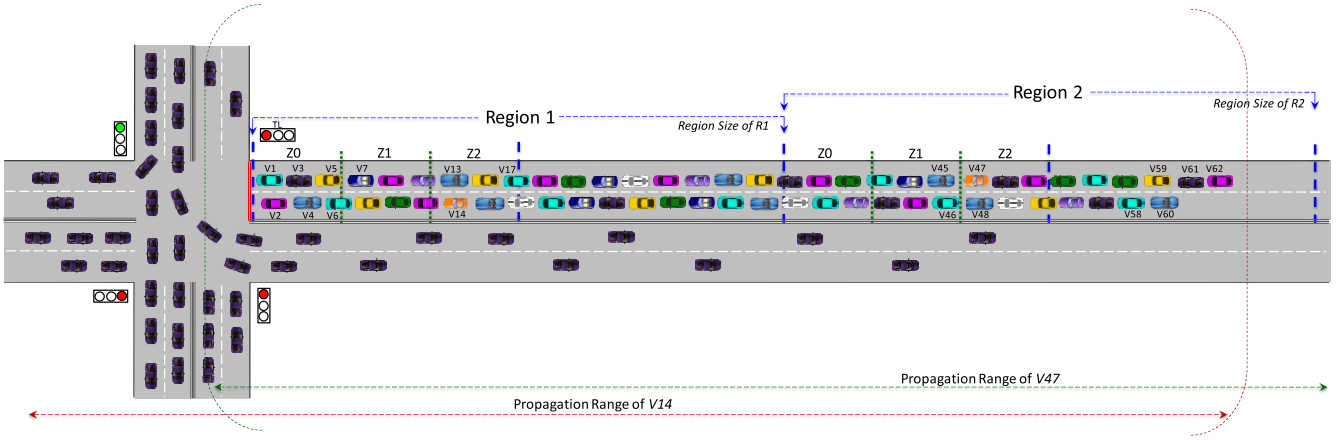

FIGURE 8. Example of the processes associated with the election of region leaders and the counting of vehicles.

V_{47} calculates that there are 28 vehicles that are in region R_2 waiting for the traffic light to change. It also knows that V_{14} is the region leader of the next adjacent region (next forwarder) in the direction of the RSU. So, V_{47} will transmit a *MsgCount* message to V_{14} (see Fig. 8) with the field *Number Vehicles Road* set to 28. When V_{14} receives the *MsgCount* from V_{47}, it determines that there are 34 vehicles in region R_1 and it forwards the *MsgCount* to the RSU with the appropriate changes, by adding 34 to the field *Number Vehicles Road* (its new value is 62). Once the RSU receives the *MsgCount* from V_{14}, it simply processes it.

IV. ENVIRONMENTS AND SCENARIOS FOR SIMULATION

This section aims to present the simulation tools that we use for the implementation of our novel algorithm. We also describe the different parameters selected in our proposal.

A. SIMULATION TOOLS

One of the purposes of this research is to evaluate the accuracy and performance of our novel algorithm, under different scenarios of traffic density. To achieve this goal, we carried out extensive experiments with different sets of parameters.

Nowadays, there are numerous simulation tools ranging from open-source to commercial products. In any research work, it is always essential to choose the most appropriate. A comprehensive study about current simulators, their characteristics, capabilities, and approaches are provided in [35].

The first step required to perform VANET simulations is to use a realistic mobility simulator. A mobility simulator is responsible for defining road networks and generating traffic flows over the roads. We used Simulation of Urban MObility (SUMO) [36], a well-known microscopic and open-source mobility simulator. To enable the vehicles to communicate with each other and with the road side infrastructure, a network simulator is required. For this, we used an object-oriented modular discrete event network simulation framework called Objective Modular Network Testbed in C++ (OMNeT++) [37]. OMNeT++[1] is an open-source, C++ based, multiplatform (Windows, MacOS, and Unix), discrete event simulator for modeling any system composed of devices interacting with each other. One of the main strengths of OMNeT++ is its Graphical User Interface (GUI). Through the GUI, users can create NED files (a description language to define the structure of the model) and inspect the state of each component during simulations [38]. OMNeT++ represents vehicles and RSUs as nodes inside the network and allows communication among these nodes. Finally, in order to bridge the gap between the two worlds (SUMO and OMNeT++), we used an open-source bi-directional simulation framework called Vehicles in Network Simulation (Veins) [13]. Veins couples SUMO with OMNeT++ using the Traffic Control Interface (TraCI) [13]. Veins already implements the WAVE protocol stacks. It is mostly noticeable for IEEE 802.11p, IEEE 1609.4 multi-channel operation, and comprehensive MAC and PHY layers models. We implemented the algorithm on top of WAVE Short Message Protocol (WSMP) and IEEE802.11p.

We chose the Veins framework because it includes a comprehensive suite of models to make vehicular network simulations as realistic as possible, without sacrificing the speed of execution. Also, the GUI and IDE of OMNeT++ and SUMO can be used for quickly setting up and interactively running simulations that facilitate the traceability and debugging of simulation models, for example, by displaying the network graphics, animating the message flows, and letting users peek into objects and variables within the model. Finally, Veins offers interesting features such as online reconfiguration and re-routing of vehicles in reaction to the network simulator.

For all our simulations, we selected WAVE (IEEE 802.11p) for the wireless communication standard. We did several simulations where we varied the transmission rates (6, 9, 12, 18, 24, and 27 Mbps), and obtained similar results. Hence, we chose a bit rate of 18 Mbps for all our simulations. The

[1] http://www.omnetpp.org

two-ray ground model was chosen as the radio propagation model. We simulated different scenarios where vehicles are stopped at a traffic light in a line of vehicles.

B. SIMULATION PARAMETERS

Table 1 summarizes the technical parameters shared by all the scenarios and simulated cases of our algorithm. It is worth noting that the bit rate, modulation, and coding were chosen based on [39].

TABLE 1. Simulation parameters.

Parameter	Value
Simulation Scenario	Simple intersection of four roads and advanced intersection of five roads
Number of Vehicles	5 – 400
Length of Road Section	8 km
Wireless Standard	IEEE 802.11p
Transmission Rate	18 Mbps
Transmission Power	20mW (13 dBm)
Receptor Sensitivity	-89 dBm
Thermal Noise	-110 dBm
Message Type	WSMP data
Channel Bandwidth	10 MHz
Frequency Band	5.850–5.925 GHz
Radio Propagation Model	Two-ray ground
Simulation Time	200 s
Vehicle Beacon Interval	0.5 s

V. ACCURACY AND PERFORMANCE EVALUATION OF THE PROPOSED ALGORITHM

In this section, we present the accuracy and performance evaluation of the proposed algorithm through simulations. The main objective is to discuss whether or not our novel algorithm can efficiently count vehicles that are stopped at the traffic lights of an intersection with a reduced response time and a reasonable number of control messages sent by the vehicles. To this end, we evaluate the algorithm in terms of the accuracy of the counting, the response time to count the vehicles, and the total number of control messages (*MsgLdrDiscover*, *MsgLdrReaffirm*, *MsgLdrLeave*, *MsgLdrChange*, and *MsgCount*) sent by the nodes during the counting.

We selected two scenarios for our simulations: 1) a simple intersection of four roads as shown in Fig. 9 and Fig. 2) an advanced intersection of five roads as shown in Fig. 14. The roads that made up the intersection of the previous scenarios had a length of 4000 m, with several lanes, and they were divided into regions of a fixed size of 75 m. We chose a propagation range value of 112.5 m. In all the scenarios, the counting process was periodically performed over time (every 5 s). The total simulation time was 200 s. The number of nodes in the simulations over time varied between 5 and 400 and the interval between Beacon messages was 0.5 s, thereby allowing the updating of position and speed information every 0.5 s.

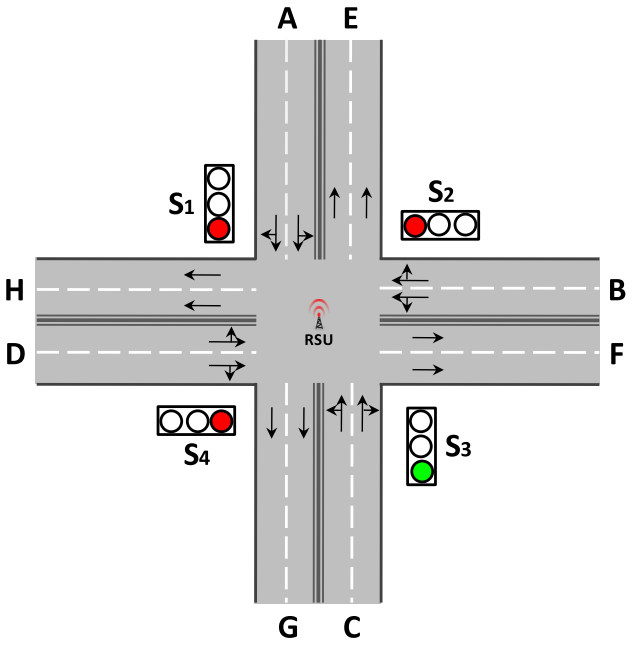

FIGURE 9. Outline of a simple intersection of four roads.

During the simulations, vehicles were introduced at the ends of the roads and they moved toward the traffic light of the intersection. Their initial speed was assigned randomly between 30 km/h and 60 km/h. As vehicles approached the red traffic light of the intersection, they reduced their speed and stopped at the end of their lane, thus forming a line. As soon as the traffic light turned green, vehicles resumed their journey and passed the intersection. It is worth clarifying that the mobility of vehicles will vary from one simulation to the next, since vehicles enter the roads with a random speed. Once a vehicle is inserted, its movement will be restricted by the roads, the traffic lights, and the other vehicles, based on the car-following-model implemented in SUMO.

The scenarios are described and evaluated in more details in the following sections.

A. SCENARIO WITH A SIMPLE INTERSECTION OF FOUR ROADS

The scenario presented in this section simulates an intersection composed of four bidirectional roads with two lanes in both directions.

In Fig. 9, labels A, B, C, D, E, F, G, and H indicate the segments of the roads. These segments of roads were divided into regions of 75 m long. Four traffic lights, denoted as S_1, S_2, S_3, and S_4, control the intersection. The arrows indicate the directions that the vehicles should take when arriving at the intersection. We placed the RSU in the center of the intersection. Since our algorithm can count vehicles in several directions simultaneously, the field *Message Direction* (see Fig. 1) must be set adequately in the *MsgCount* messages,

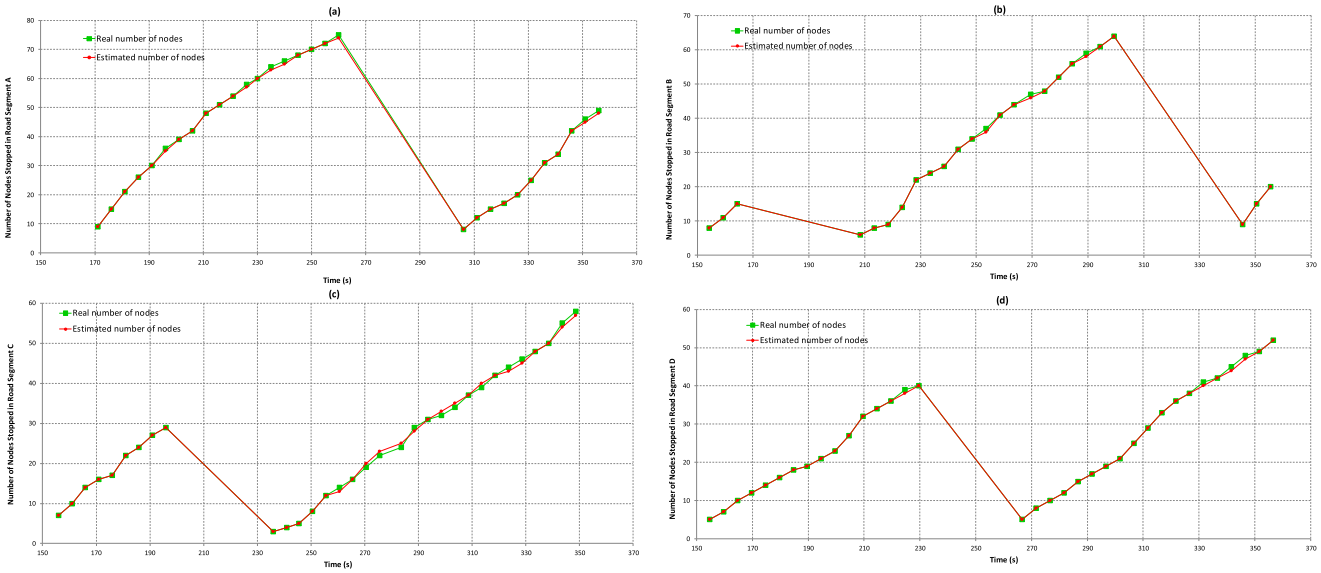

FIGURE 10. Number of vehicles counted in the simple intersection of four roads.

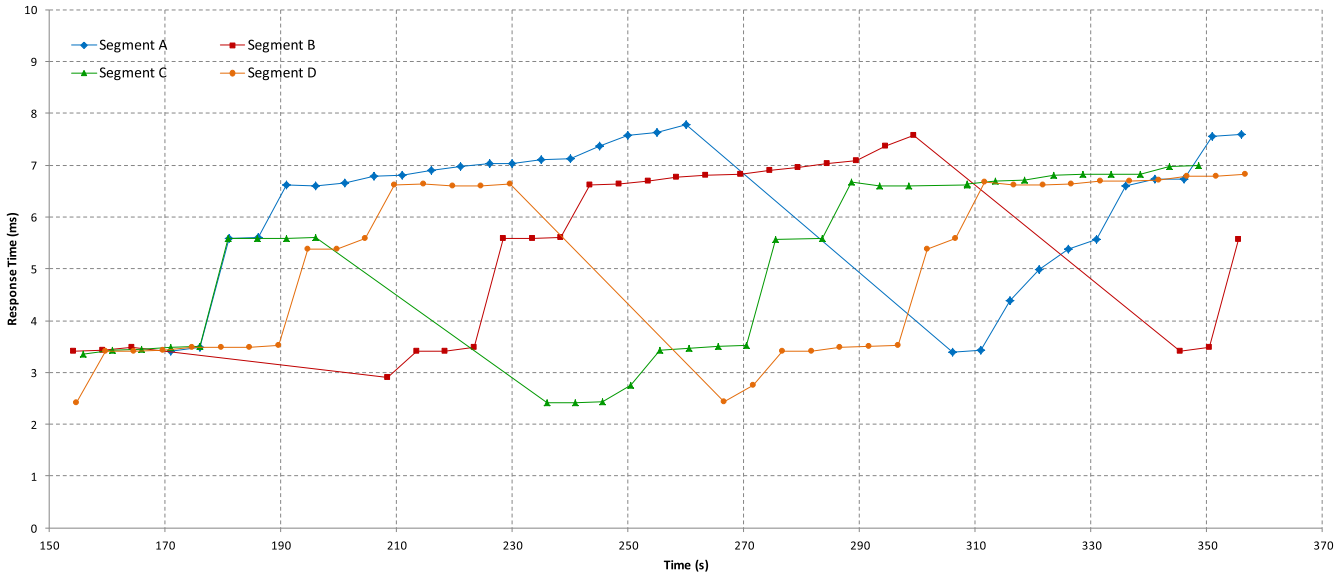

FIGURE 11. Response time for the counting of vehicles in the simple intersection of four roads.

by the leader of the last region of each road. In these experiments, we determined the number of vehicles in segments A, B, C, and D, taking into account the following four light phases: 1) S_1 is green while S_2, S_3 and S_4 are red, 2) S_2 is green while S_1, S_3 and S_4 are red, 3) S_3 is green while S_1, S_2 and S_4 are red, and 4) S_4 is green while S_1, S_2 and S_3 are red. Each phase lasted 35 s.

Fig. 10(a), Fig. 10(b), Fig. 10(c), and Fig. 10(d) depict the real and estimated (using our algorithm) number of vehicles stopped at the traffic lights in road segments A, B, C, and D, respectively, vs. time. The curve in green (with squares) represents the real number of nodes, and the curve in red (with circles) is the number of nodes counted by our algorithm, at a particular time. The real number of nodes is extracted from the SUMO trace files using a script that computes the actual number of vehicles stopped on a particular segment of road, at any given time. The experiments show that the number of vehicles computed by our novel algorithm is very close to the real number of vehicles stopped at the traffic light. The counting error can be defined by using (3). According to our simulations, the counting errors in road segments A, B, C, and D are between 0% and 2.77%, 0% and 2.70%, 0% and 7.14%, and 0% and 2.56%, respectively, which are very low, and permit us to validate our algorithm.

$$Error = \frac{Real\ Number\ of\ Nodes - Estimated\ Number\ of\ Nodes}{Real\ Number\ of\ Nodes} \quad (3)$$

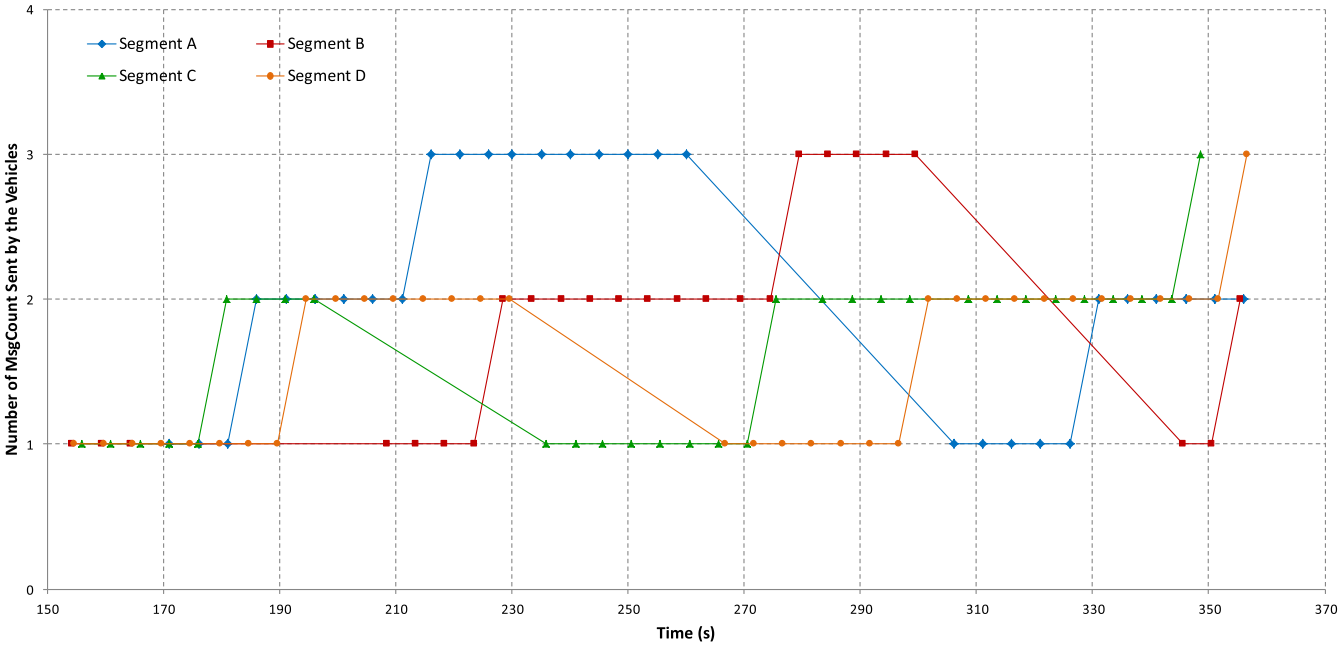

FIGURE 12. Number of MsgCount sent by the nodes during the counting in the simple intersection of four roads.

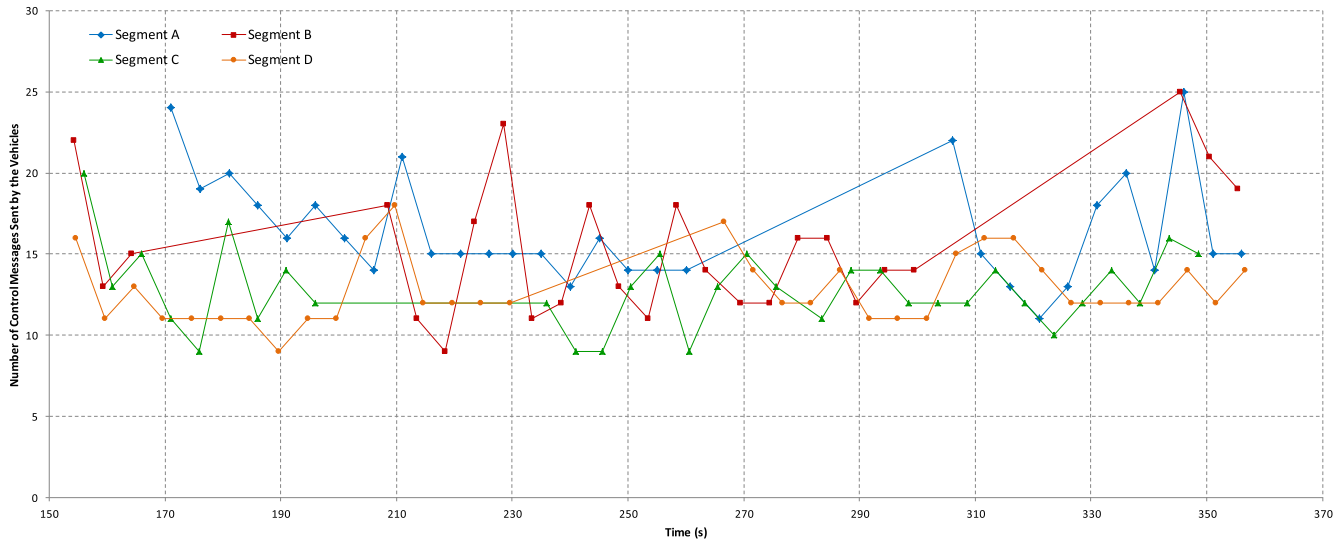

FIGURE 13. Total number of control messages sent by the nodes during the counting in the simple intersection of four roads.

In Fig. 10(a), it is noticeable that the counting of vehicles takes place every 5 s (red dots). There is no counting after time 260 s and before time 305 s. In this period, segment A has the green light. Hence the vehicles are moving and no counting is performed by them. Also, Fig. 10(b), Fig. 10(c), and Fig. 10(d) confirm the traffic light schedule (phase duration of 35 s), resulting in the green light for segment B when $160 \leq \text{time} \leq 195$, for segment C when $195 \leq \text{time} \leq 230$, for segment D when $230 \leq \text{time} \leq 265$, for segment A when $265 \leq \text{time} \leq 300$, for segment B when $300 \leq \text{time} \leq 335$, etc.

Fig. 11 depicts the total time required for the counting to be completed (response time), that is, the time it takes to propagate a *MsgCount* message from the leader of the last region of the road, to the RSU. Fig. 12 represents the total number of *MsgCount* messages sent by the nodes during the counting process vs. time. That is, it gives information on the number of regions that were filled with vehicles when the counting process started. Using Fig. 10(a) and Fig. 12, we can approximately infer that in segment A, the counting process is done with one *MsgCount* message when the number of vehicles is less than 20. The algorithm required two *MsgCount* messages for a number of vehicles between 20 and 50. Finally, three *MsgCount* messages were needed for 50 vehicles or more (notice that in the reported experiments, the number of vehicles in segment A was lower than 80).

Similar results were obtained for the other three segments (B, C, and D).

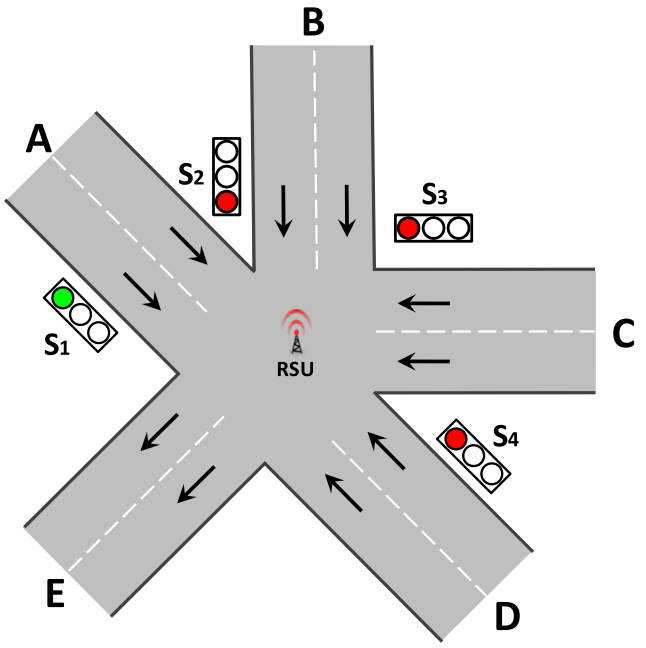

FIGURE 14. Outline of an advanced intersection of five roads.

In Fig. 13, we can see the total number of control messages (*MsgLdrDiscover*, *MsgLdrReaffirm*, *MsgLdrLeave*, *MsgLdrChange*, and *MsgCount*) sent by the nodes in the segments of the simple intersection of four roads during the counting process.

B. SCENARIO WITH AN ADVANCED INTERSECTION OF FIVE ROADS

In this section, we look at the behavior of the proposed algorithm in a more advanced intersection (see Fig. 14). This scenario is composed of four incoming roads (A, B, C, and D) and one outgoing road (E). In the corners of the intersection, there are four traffic lights, denoted as S_1, S_2, S_3, and S_4 to control the traffic. We placed the RSU in the center of the intersection.

In our experiments, we counted the total number of stopped vehicles in road segments A, B, C, and D, vs. time, taking into account the following four light phases: 1) S_1 is green while S_2, S_3 and S_4 are red, 2) S_2 is green while S_1, S_3 and S_4 are red, 3) S_3 is green while S_1, S_2 and S_4 are red, and 4) S_4 is green while S_1, S_2 and S_3 are red. Each phase lasted 35 s, and the vehicles crossed the intersection in the direction of segment E.

Fig. 15, Fig. 16, and Fig. 17 report the results that we obtained in the simulations in relation to the counting of vehicles stopped at traffic lights, the response time to complete the vehicle counting, and the total number of control messages (*MsgLdrDiscover*, *MsgLdrReaffirm*, *MsgLdrLeave*, *MsgLdrChange*, and *MsgCount*) sent by the vehicles in each segment to perform the counting, respectively, over time. Once again, the simulation results evidence that our algorithm effectively performs the node counting, with a short response time, and a low number of control messages sent by the nodes.

Fig. 15 confirms the traffic light schedule (phase duration of 35 s), resulting in the green light for segment C when $60 \leq$ time ≤ 95, for segment D when $95 \leq$ time ≤ 130, for segment A when $130 \leq$ time ≤ 165, for segment B when $165 \leq$ time ≤ 200, for segment C when $200 \leq$ time ≤ 235, etc.

We also compared the performance of our algorithm with the "Four Way Road Intersection Model" proposed in [30], and described in Section II. To make it easier, let us call "Algorithm I" the algorithm presented in this paper and "Algorithm II" the algorithm proposed in [30]. We adapted and implemented Algorithm II using the Veins simulation framework. We decided to choose the scenario of Fig. 9 to compare both counting strategies. We first compared the performance of the two mechanisms, by counting the total number of vehicles that are stopped in each segment of the simple intersection of four roads. And then, we contrasted the total number of control messages sent by the vehicles to complete each counting process. The parameters for these new simulations were the same as the ones previously used for Algorithm I in Fig. 10 and Fig. 13. Algorithm II does have specific parameters, and we used the same values as the ones specified by the authors in [30]. The vehicle counting was performed every 5 s, with a total simulation time of 200 s, a transmission range of 112.5 m, and a region size of 75 m. The number of vehicles ranged from 5 to 400 and the beacon message interval was 0.5 s.

Fig. 18 shows the real and estimated (using Algorithms I and II) number of vehicles stopped at the traffic lights in segment A, vs. time. For each value of the time reported in the abscissa, the results are illustrated in groups of three bars. The first yellow bar represents the real number of nodes stopped at segment A, the second blue bar corresponds to the total number of nodes in segment A computed by Algorithm I, and the third red bar to the number of vehicles in segment A calculated by Algorithm II. It is worth clarifying that there was a counting process at time 260 s, and the next one took place at time 306 s, a difference of 46 s. To justify this behavior, it is essential to recall that no counting is performed while the vehicles have the green light, and as specified previously, the light phase is 35 s. The simulations reported that both algorithms compute a total number of vehicles very close to the real values, with a slight advantage for Algorithm I.

Fig. 19 illustrates the network overhead to count the nodes, which is proportional to the number of control messages sent by the vehicles to carry out the counting process. For each value of the time reported in the abscissa, the results are shown in groups of two bars. The blue and red bars indicate the number of messages sent by the vehicles of segment A with Algorithm I and II, respectively. We can observe that Algorithm I has less network overhead than Algorithm II. Similar results were obtained for the other three segments (B, C, and D), for both the counting process and the overhead generated by the algorithms.

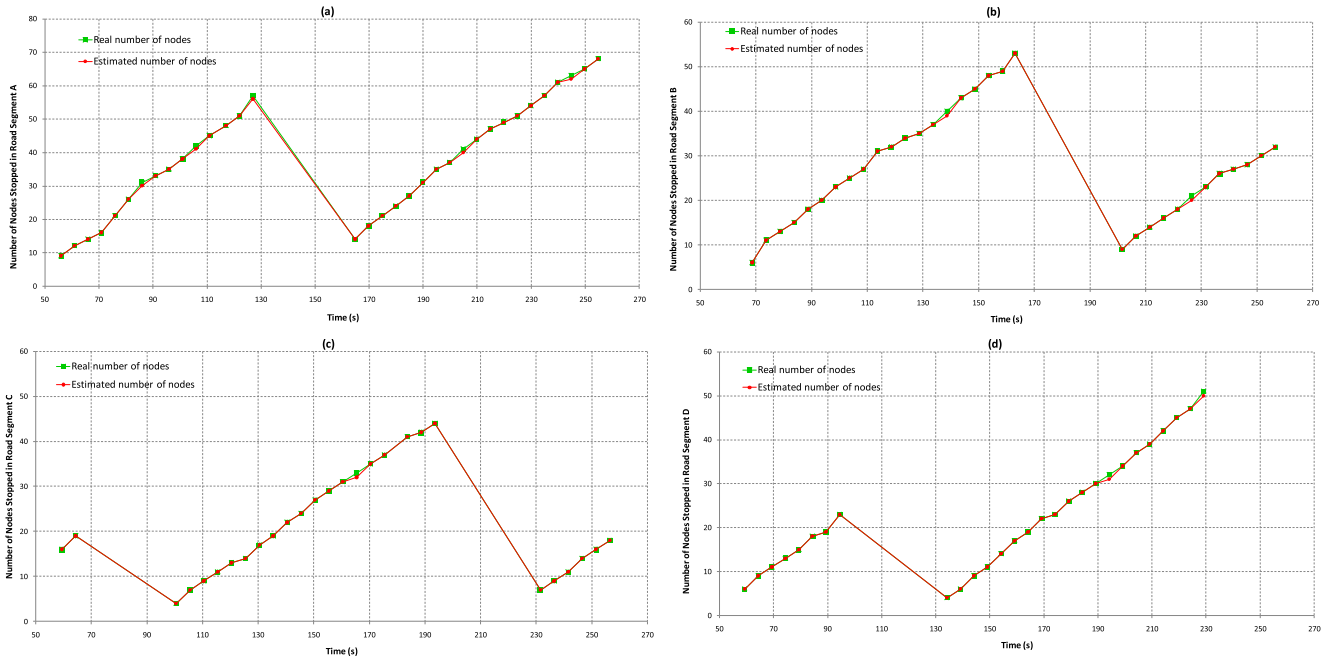

FIGURE 15. Number of vehicles counted in the advanced intersection of five roads.

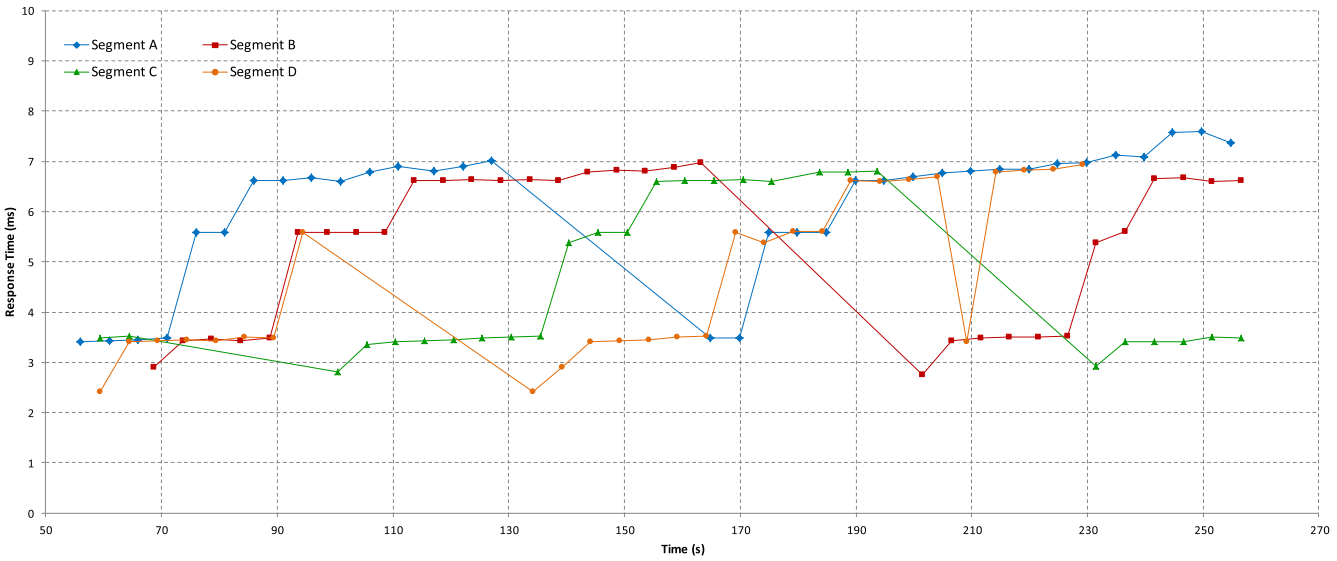

FIGURE 16. Response time for the counting of vehicles in the advanced intersection of five roads.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented a distributed algorithm to determine in real-time the number of vehicles stopped at a traffic light at an intersection using WAVE technology. To do this, we proposed a mechanism to divide segments of roads into fixed-size road regions, where a region leader is in charge of computing and propagating the total number of nodes in its region.

The calculation of the number of vehicles in vehicular environments, specifically at intersections, has many potential applications, specifically in the development of ATCSs. The algorithm proposed in this research can be used to make decisions and improve the traffic conditions at intersections, especially during peak hours.

The algorithm was simulated in different scenarios using SUMO and OMNeT++ as simulators, and Veins as a framework to bidirectionally couple these simulators through the TraCI interface. We performed extensive simulations to analyze and evaluate the accuracy and performance of the proposed algorithm, mainly in two scenarios: 1) a simple intersection of four roads and 2) an advanced intersection of five roads. The results of the experiments showed that our algorithm is very efficient in terms of accuracy in vehicle counting, response time, and number of control messages

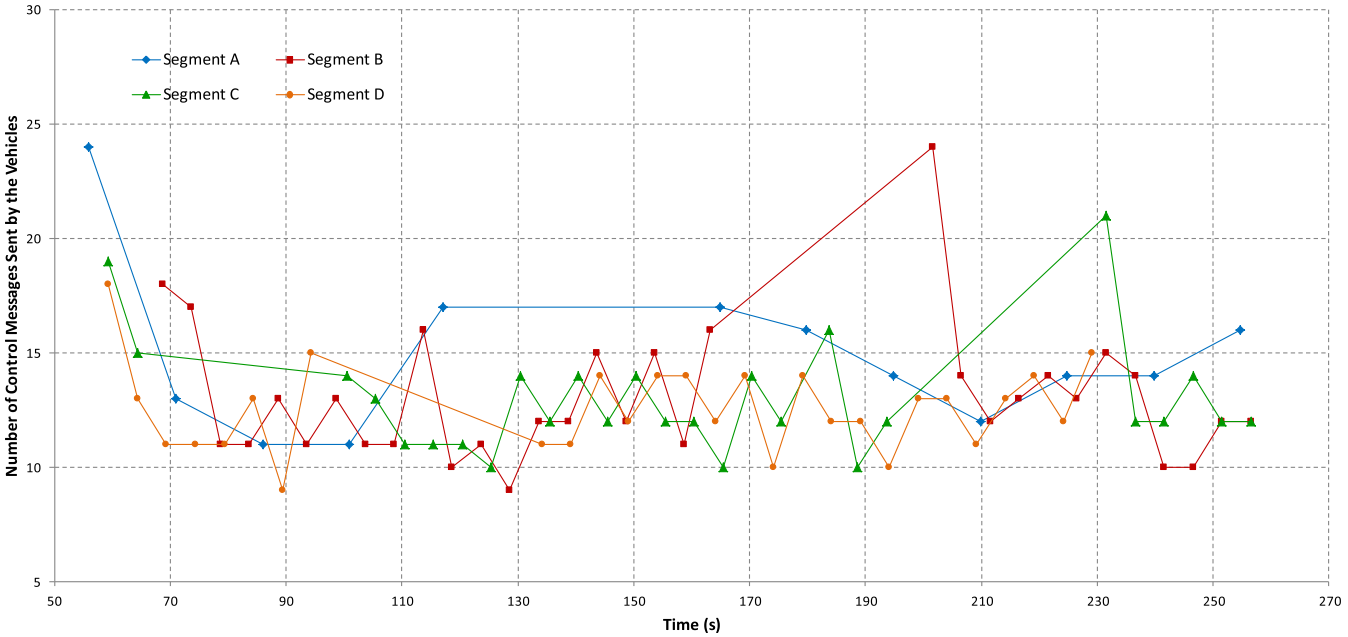

FIGURE 17. Total number of control messages sent by the nodes during the counting in the advanced intersection of five roads.

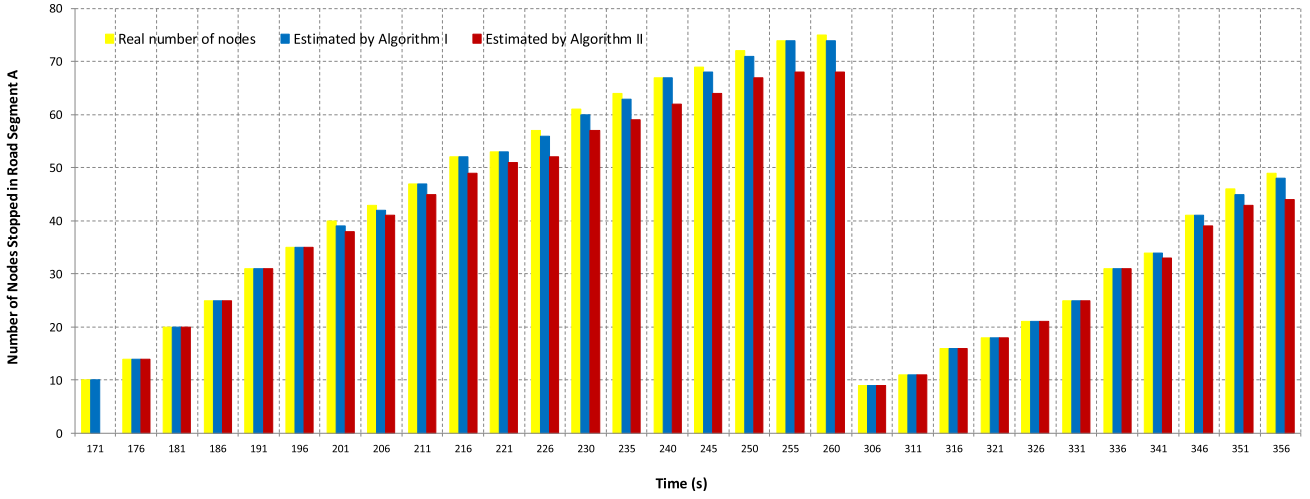

FIGURE 18. Comparison of the number of vehicles counted in segment A.

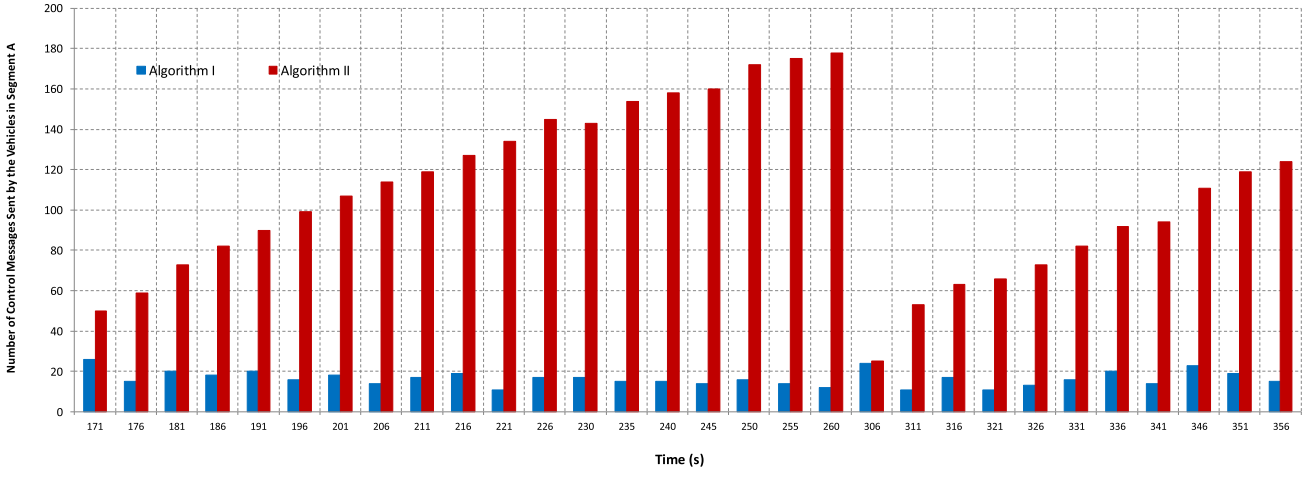

FIGURE 19. Comparison of the number of control messages sent by the nodes during the counting in segment A.

sent by the nodes during the counting process. Additionally, we compared our algorithm with the one proposed in [30], and our simulation results seem to indicate that our technique outperforms this work, especially at the level of the transmission overhead.

As possible future work, we are interested in proposing new counting algorithms in the context of parking lot management systems with the intention of counting the total number of vehicles currently parked and reporting the number of available parking slots with their respective location, with the aim of helping vehicles arriving at a parking in search of a vacant slot. Also, as another research direction, we plan to explore the optimization of traffic flows in cities using the proposed algorithm.

REFERENCES

[1] P. M. Daigavane and P. R. Bajaj, "Real-time vehicle detection and counting method for unsupervised traffic video on highways," *Int. J. Comput. Sci. Netw. Secur.*, vol. 10, no. 8, pp. 112–117, Aug. 2010.
[2] H. Phule, "Public safety application for approaching emergency vehicle alert and accident reporting in VANETs using WAVE," M.S. thesis, Dept. Comput. Eng., Rochester Inst. Technol., Rochester, NY, USA, May 2012.
[3] L. Ramachandran, S. Sukumaran, and S. R. Sunny, "An intersection based traffic aware routing with low overhead in VANET," *Int. J. Digit. Inf. Wireless Commun.*, vol. 3, no. 2, pp. 190–196, 2013.
[4] E. Bas, A. M. Tekalp, and F. S. Salman, "Automatic vehicle counting from video for traffic flow analysis," in *Proc. IEEE Intell. Vehicles Symp. (IVS)*, İstanbul, Turkey, Jun. 2007, pp. 392–397.
[5] R. Mao and G. Mao, "Road traffic density estimation in vehicular networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Shanghai, China, Apr. 2013, pp. 4653–4658.
[6] A. Ribeiro and P. Correia, "Automatic vehicle counting based on surveillance video streams," in *Proc. 8th Congr. Portuguese Committee URSI*, Lisbon, Portugal, Nov. 2014, pp. 1–10.
[7] N. Garber and L. Hoel, *Traffic and Highway Engineering*, 5th ed. Boston, MA, USA: Cengage Learning, Jan. 2014.
[8] N. Akhtar, S. C. Ergen, and O. Ozkasap, "Analysis of distributed algorithms for density estimation in VANETs (poster)," in *Proc. IEEE Veh. Netw. Conf. (VNC)*, Seoul, South Korea, Nov. 2012, pp. 157–164.
[9] M. Lei, D. Lefloch, P. Gouton, and K. Madani, "A video-based real-time vehicle counting system using adaptive background method," in *Proc. IEEE Int. Conf. Signal Image Technol. Internet Based Syst.*, Bali, Indonesia, Nov. 2008, pp. 523–528.
[10] M. Liang, X. Huang, C.-H. Chen, X. Chen, and A. Tokuta, "Counting and classification of highway vehicles by regression analysis," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 5, pp. 2878–2888, Oct. 2015.
[11] R. Bauza, J. Gozalvez, and J. Sanchez-Soriano, "Road traffic congestion detection through cooperative vehicle-to-vehicle communications," in *Proc. 35th IEEE Local Comput. Netw. Conf.*, Denver, CO, USA, Oct. 2010, pp. 606–612.
[12] O. Tomescu, I. M. Moise, A. E. Stanciu, and I. Batros, "Adaptive traffic light control system using ad-hoc vehicular communications networks," *Taiwanese Assoc. Artif. Intell.*, vol. 74, no. 2, pp. 1–8, 2012.
[13] C. Sommer, R. German, and F. Dressler, "Bidirectionally coupled network and road traffic simulation for improved IVC analysis," *IEEE Trans. Mobile Comput.*, vol. 10, no. 1, pp. 3–15, Jan. 2011.
[14] G. Leduc, "Road traffic data: Collection methods and applications," Eur. Commission, Joint Res. Center, Inst. Prospective Technol. Stud., Seville, Spain, Tech. Rep. JRC 47967, 2008.
[15] I.-C. Morarescu and C. Canudas-de-Wit, "Highway traffic model-based density estimation," in *Proc. Amer. Control Conf.*, San Francisco, CA, USA, Jun. 2011, pp. 2012–2017.
[16] A. Tabibiazar and O. Basir, "Kernel-based optimization for traffic density estimation in ITS," in *Proc. IEEE Veh. Technol. Conf. (VTC Fall)*, San Francisco, CA, USA, Sep. 2011, pp. 1–5.
[17] M. Balcilar and A. Sonmez, "Extracting vehicle density from background estimation using Kalman filter," in *Proc. Int. Symp. Comput. Inf. Sci. (ISCIS)*, İstanbul, Turkey, Oct. 2008, pp. 1–5.
[18] R. A. Anand, L. Vanajakshi, and S. C. Subramanian, "Traffic density estimation under heterogeneous traffic conditions using data fusion," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Baden-Baden, Germany, Jun. 2011, pp. 31–36.
[19] C. Ozkurt and F. Camci, "Automatic traffic density estimation and vehicle classification for traffic surveillance systems using neural networks," *Math. Comput. Appl.*, vol. 14, no. 3, pp. 187–196, Dec. 2009.
[20] N. Abbas, M. Tayyab, and M. T. Qadri, "Real time traffic density count using image processing," *Int. J. Comput. Appl.*, vol. 83, no. 9, pp. 16–19, Dec. 2013.
[21] C. Meng, Z. Weihua, and M. Barth, "Mobile traffic surveillance system for dynamic roadway and vehicle traffic data integration," in *Proc. Int. IEEE Conf. Intell. Transp. Syst. (ITSC)*, Washington, DC, USA, Oct. 2011, pp. 771–776.
[22] V. Tyagi, S. Kalyanaraman, and R. Krishnapuram, "Vehicular traffic density state estimation based on cumulative road acoustics," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 3, pp. 1156–1166, Sep. 2012.
[23] Y. Yuan, J. W. C. van Lint, R. E. Wilson, F. Van Wageningen-Kessels, and S. P. Hoogendoorn, "Real-time Lagrangian traffic state estimator for freeways," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 1, pp. 59–70, Mar. 2012.
[24] S. Panichpapiboon and W. Pattara-Atikom, "Exploiting wireless communication in vehicle density estimation," *IEEE Trans. Veh. Technol.*, vol. 60, no. 6, pp. 2742–2751, Jul. 2011.
[25] M. Jerbi, S.-M. Senouci, T. Rasheed, and Y. Ghamri-Doudane, "An infrastructure-free traffic information system for vehicular networks," in *Proc. IEEE 66th Veh. Technol. Conf.*, Baltimore, MD, USA, Sep. 2007, pp. 2086–2090.
[26] S. Panichpapiboon and W. Pattara-Atikom, "Evaluation of a neighbor-based vehicle density estimation scheme," in *Proc. 8th Int. Conf. ITS Telecommun.*, Phuket, Thailand, Oct. 2008, pp. 294–298.
[27] S. Muhammad, A. Rehman, S. Ullah, S. Madani, B. Nazir, and M. Othman, "Road oriented traffic information system for vehicular ad hoc networks," *Wireless Pers. Commun.*, vol. 77, no. 4, pp. 2497–2515, Aug. 2014.
[28] M. Contreras and E. Gamess, "An algorithm based on VANET technology to count vehicles stopped at a traffic light," *Int. J. Intell. Transp. Syst. Res.*, vol. 18, no. 1, pp. 122–139, May 2019.
[29] E. Shaghaghi, M. R. Jabbarpour, R. M. Noor, H. Yeo, and J. J. Jung, "Adaptive green traffic signal controlling using vehicular communication," *Frontiers Inf. Technol. Electron. Eng.*, vol. 18, no. 3, pp. 373–393, Mar. 2017.
[30] A. Ranjan, "VANET based four way road intersection traffic light control model," M.S. thesis, Dept. Comput. Sci. Eng., Nat. Inst. Technol. Rourkela, Rourkela, India, May 2015.
[31] H. Chen, C. Yang, and X. Xu, "Clustering vehicle temporal and spatial travel behavior using license plate recognition data," *J. Adv. Transp.*, vol. 2017, Apr. 2017, Art. no. 1738085.
[32] V. Praveen, K. Kousalya, and K. P. Kumar, "A nearest centroid classifier based clustering algorithm for solving vehicle routing problem," in *Proc. 2nd Int. Conf. Adv. Elect., Electron., Inf., Commun. Bio-Inform. (AEEICB)*, Chennai, India, Aug. 2016, pp. 414–419.
[33] *Family of Standards for Wireless Access in Vehicular Environments (WAVE)*, IEEE Standard 1609, U.S. Department of Transportation, Jan. 2006.
[34] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, *Global Positioning System: Theory and Practice*, 5th ed. New York, NY, USA: Springer-Verlag, 2004.
[35] F. J. Martinez, C. K. Toh, J.-C. Cano, C. T. Calafate, and P. Manzoni, "A survey and comparative study of simulators for vehicular ad hoc networks (VANETs)," *Wireless Commun. Mobile Comput.*, vol. 11, no. 7, pp. 813–828, Jul. 2011.
[36] D. M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "SUMO—Simulation of urban mobility: An overview," in *Proc. 3rd Int. Conf. Adv. Syst. Simulation (SIMUL)*, Barcelona, Spain, Oct. 2011, pp. 1–6.
[37] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proc. 1st Int. Conf. Simulation Tools Techn. Commun., Netw. Syst. (SIMUTools)*, Marseille, France, Mar. 2008, pp. 1–10.
[38] E. Gamess and M. Contreras, "A proposal for an algorithm to count nodes using wireless technologies," *Int. J. High Perform. Comput. Netw.*, vol. 8, no. 4, pp. 345–357, 2015.
[39] K. Wessel, M. Swigulski, A. Kopke, and D. Willkomm, "MiXiM: The physical layer an architecture overview," in *Proc. 2nd Int. Conf. Simulation Tools Techn. (SIMUTools)*, Rome, Italy, Mar. 2009, pp. 1–8.

MANUEL CONTRERAS received the M.S. degree in computer science from the University of Los Andes, Merida, Venezuela, in 2005. He is currently pursuing the Ph.D. degree in computer science with the Central University of Venezuela, Caracas, Venezuela. He is also a Professor with the University of Los Andes, Trujillo, Venezuela. His research interests include computer networks, software engineering, and computer science education.

ERIC GAMESS received the M.S. degree in industrial computation from the National Institute of Applied Sciences (INSA), Toulouse, France, in 1989, and the Ph.D. degree in computer science from the Central University of Venezuela, Caracas, Venezuela, in 2010. He was a Professor with the University of Puerto Rico, San Juan, Puerto Rico, the Central University of Venezuela, and the Universidad del Valle, Cali, Colombia. He is currently a Professor with Jacksonville State University, Jacksonville, AL, USA. His research interests include vehicular ad hoc networks, network performance evaluation, IPv6, network protocol specifications, and cybersecurity. He was on organizing and program committees of several national and international conferences. He is the Co-Founder of the Venezuelan Society of Computing and the Director of The Venezuelan Journal of computing.