LEADER ELECTION AND GROUP MANAGEMENT IN VEHICULAR AD HOC NETWORK

BY

YIXIAO NIE

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Adviser:

Professor Nitin Vaidya

ABSTRACT

As automobiles become more intelligent, research on the Vehicular Ad Hoc Network (VANET) also becomes more important. Leader election is an important piece of the puzzle that can be utilized to solve many other problems in VANET. However, most existing literatures either focus on Virtual Traffic Light (VTL) application or leader election in regular ad hoc networks. In this thesis, we focus on creating a generalized algorithm for leader election in VANET and designing a group management mechanism to address various scenarios. In addition, simulations are conducted to evaluate performance of proposed algorithms.

To Mother and Father

To My Love Bo

ACKNOWLEDGMENTS

This thesis would not have been possible without the support of many people. I would like to thank my advisor, Professor Nitin Vaidya, who offered great guidance throughout the entire process and provided excellent suggestions to clarify my confusion. Also thanks to Shegufta Ahsan, who generously provided help in solving difficulties I met during experiments. I would also like to show my gratitude to Janice Progen from ECE Editorial Services for her dedicated guidance in editing my thesis. In addition, I am also immensely grateful to the University of Illinois Graduate College and Electrical and Computer Engineering Department for offering me a teaching assistant position and for providing me with financial aid to complete the degree in Master of Science. Finally yet importantly, I would like to thank my parents, girlfriend, and numerous friends, without whom it would be impossible for me to reach my goals this far.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION AND BACKGROUND

The contents in this thesis are organized in the following structure. In chapter 1, we introduce important concepts and a concrete problem statement. In chapter 2, we review existing literature on similar topics in the past decades. In chapter 3, we propose multiple versions of Leader Election Protocols as well as a comprehensive mechanism for group management. In chapter 4, we simulate the protocols and evaluate the results. In chapter 5, we provide discussion on the tradeoffs and gains for our algorithm and offer several suggestions for optimization. In chapter 6, we draw conclusions and list directions of future work.

1.1 Vehicular Ad Hoc Network

The Vehicular Ad Hoc Network (VANET) [1] is a special case of the Mobile Ad Hoc Network that specifically applies to communication between vehicles. To be more specific, in this thesis, we study VANET as a type of decentralized wireless network, through which the vehicles can directly communicate with each other without relying on routers or access points. Key features in VANET include high mobility, which may result in frequent changes in network topology. The mobility of vehicles can be somewhat more predictable than general mobile ad hoc networks.

1.2 Leader Election

Leader election, in VANET, is the process of cooperatively selecting a unique vehicle to organize a group of vehicles. This process is non-trivial, since there are many issues that may result in failure. In most cases, leader election is a particular type of consensus problem, which is theoretically proven to be impossible to solve in an asynchronous system [2] or under unbounded message losses [3], [4], [5] in process or node failure. However, in real-world applications, it is still beneficial to design a Leader Election Protocol with high accuracy and low latency.

1.3 Virtual Traffic Light

The Virtual Traffic Light (VTL) [6] is a new way of managing traffic lights at road intersections. Traditionally, physical traffic lights are used for controlling the traffic signals, which is less efficient and consumes more energy. With the help of VANET and leader election, we are able to create a VTL among vehicles, which is controlled by the leader. Such optimization eliminates the need of physical traffic lights and allows the vehicles to change the traffic signals intelligently.

1.4 Problem Statement

1.4.1 Problem Setting

Each vehicle has reliable GPS equipment that can be used to determine its location and velocity.

Each vehicle has an accurate clock that is synchronized with other vehicles. Clock skew and drift will not be considered in our study.

IEEE 802.11 is used as the communication specifications between vehicles, which may be lossy under heavy load or through long distance.

Each vehicle maintains an accurate list of neighbors, updated through BEACON messages. This is achieved by periodically broadcasting BEACON messages. In addition, the location and velocity of each vehicle are sent through BEACON messages and stored in the list of neighbors.

All experiments in this study are conducted in computer simulation.

1.4.2 Problem Definition

Our goal is to design, build, and test an algorithm for efficient fault-tolerant leader election and group management in VANET, which later can be applied to applications such as VTL.

CHAPTER 2
RELATED WORK

There exist works related to the problem we studied. Most of them focus on two different aspects: VTL applications and leader election.

One of the first proposals of an intelligent traffic light using the communication between vehicles and fixed controller nodes in the intersection was presented in [6]. However, as it still requires the presence of a physical traffic light, it may not be considered as a VTL application.

Ferreira et al. [7] presented a more complete VTL protocol to replace the roadside-based infrastructures. When vehicles are approaching the intersection, they cooperatively decide one vehicle to be the leader based on two conditions: (i) the vehicle should be stopped at a red light; (ii) the vehicle should be the closest one to the intersection. After a vehicle is elected as the leader, it will be responsible for controlling the virtual traffic light subsequently. When green traffic light is in the leader’s lane, it will pass the intersection and hand over the leadership to one of the vehicles stopping before the red light. However, [7] did not mention any details of leader election protocol nor the protocol for leader handover. In addition, it is assumed that the reliability and latency of wireless communication is adequate for the VTL protocol.

A prototype for a VTL application using Android-based smartphones was presented in [8]. The algorithm they implemented is based on the self-organized traffic control paradigm presented in [7]. Yet details of the algorithms used are not presented.

In [9], a fail-safe protocol for the VTL application was presented, and it was verified using the PROMELA modeling language and SPIN model checker [10]. In their protocol, a vehicle that is closest to the intersection is first chosen as the Cluster Leader of that road segment by using reliable line-of-sight communication. Then one of the up to four Cluster Leaders is chosen as the VTL Leader to control the traffic light. When the VTL Leader leaves the intersection, a handover is performed to pass the leadership to another vehicle. However, not enough protocol details are provided for the implementation of the leader election or handover.

Past work has shown that it is impossible to achieve guaranteed consensus in both synchronous system and asynchronous system if the number of message losses is not bounded [2], [3], [4], [5]. Therefore, it is unrealistic to discuss leader election protocol in VANETs without a discussion of reliability, i.e., the probability for a protocol to achieve consensus under different scenarios.

A thorough study of a family of simple round-based consensus algorithms for the VTL leader election protocol was performed in [11]. It shows the probability of disagreement for a given algorithm depends on three parameters: (i) the number of vehicles in the group; (ii) the number of rounds performed in the algorithm; and (iii) the probability of message loss. However, for the third parameter, they assumed a fixed probability of message loss q for all messages, which may not be ideal in VANET settings. The q should depend on the distance between vehicles and the probability of other nodes transmitting a packet in the same slotted time (transmission collision).

One of the few papers that included a thorough explanation for the leader election algorithm and considered packet loss is [12]. The VTL algorithm used in [12] is adopted from [13] for dynamic ad hoc networks. They simulated the algorithm on Vein, a simulation framework that combines the network simulator OMNeT++ and the traffic simulator SUMO. Results show that VTL is able to reduce travel times by up to 35%. However, when the intersection topologies become increasingly complex, they noticed that the case when more than one leader is elected may increase to as high as 23%, and they have to replace the VTL leader election with a perfect Oracle leader election. In addition, they discovered that as the density of traffic grows, packet loss increases as well, mostly due to collisions.

On the other hand, [13] presented a leader election algorithm that can adapt to any topological changes in mobile ad hoc networks. Three phases are used in the protocol: (i) the source node sends out an Election message to all neighbors. Other nodes, upon receiving the first Election message, will designate the sender as parent and propagate the Election message; (ii) when a node i receives an Election message from a node j that is not its parent, an Ack message will be responded to j immediately. An Ack message to parent node will only be sent once all remaining neighbors responded with an Ack message; and (iii) once the source node receives Ack messages from all the children, it will broadcast a Leader message to all nodes. Essentially, this algorithm completes the leader election by growing and shrinking a spanning tree. However, the protocol requires that communication between nodes takes place using reliable transport protocol, so that the sender knows whether the packet has been received by the destination node or not. Therefore, it is uncertain how it would behave under a massive packet loss rate.

CHAPTER 3

LEADER ELECTION AND GROUP MANAGEMENT

3.1 Leader Election Protocol

In this section, we demonstrate the process of developing a new Leader Election Protocol. To start, we build a first version from scratch. Then we adopt an idea from similar work to create a second version. Last, we make improvements on the strategy to reach a third and final version.

To help better understand the protocols, a variable list for each vehicle is provided in Table 1.

Table 1: A Variable Table for Each Vehicle

Variable	Explanation
In_election	A flag indicating if the vehicle is in election or not
Vid	A variable set to the vehicle's id at initialization
Leader	A variable storing the leader's id
LeaderList	A container for all leaders' id
Proposal	A variable storing the value of best proposal value collected so far
ProposalId	A variable storing the vehicle id associated with the value in Proposal
ProposalList	A container for all proposals
NeighborList	A container for all neighbors' information, updated upon receiving the BEACON message
AckList	A container for storing the acknowledgements from neighbors
Parent	A variable storing the parent's id when a spanning tree is constructed, -1 for root
ChilrenList	A container for storing children's ids when a spanning tree is constructed

3.1.1 First Version of Leader Election Protocol

When a vehicle enters a new area, it will first check for the existence of a leader. If the leader exists, the vehicle joins as a member and follows the instructions from the leader. Otherwise, the vehicle starts the Algorithm 1 to elect a leader.

In initialization, calculate_proposal() is a function to be defined. For simplicity, it is defined to return Vid in the simulation. To achieve better performance, we can define it to return an estimated time of how long it may stay within a given region, using the location and velocity information provided by a GPS device.

In all algorithms, words with all capital letters represent messages, and the attributes associated with messages are placed in double quotes “ ”. Broadcasts are all sent to the MAC broadcast address (FF:FF:FF:FF:FF:FF). $T_{timeout}$ is set to 1 s in the simulation. T_{rerun} is set to 0.2 s in the simulation.

```
Initialization
In_election=false, Leader=-1, Vid=get_self_id(), Proposal=calculate_proposal(),
ProposalList=<>, AckList=<>.

On entering new region:
    If In_election = false and Leader = -1
        Set In_election=true
        Broadcast ELECTION(“Sender”=Vid, “Proposal”=Proposal)
        Set a timeout after $T_{timeout}$ seconds

On receiving ELECTION message m:
    If In_election = false
        Set In_election=true
        broadcast ELECTION(“Sender”=Vid, “Proposal”=Proposal)
        Set a timeout after $T_{timeout}$ seconds
    If m.Sender is not in AckList
        Add m.Sender to AckList; add m.Proposal and m.Sender to ProposalList;
        broadcast ELECTION(“Sender”=m.Sender, “Proposal”=m.Proposal)

On timeout:
    Set Leader = vehicle with max proposal in ProposalList
    Broadcast ANNOUNCE(“Leader”=Leader)

On receiving ANNOUNCE message m:
    If m.Leader $\neq$ Leader
        Broadcast ABORT

On receiving ABORT message m:
    Stop leader election; propagate the ABORT message;
    Schedule a new round of election to run after $T_{rerun}$ seconds if self was initiator
```

Algorithm 1: First Version of Leader Election Protocol

On success and with no message loss, the total number of messages broadcasted should be (n+1) * n for a group of n vehicles. This is because each node will broadcast 1 ELECTION message of its own, rebroadcast n-1 ELECTION messages from the other nodes, and broadcast 1 ANNOUNCE message.

3.1.2 Second Version of Leader Election Protocol

Continuing from the first version, we adopt the idea of creating a spanning tree from [13] to implement a second version of Leader Election Protocol as shown in Algorithm 2. This implementation will reduce the number of messages rebroadcasted due to the use of a spanning tree.

In addition, we make the following assumption about participating nodes. Each vehicle is periodically broadcasting a reliable BEACON message (see Section 1.4.1) with its own information (location, speed). Once a BEACON message is received, the recipient will denote the sender as a neighbor and store it in a list of neighbors. If no BEACON message is received from a neighbor within k periods (default set to k = 1 in simulation), that vehicle will be removed from the list of neighbors.

```
Initialization
In_election=false, Leader=-1, Parent=-1, Vid=get_self_id(), AckList=<>,
NeighborList=<all neighbors>, Proposal=calculate_proposal(), ProposalId=Vid.

On entering new region:
    If In_election = false and Leader = -1
        Set In_election=true
        Broadcast ELECTION("Sender"=Vid)
        Set a timeout after $T_{timeout}$ seconds

On receiving ELECTION message m:
    If In_election = false
        Set In_election=true and Parent=m.Sender
        Broadcast ELECTION("Sender"=Vid)
        Set a timeout after $T_{timeout}$ seconds
    Else
        Broadcast ELECTIONACK("Sender"=Vid)
```

Algorithm 2: Second Version of Leader Election Protocol

```
On receiving ELECTIONACK message m:
    Add m.Sender to AckList
    If AckList = NeighborList
        Broadcast PROPOSAL("Sender"=Vid, "Receiver"=Parent,
"Proposal"=Proposal, "ProposalId"=ProposalId)

On receiving PROPOSAL message m:
    Add m.Sender to AckList
    If m.Proposal > Proposal
        Set Proposal= m.Proposal and ProposalId = m.ProposalId
    If AckList = NeighborList
        If Parent ≠ -1
            Broadcast PROPOSAL("Sender"=Vid, "Receiver"=Parent,
"Proposal"=Proposal)
        Else (NOTE: this is the case for initiator)
            Set Leader = ProposalId; cancel timeout
            Broadcast LEADER("Leader"= Leader)

On receiving LEADER message m:
    Set Leader = m.Leader and propagate the LEADER message
    Cancel timeout

On receiving ABORT message m:
    Stop leader election; propagate the ABORT message;
    Schedule a new round of election to run after $T_{rerun}$ seconds if self was initiator

On timeout:
    Broadcast ABORT and stop leader election
```

Algorithm 2 (cont.)

On success and with no message loss, the total number of messages broadcasted should be (2+avg(deg(v))) * n for a group of n vehicles. This is because each node v will broadcast 1 ELECTION message, 1 PROPOSAL message, 1 LEADER message, and deg(v)-1 ELECTIONACK messages (no ELECTIONACK message is sent to the parent, replaced by a PROPOSAL message).

3.1.3 Third Version of Leader Election Protocol

To further reduce the message overhead in the second version, we take advantage of the overhearing property in wireless network and remove ELECTIONACK messages from the protocol. To make up for the lack of the ELECTIONACK message, a Parent attribute is added to the ELECTION message. In Algorithm 3 we present the third and final version of our Leader Election Protocol.

```
Initialization
In_election=false, Leader=-1, Parent=-1, Vid=get_self_id(),
Proposal=calculate_proposal(), ProposalId=Vid, AckList=<>, ChildrenList=<>
NeighborList=<all neighbors>.

On entering new region:
    If In_election = false and Leader = -1
        Set In_election=true
        Broadcast ELECTION("Sender"=Vid, "Parent"=-1, "ProposalId" =
ProposalId)
        Set a timeout after $T_{timeout}$ seconds

On receiving ELECTION message m:
    Add m.Sender to AckList
    If In_election = false
        Set In_election=true and Parent=m.Sender
        Broadcast ELECTION("Sender"=Vid, "Parent"=Parent)
        Set a timeout after $T_{timeout}$ seconds
    If m.Parent = Vid
        Add m.Parent to ChildrenList
    If AckList = NeighborList and ChildrenList is empty
        Broadcast PROPOSAL("Sender"=Vid, "Proposal"=Proposal, "ProposalId" =
ProposalId)

On receiving PROPOSAL message m:
    If m.Sender is not in ChildrenList Then return
    If m.Proposal > Proposal
        Set Proposal= m.Proposal and ProposalId = m.ProposalId
    Remove m.Sender from ChildrenList
    If AckList = NeighborList and ChildrenList is empty
        If Parent $\neq$ -1
            Broadcast PROPOSAL("Sender"=Vid, "Proposal"=Proposal,
"ProposalId" = ProposalId)
        Else (NOTE: this is the case for initiator)
            Set Leader = ProposalId; cancel timeout
            Broadcast LEADER("Leader"= Leader)
```

Algorithm 3: Third Version of Leader Election Protocol

```
On receiving LEADER message m:
    Set Leader = m.Leader and propagate the LEADER message
    Cancel timeout

On receiving ABORT message m:
    Stop leader election; propagate the ABORT message;
    Schedule a new round of election to run after $T_{rerun}$ seconds if self was initiator

On timeout:
    Broadcast ABORT() and stop leader election
```

Algorithm 3 (cont.)

On success and with no message loss, the total number of messages broadcasted should be 3n for a group of n vehicles. This is because each node will broadcast 1 ELECTION message, 1 PROPOSAL message, and 1 LEADER message.

Figure 1 is a flowchart for the third version of Leader Election Protocol.

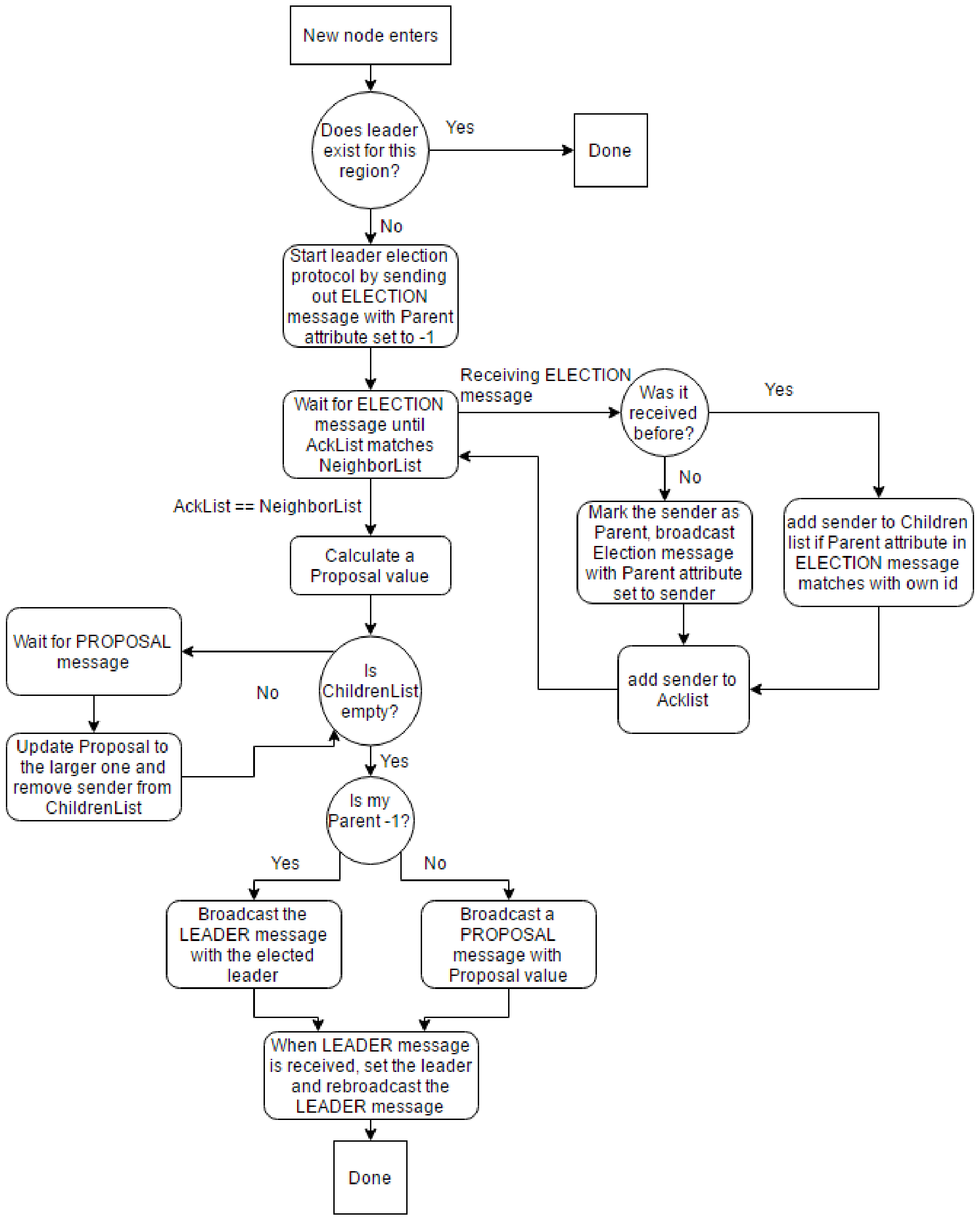
3.2 Failure Handling

In both reality and the simulation, leader election may fail because of many different reasons (e.g. co-channel interference, topology changes). Therefore, this section discusses a couple of common failures and the methods to handle them during leader election.

3.2.1 Group Overextension

In dense network topology, such as downtown Manhattan, the size of a group may become extremely large because the distances between vehicles are usually very small. Therefore, if no restriction is enforced on the vehicles, a single leader election may involve hundreds of cars, which is both unnecessary and unrealistic.

In order to avoid such situations, we allow the initiator of leader election to apply a restriction by specifying the vehicles allowed to participate. For example, in our



New node enters
Does leader exist for this region?
Yes
Done
No
Start leader election protocol by sending out ELECTION message with Parent attribute set to -1
Wait for ELECTION message until AckList matches NeighborList
Receiving ELECTION message
Was it received before?
Yes
No
Mark the sender as Parent, broadcast Election message with Parent attribute set to sender
add sender to Children list if Parent attribute in ELECTION message matches with own id
add sender to Acklist
AckList == NeighborList
Calculate a Proposal value
Wait for PROPOSAL message
No
Is ChildrenList empty?
Update Proposal to the larger one and remove sender from ChildrenList
Yes
Is my Parent -1?
Yes
No
Broadcast the LEADER message with the elected leader
Broadcast a PROPOSAL message with Proposal value
When LEADER message is received, set the leader and rebroadcast the LEADER message
Done

Figure 1: Flowchart for the Third Version of Leader Election Protocol

simulation, this restriction is usually specified as an n * n square centered at an intersection.

3.2.2 Rebroadcast Collision

When a vehicle broadcasts a message, all neighbors of the vehicle will be able to receive it and rebroadcast. However, if rebroadcasts all happen immediately after receipt of the first message, collisions may happen due to co-channel interference.

In order to reduce the collision rate, we adopt a strategy similar to the counter-based scheme in [14], by using the following formula to insert a random waiting time before rebroadcast:

$$wait\ \ T_{random} = (random()\%C_n) * T_{slot}$$

where T_{slot} is the slot size in microsecond, and C_n is the maximum number of slots a rebroadcast can wait for. In the experiments, parameters are chosen such that $T_{slot} = 10$ µs and $C_n = 1000$.

While introducing randomness in the rebroadcast process is very effective in reducing collisions, it does not guarantee to eliminate all collisions. We will discuss the solution for this issue in the next section.

3.2.3 Message Loss and Requesting Resend Mechanism

Message loss is the most common error in leader election, and it may happen due to unavoidable reasons such as interference and hardware malfunction. In order to recover from message loss and continue execution of the Leader Election Protocol, we implemented a Requesting Resend Mechanism.

After the leader election starts, all vehicles will periodically check the status of its own AckList and ChildrenList. For any vehicle, if a certain message (ELECTION or PROPOSAL) is not received, a REQUESTRESEND message will be transmitted to the corresponding vehicle to request for retransmission.

```
If AckList $\neq$NeighborList
    For each vehicle i in NeighborList but not in AckList
        Broadcast REQUESTRESEND(“Sender”=Vid, “Receiver”=i)
Else if ChilrenList is not empty:
    For each vehicle i in ChildrenList:
        Broadcast REQUESTRESEND(“Sender”=Vid, “Receiver”=i)
```

Algorithm 4: Requesting Resend Mechanism

Upon receiving a REQUESTRESEND message, the target vehicle will resend the missing message.

3.2.4 Vehicle Failure and Topology Change

In rare situations, vehicles may fail without notifying their neighbors, or topology changes may occur in the short time span of leader election. To address such issues, in Requesting Resend Mechanism, a neighbor vehicle is labeled as unreachable after n (chosen as n = 3 for the experiment) consecutive failed attempts and is removed from the neighbor list after the current round of the leader election aborts. As a result, in the next round of leader election, we will have an updated and accurate network topology.

3.2.5 Other Failures

In this thesis, our protocol does not address the following failures:

GPS Error, Unidirectional Link Failure. Equipping vehicles with high-end GPS and network devices may resolve this issue.

BEACON Message Loss. The loss of a BEACON message may result in an inaccurate neighbor list, which will significantly affect the efficiency and accuracy of the Leader Election Protocol. However, we could use other technologies in conjunction to create a complete and accurate neighbor list. For example, communication through a cellular network or wireless LAN access point may render higher accuracy, with the tradeoff of larger latency.

3.3 Group Maintenance, Joining, and Leaving

3.3.1 Group Maintenance

In order to keep track of the status of the group, the leader will periodically broadcast a Leader Heartbeat with a timestamp to notify the neighbors (heartbeat interval is set to 1 s in simulation). On receiving a Leader Heartbeat, each group member will update the local timestamp accordingly and rebroadcast to propagate the message to others (if it has not already done so).

An example is provided in Figure 2. In (a), when the simulation reaches 2 s (from the beginning of simulation), leader A updates its local timestamp to 2 s and broadcasts a Leader Heartbeat with timestamp 2 s. In (b), after receiving a Leader Heartbeat from A, B updates its local timestamp (associated with leader A) to 2 s and rebroadcasts. In (c), though C missed the last Leader Heartbeat message at 1 s and has local timestamp 0 s, upon receiving the new Leader Heartbeat, it updates its local timestamp to 2 s and rebroadcasts. In (d), after receiving the Leader Heartbeat, D updates its local timestamp to 2 s and rebroadcasts.

The Leader Heartbeat message serves two purposes. First, it notifies a new node entering the region the existence of the leader. Second, it allows group members to keep track of the leader and remove an inactive leader (explained in more detail in Section 3.3.3).

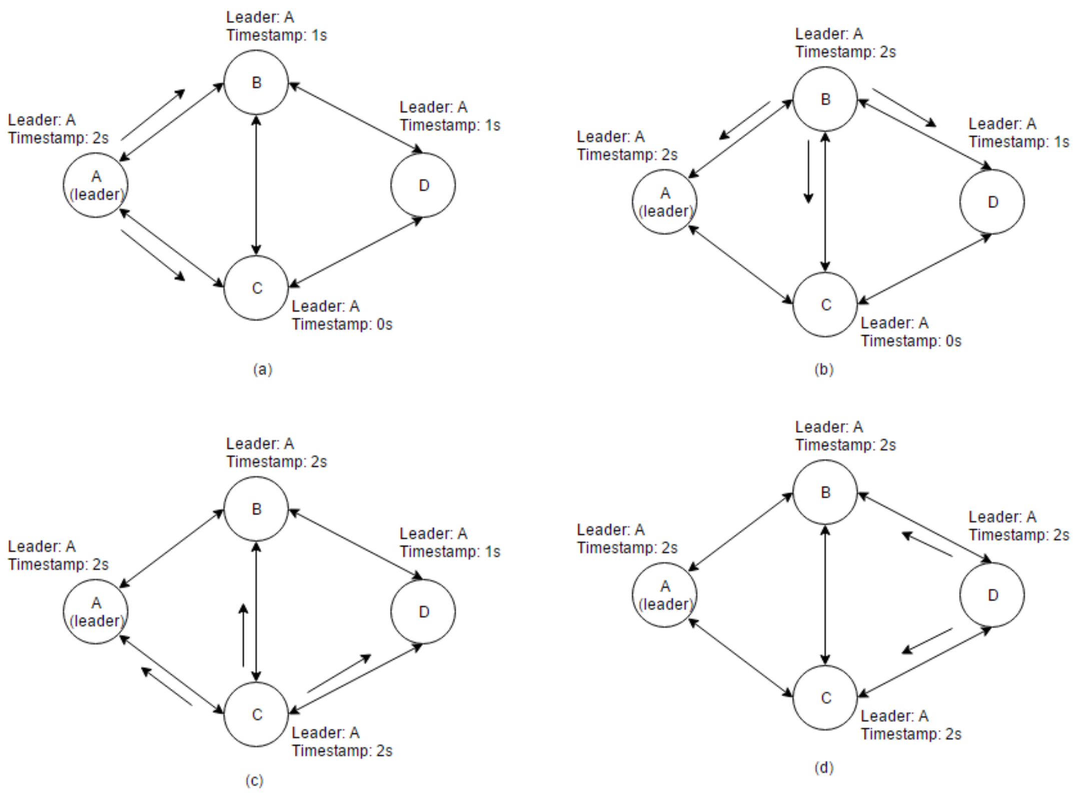


Figure 2: Group Maintenance

3.3.2 Group Joining

When a new vehicle enters a region, it will first listen for a Leader Heartbeat message. If such a message is received, the new vehicle marks the corresponding vehicle as the leader and considers itself as having joined the group. If no such message is received within a period of time, the new vehicle may start the Leader Election Protocol as described in Section 3.1.3.

3.3.3 Group Leaving

If a vehicle does not receive any Leader Heartbeat message after a certain period of time (default is set to three heartbeat intervals), it will automatically consider itself as removed from the group and clear the information about the leader. If in the future a Leader Heartbeat message is received again, the vehicle rejoins the group.

3.4 Conflict Resolving and LAMP

When two or more different Leader Heartbeats are received by a vehicle, we define such situation as a conflict. Conflicts may happen due to groups merging, temporary partition and recovery, etc. Our protocol handles conflict using a schema called Leader Active Member Passive (LAMP).

LAMP requires the leader to actively resolve conflicts while members passively wait for updates. For example, if a member vehicle receives a Leader Heartbeat from a different leader, it will simply forward it to other members inside the group. However, if a leader receives a Leader Heartbeat from a different leader, it will pick a winner from the two leaders using predefined rules (such as the vehicle with a larger id). If the winner is itself, nothing will happen for this leader; if the winner is the other vehicle, a Leader Resignation message will be broadcasted to inform members about the change of leadership.

In addition to the rules already mentioned, each group member also keeps track of the Leader Heartbeat messages. For a particular leader, if more than three consecutive Leader Heartbeat messages are missing, the group member will remove that leader from the leader position.

A detailed example of group merging will be discussed in Section 3.7.2.

3.5 Random Leader Election Protocol

In the Leader Election Protocols described in Section 3.1, each vehicle is required to make a proposal with a value it selects. The initiator will collect all the proposals and declare the vehicle with the largest proposal value as the leader. However, there are situations where we just want a random vehicle to be the leader. Therefore, we propose the following Random Leader Election Protocol (Algorithm 5) to allow a random vehicle to become the leader.

```
Initialization
In_election=false, Leader=-1, Vid=get_self_id(), AckList=<>, NeighborList=<all neighbors>

On entering new region:
    If In_election = false and Leader = -1
        Set In_election=true and Leader = Vid
        Broadcast LEADER("Sender"=Vid, "Leader"=Vid)
        Set a timeout after $T_{timeout}$ seconds

On receiving LEADER message m:
    If In_election = false
        Set In_election=true and Leader=m.Leader
        Broadcast LEADER("Sender"=Vid, "Leader"= Leader)
        Set a timeout after $T_{timeout}$ seconds
    Add m.Sender to AckList
    If AckList = NeighborList
        Cancel timeout

On timeout:
    Set Leader = -1; broadcast ABORT() and stop random leader election
```

Algorithm 5: Random Leader Election Protocol

On success and with no message loss, the total number of messages broadcasted should be n for a group of n vehicles. This is because each node will only broadcast one LEADER message.

Figure 3 is a flowchart for the Random Leader Election Protocol.

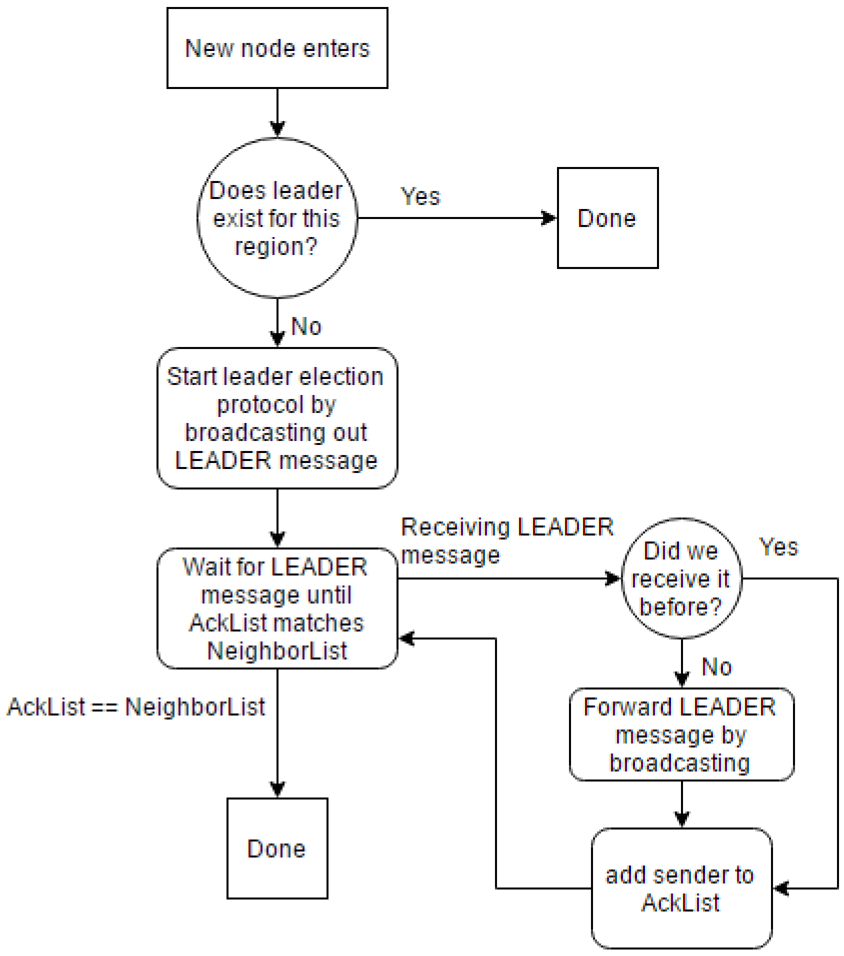

New node enters
Does leader exist for this region?
Yes
Done
No
Start leader election protocol by broadcasting out LEADER message
Receiving LEADER message
Wait for LEADER message until AckList matches NeighborList
Did we receive it before?
Yes
No
Forward LEADER message by broadcasting
AckList == NeighborList
Done
add sender to AckList

Figure 3: Flowchart for Random Leader Election Protocol

3.6 Leader Handover Protocol

In traditional networks, once a leader is elected, it seldom leaves the group. However, in VANET, the vehicle designated as the leader constantly moves from one region to another, which results in unnecessary and inefficient reelections.

Fortunately, one unique feature of VANET is that the vehicles have access to their locations and velocities via GPS devices. Thus, as one optimization to avoid performing a new round of leader election, the leader may start a Leader Handover when it is about to leave its region.

In the following introduction of two protocols, we assume the leader has already decided which vehicle should be the new leader. Otherwise, the old leader can simply start a new leader election by using the Leader Election Protocol (third version).

3.6.1 Leader Handover Protocol Derived from the Random Leader Election Protocol

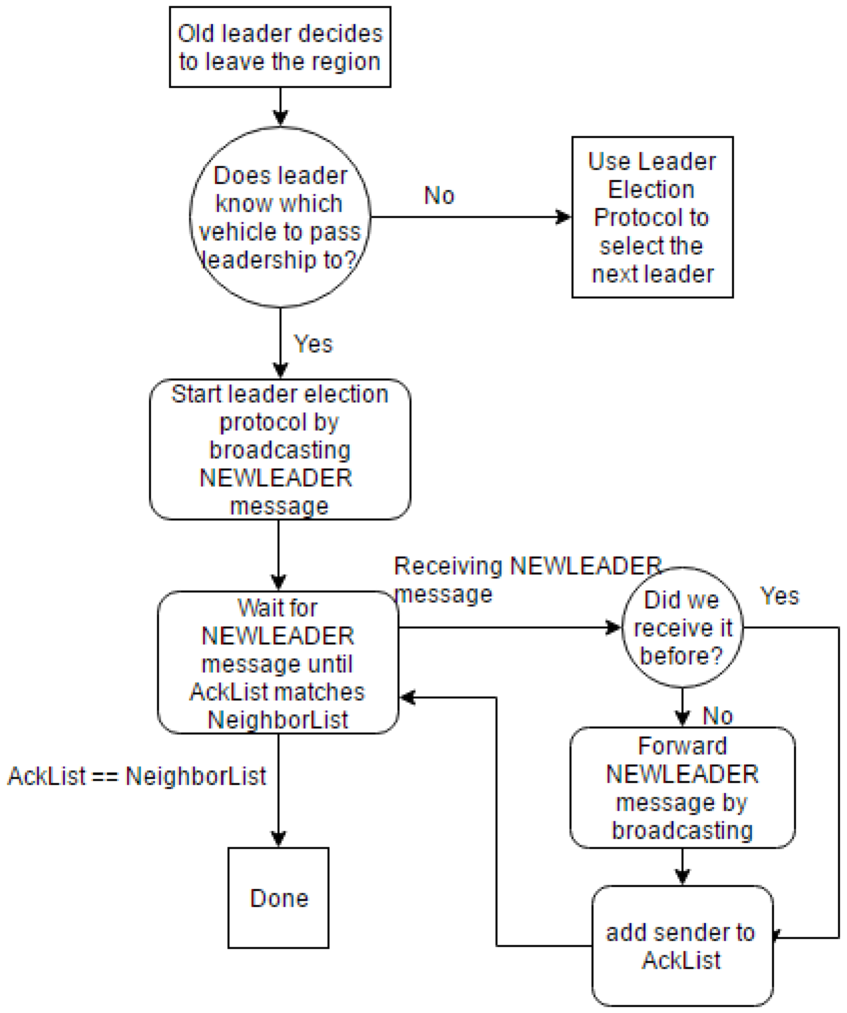
Assuming the leader has already decided which vehicle should be the new leader, we can adopt the Random Leader Election Protocol to perform the Leader Handover, as shown in Algorithm 6.

```
Initialization
In_election=false, Vid=get_self_id(), AckList=<>, NeighborList=<all neighbors>

For original leader:
Before leaving region:
    Set Leader = calculate_new_leader()
    Broadcast NEWLEADER("Sender"=Vid, "Leader"= Leader)
    Set a timeout after $T_{timeout}$ seconds

For all vehicles:
On receiving NEWLEADER message m:
    If In_election = false
        Set In_election=true and Leader=m.Leader
        Broadcast NEWLEADER("Sender"=Vid, "Leader"= Leader)
        Set a timeout after $T_{timeout}$ seconds
    Add m.Sender to AckList
    If AckList = NeighborList
        Cancel timeout

On timeout:
    Restore Leader to old leader; broadcast ABORT() and stop leader handover
```

Algorithm 6: Leader Handover Protocol from the Random Leader Election Protocol

The advantage of using this Leader Handover Protocol is that it requires acknowledgment of all neighbors in order to complete the Leader Handover process. Therefore, it will ensure at most one leader exists at all times.

Figure 4 is a flowchart for the Leader Handover Protocol.



Old leader decides to leave the region
Does leader know which vehicle to pass leadership to?
No
Use Leader Election Protocol to select the next leader
Yes
Start leader election protocol by broadcasting NEWLEADER message
Receiving NEWLEADER message
Wait for NEWLEADER message until AckList matches NeighborList
Did we receive it before?
Yes
No
Forward NEWLEADER message by broadcasting
AckList == NeighborList
Done
add sender to AckList

Figure 4: Flowchart for Leader Handover Protocol from Random Leader Election Protocol

3.6.2 Leader Handover Protocol by LAMP

In addition to the previous Leader Handover Protocol, we leverage the nice property of LAMP to implement a much easier protocol, on the condition that the user does not care if two leaders exist simultaneously for a short period of time.

The advantage of this Leader Handover Protocol is that it is easy to implement, as this protocol is built on top of the LAMP property. Once the new leader starts broadcasting Leader Heartbeat messages, the old leader will resign and consider the handover finished. Even if the Leader Resignation message does not reach all members, they will passively drop the old leader after three consecutive misses of Leader Heartbeats.

```
Initialization
In_election=false, LeaderList=<Leader>, Vid=get_self_id(), AckList=<>,
NeighborList=<all neighbors>

For original leader:
Before leaving region:
    Set Leader = calculate_new_leader()
    Broadcast NEWLEADER("Sender"=Vid, "Leader"= Leader)
    Stop propagating LEADERHEARTBEAT message

For all vehicles:
On receiving NEWLEADER message m:
    If m.Leader = Vid
        Set Leader = Vid; start propagating LEADERHEARTBEAT("Leader"=Vid)

On receiving LEADERHEARTBEAT message m:
    Propagate the LEADERHEARTBEAT message received
    If m.Leader = Newleader and self is the original leader
        Broadcast LEADERRESIGNATION("Leader"=Vid)

On receiving LEADERRESIGNATION message m:
    Propagate the LEADERRESIGNATION message received
    Remove m.Leader from LeaderList
    If size(LeaderList) = 1
        Leader = LeaderList[0]
```

Algorithm 7: Leader Handover Protocol by LAMP

3.7 Group Partition and Merging

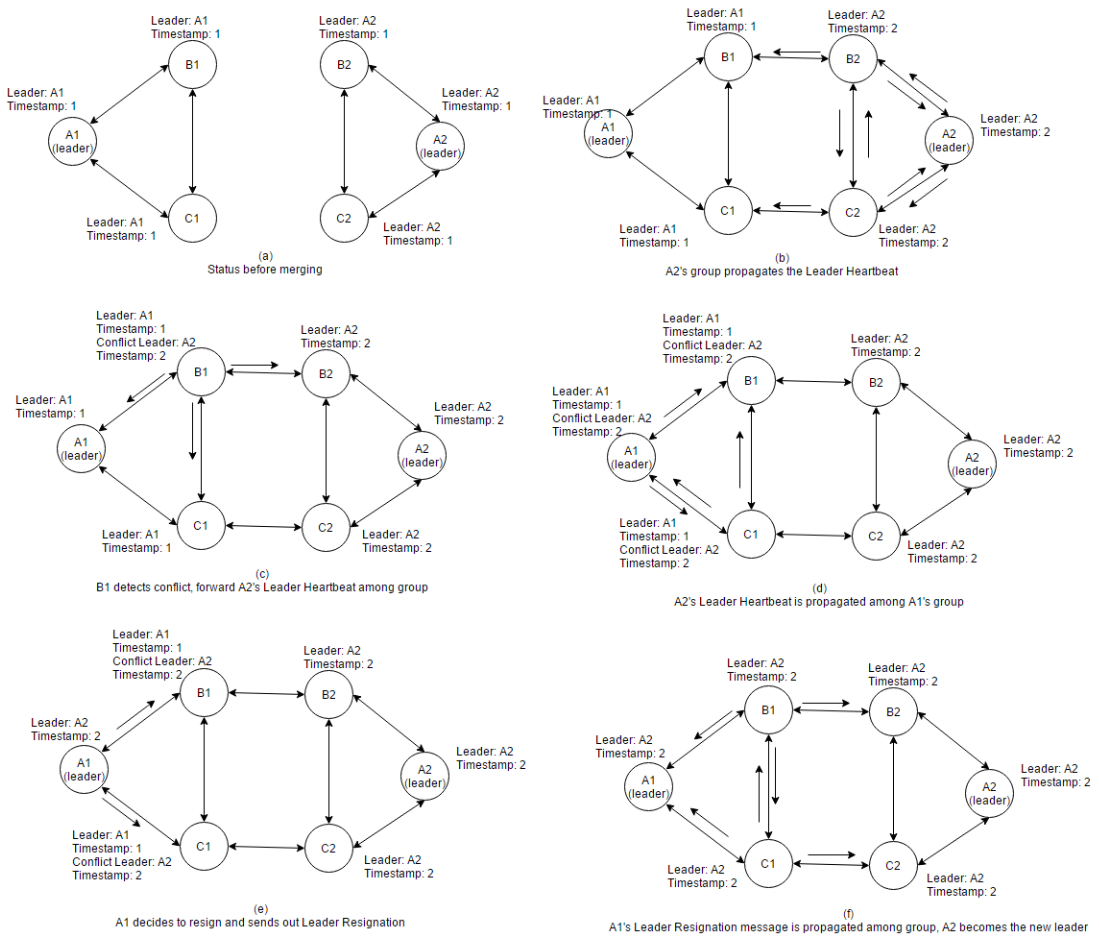
3.7.1 Group Partition

Upon a group partition, vehicles in the partition without the leader will eventually consider themselves as removed from the group because the Leader Heartbeat can no longer reach them. After a certain period of time (default is set to three heartbeat intervals in simulation), one of the vehicles will start the Leader Election Protocol to select a new leader.

3.7.2 Group Merging

Group merging is one of the most complicated scenarios in group management. In our protocol, group merging is done through active merging using LAMP. For example, when members of one group receive a Leader Heartbeat from the other leader, they will forward a Leader Heartbeat to the rest of the group. If the original leader receives the message and decides to resign leadership, it will both stop sending its own Leader Heartbeat and send out a Leader Resignation message. After receiving a Leader Resignation message, members from the old group will set the other leader as the new leader. Occasionally, the Leader Resignation message may get lost due to collision. In such cases, the resigned leader will not be removed immediately. Instead, all member vehicles will passively remove the old leader after three consecutive Leader Heartbeat messages are missing.

An example is illustrated in Figure 5. In (a), before merging, group {A1, B1, C1} has leader A1 and group {A2, B2, C2} has leader A2. In (b), two groups merge and A2 happens to send out a Leader Heartbeat first. In (c), B1 receives A2's Leader Heartbeat, adds A2 to the leader list, and forwards A2's Leader Heartbeat to the group. In (d), every vehicle in the original A1's group receives A2's Leader Heartbeat. In (e), because A2 has a higher vehicle id, A1 decides to resign and broadcasts a Leader Resignation message. In (f), every vehicle in the original A1's group receives A1's Leader Resignation message and removes A1 from the leader list. Now A2 becomes the new leader.



(a)
Status before merging
(b)
A2's group propagates the Leader Heartbeat
(c)
B1 detects conflict, forward A2's Leader Heartbeat among group
(d)
A2's Leader Heartbeat is propagated among A1's group
(e)
A1 decides to resign and sends out Leader Resignation
(f)
A1's Leader Resignation message is propagated among group, A2 becomes the new leader
A1 (leader)
A2 (leader)
B1
B2
C1
C2
Leader: A1
Timestamp: 1
Leader: A2
Timestamp: 1
Leader: A2
Timestamp: 2
Conflict Leader: A2
Timestamp: 2

Figure 5: Group Merging

CHAPTER 4

SI MULATION AND RESULTS

4.1 Simulation Environment

Our tests are conducted on a platform called Vehicular Networks Simulator (VNS) [15], a high-performance simulation framework that combines mobility models and network models together.

For the mobility model, our test uses the Manhattan mobility model as shown in Figure 6, which is a grid road topology that consists of vertical and horizontal streets. During the simulation, vehicles will appear from the start of each road under Poisson distribution, using the following formula for the interval:

$$\mathrm{T}_{\mathrm{Interval}} = T_{mean} * \left(-\log\left(\mathrm{random}(0.0001,1.0)\right)\right)$$

where T_{mean} is set to 2 seconds in our simulation.

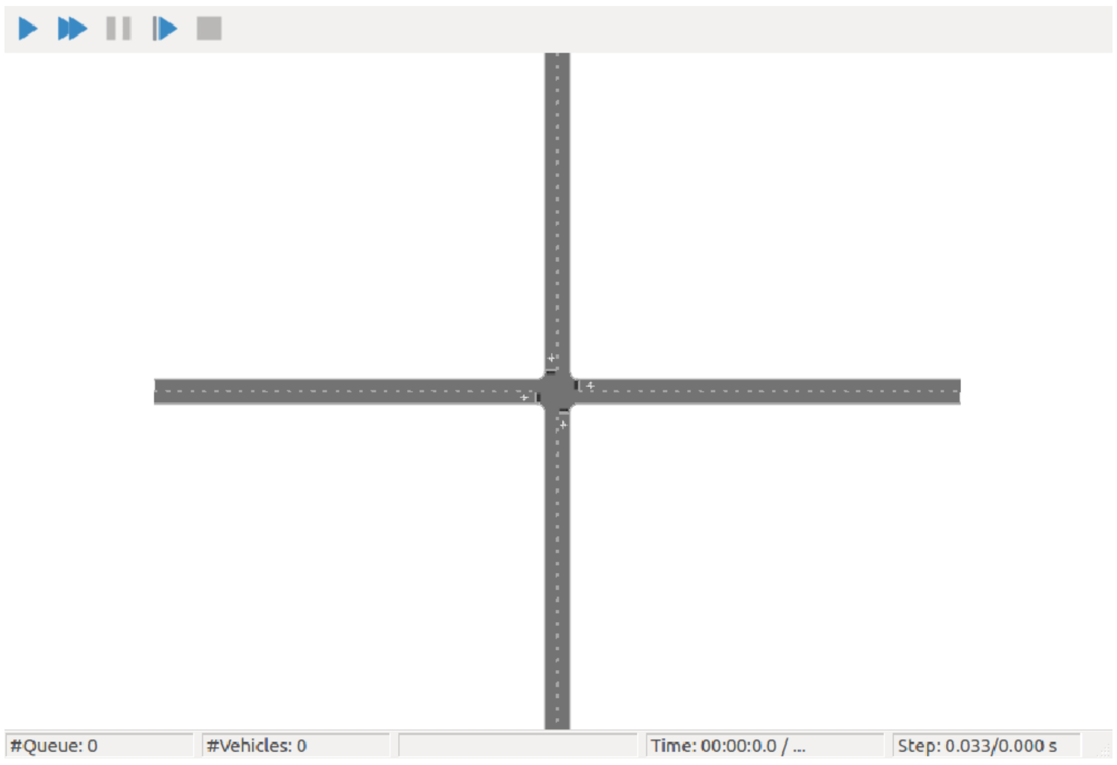


Figure 6: Manhattan Mobility Model

For the network model, our test utilizes NS-3 [16], a discrete-event network simulator. We utilize IEEE 802.11 as the communication protocol for lower layers, since it is the most popular protocol in VANET.

4.2 Leader Election Protocol Evaluation

In this section, we evaluate the accuracy and efficiency of the third version of Leader Election Protocol proposed. To simplify the testing environment, we run our simulation with a 1-by-1 Manhattan map, i.e., one vertical street and one horizontal street. The size of map is 100 m * 100 m, and a periodic traffic light is installed at the intersection (traffic signals set to 25 s for both green and red).

In this test, group management properties, such as Leader Heartbeat, are not enabled. Therefore, vehicles recently entering the map will not be able to detect the existence of a leader and will always initiate a new leader election. We run each experiment for 60 seconds and record important information. The result of an example run is given in Table 2.

A trial is defined as leader election(s) started by one vehicle within a group. The number of vehicles denotes the size of the group in which the leader election is executed. Start time is defined as the moment that a vehicle initiates a leader election by broadcasting an ELECTION message. End time is defined as the moment that the last member in a group receives the LEADER message. Number of rounds is the number of Leader Election Protocol (third version) being executed for a single trial. For example, if the leader election succeeds in the first execution (no ABORT sent), the number of rounds should be 1. But if the first execution ends by ABORT while the second one succeeds, the number of rounds should be 2. We confirm correctness by checking if the leader elected is the one with the largest Vid within the group.

Table 2: A 60 s Experiment for the Leader Election Protocol (third version)

Trial	Number of vehicles	Start time (s)	End time (s)	Time elapsed (s)	Number of rounds	Correctness
1	2	3.066	3.081	0.015	1	TRUE
2	2	6.333	6.363	0.030	1	TRUE
3	2	6.498	6.530	0.032	1	TRUE
4	5	6.696	6.776	0.080	1	TRUE
5	2	9.105	9.133	0.028	1	TRUE
6	18	10.425	10.578	0.153	1	TRUE
7	23	13.263	13.684	0.421	1	TRUE
8	22	20.292	20.892	0.600	1	TRUE
9	2	22.437	22.710	0.273	1	TRUE
10	28	24.351	24.930	0.579	1	TRUE
11	28	27.717	29.451	1.734	2	TRUE
12	30	32.139	32.722	0.583	1	TRUE
13	32	33.259	33.587	0.328	1	TRUE
14	34	34.607	35.473	0.866	1	TRUE
15	36	39.036	40.820	1.784	2	TRUE
16	37	43.260	43.963	0.703	1	TRUE
17	38	46.593	47.322	0.729	1	TRUE
18	38	50.652	52.445	1.793	2	TRUE
19	38	56.625	57.271	0.646	1	TRUE
20	39	60.519	62.216	1.697	2	TRUE

It is worth noting that four of the trials took two rounds, i.e., two executions of Algorithm 3, to elect a leader. After thorough analysis, we realize that one of the vehicles exited the simulation during the leader election, which resulted in a topology change. Thus an additional round is required to reach consensus.

Table 3 is a summary of five experiments conducted. Notice the running time for [31, 50] vehicles is actually smaller than [21, 30] vehicles. This is because in simulation, a few more cases for [21, 30] vehicles required two rounds. Nonetheless, the execution time for all cases is below 1 second.

Table 3: Summary of Five Experiments for the Leader Election Protocol (third version)

Number of vehicles	[2, 10]	[11, 20]	[21, 30]	[31, 50]
Average running time (s)	0.0682962963	0.435375	0.834273913	0.7639272727

One of the problems encountered in the simulation is that when two or more vehicles simultaneously start the Leader Election Protocol, a vehicle with a sub-optimal proposal may occasionally be selected as leader. However, we noticed that even in such cases, no more than one leader existed in the same group.

4.3 Group Management Evaluation

Though the Leader Election Protocol is efficient and accurate, it does not provide any property of group maintenance. For example, since no Leader Heartbeat message is sent, vehicles recently entering the region will not be able to detect the existence of the leader and therefore restart the leader election. As a result, 20 elections are executed in a one-minute experiment. In this section, we will test the group management protocols discussed from Section 3.3 to 3.7.

4.3.1 Group Management without Leader Handover

In this experiment, in addition to the Leader Election Protocol, we enabled all group management messages except Leader Handover. We run each experiment for 180 seconds and record important information. Table 4 shows the result from one experiment.

The start time of the Group Merging event is defined as the first timestamp at which a conflict is discovered (there are Leader Heartbeats from two unique leaders). The end time of the Group Merging event is defined as the timestamp at which the last member in the group removes the leader resigned. We confirm correctness by checking if the leader who resigned during merging is the one with a smaller id, which is the rule we set to solve conflicts.

Table 4: A 150 s Experiment for Group Management Protocols (without Leader Handover)

Event number	Event type	Number of vehicles	Start time (s)	End time (s)	Time elapsed (s)	Number of rounds	Correctness
1	Leader Election	2	6.650	6.669	0.019	1	TRUE
2	Leader Election	2	7.013	7.034	0.021	1	TRUE
3	Leader Election	2	7.937	7.957	0.020	1	TRUE
4	Group Merging	4	8.046	8.066	0.020	NA	TRUE
5	Group Merging	6	8.977	9.032	0.055	NA	TRUE
6	Leader Election	3	9.290	9.351	0.061	1	TRUE
7	Group Merging	9	10.386	13.412	3.026	NA	TRUE
8	Leader Election	18	13.778	14.093	0.315	1	TRUE
9	Leader Election	33	48.362	48.96	0.598	1	TRUE
10	Leader Election	40	98.819	99.416	0.597	1	TRUE
11	Leader Election	36	151.949	152.873	0.924	1	TRUE

At the initial stage of the experiment, vehicles enter the map in four directions and form independent groups separately. Later, as the groups move toward the center of the map, they start to merge into a larger group. The process continues until event 7 is finished. Once the density of vehicles increases on the map, a single large group will be formed. In the single large group, the leader being elected will never change until it leaves the region. This explains why leader election occurs much less frequently in the later part of the simulation. No Leader Handover is performed as this property is not enabled.

Note in event 7 that we have an outlier in Time elapsed when compared to other Group Merging events. After investigation, it turns out that the Leader Resignation message was lost and the Group Merging process is not able to complete immediately. However, due to LAMP property, after three consecutive missing heartbeats, the members will automatically remove the resigned leader, which explains why event 7 terminates roughly 3 seconds after the groups start to merge.

4.3.2 Group Management with Leader Handover

In this experiment, we enabled all group management messages including Leader Handover Protocol (by LAMP). We run each experiment for 180 seconds and record important information. Table 5 shows the result from one experiment.

The start time of the Leader Handover event is defined as the timestamp at which the leader broadcasts a NEWLEADER message in Algorithm 6. The end time of the Leader Handover event is defined as the timestamp at which the last member in the group removes the old leader. We confirm the correctness for Leader Handover by checking if every member in the group changes the leader to the new leader.

At the initial stage of the experiment, vehicles enter the map in four directions and form independent groups separately. Later, as the groups move toward the center of the map, they started to merge into a larger group. The process continues until event 6 is finished. Once the density of vehicles increases on the map, a single large group will be formed. In the single large group, the leader being elected will never change until it performs a Leader Handover.

In this simulation, we define that the leader will perform a Leader Handover immediately after it passes the intersection. The rule for selection of a new leader gives priority to the vehicles on the perpendicular road. As a result, a pattern will emerge: (i) the leader waits at a red traffic light; (ii) the traffic light for leader turns from red to green after the red cycle ends, and the traffic light for the perpendicular road turns from green to red after the green cycle ends; (iii) the leader passes the intersection and hands over the leadership to another vehicle waiting for red light on the perpendicular road; and (iv) the new leader waits at the red traffic light and repeats from (i). Because the periods for both red and green traffic lights are set to 25 seconds, the Leader Handover should take place roughly every 25 seconds, which matches the observation in the latter part of the simulation.

Table 5: A 150 s Experiment for Group Management Protocols (with Leader Handover)

Event number	Event type	Number of vehicles	Start time (s)	End time (s)	Time elapsed (s)	Number of rounds	Correctness
1	Leader Election	2	7.244	7.276	0.032	1	TRUE
2	Leader Election	7	7.574	7.88	0.306	1	TRUE
3	Group Merging	9	8.279	9.104	0.825	1	TRUE
4	Leader Election	16	13.844	14.284	0.440	1	TRUE
5	Leader Election	3	20.114	20.158	0.044	1	TRUE
6	Group Merging	19	21.168	21.369	0.201	NA	TRUE
7	Leader Handover	33	42.852	43.327	0.475	NA	TRUE
8	Leader Handover	35	61.287	61.511	0.224	NA	TRUE
9	Leader Handover	38	89.008	89.228	0.220	NA	TRUE
10	Leader Handover	34	113.193	113.420	0.227	NA	TRUE
11	Leader Handover	38	139.254	139.465	0.211	NA	TRUE
12	Leader Handover	40	163.404	163.633	0.229	NA	TRUE

Table 6 is a summary of five experiments conducted with all group management protocols enabled. Note when calculating the average, we exclude the cases where

Leader Resignation message is lost. The LAMP property will take care of those situations and the average running time is around 3 second. For the remaining cases, the average is much less than 1 second.

Table 6: Summary of Five Experiments for Group Management Protocols

Event	Group Merging	Leader Handover
Average running time (s)	0.245	0.256

4.4 VTL Application

In the last simulation, we apply our Leader Election Protocol and Group Management Protocols to build a VTL application. In the simulation, we define the leader to be the only vehicle allowed to change the traffic light signal. We also adopt the idea from [7] and design a simple algorithm for the leader to follow as shown in Algorithm 8.

```
Before passing the intersection:
    If the traffic light on my road is green
        Pass the intersection
    Else
        If my speed is below $V_{Threshold}$
            Change the traffic light on the perpendicular road to red
            Change the traffic light on my road to green
            Pass the intersection
        Else
            Brake and slow down

After passing the intersection:
    If there is any vehicle on the perpendicular road
        Select a random vehicle from the perpendicular road
        Start Leader Handover Protocol
    Else
        Select a random vehicle
        Start Leader Handover Protocol
```

Algorithm 8: A Simple Algorithm for the Leader in a VTL Application

The basic idea of Algorithm 8 is to allow the leader to change the traffic light of its lane from red to green whenever the leader stops at a red light. However, in simulation, it takes a long time before the velocity of the leader reaches 0. Therefore, a threshold for velocity is defined, and all vehicles with speed lower than $V_{Threshold}$ will be considered as stopped. In the experiments, we set $V_{Threshold} = 0.8$ mile/hour.

We run the same experiment five times with 120 s for each. All other conditions are set the same as in Section 4.3.2 except that the traffic light signal is now controlled by the leader. The passing time is defined as the period for a vehicle between entering and leaving the map.

Table 7: Simulation with VTL Application

Virtual Traffic Light	Simulation time(s)	Vehicles passed	Average passing time(s)
Trial 1	122	49	22.107
Trial 2	143	38	19.249
Trial 3	130	36	23.492
Trial 4	132	48	24.014
Trial 5	128	45	23.643
Final average passing time			22.501

For comparison, we conduct the experiment with the same parameters using Periodic Traffic Lights. The period for both the red and green lights is set to 35 s, 25 s, and 15 s respectively. For each Periodic Traffic Light, we simulate three trials to obtain a more stable result.

Table 8: Simulation with Periodic Traffic Light (35 s)

Periodic Traffic Light (35 s)	Simulation time(s)	Vehicles passed	Average passing time(s)
Trial 1	131	41	32.271
Trial 2	151	49	29.577
Trial 3	131	43	31.151
Final average passing time			31.000

Table 9: Simulation with Periodic Traffic Light (25 s)

Periodic Traffic Light (25 s)	Simulation time(s)	Vehicles passed	Average passing time(s)
Trial 1	152	44	25.052
Trial 2	131	34	28.491
Trial 3	133	45	29.614
Final average passing time			27.719

Table 10: Simulation with Periodic Traffic Light (15 s)

Periodic Traffic Light (15 s)	Simulation time(s)	Vehicles passed	Average passing time(s)
Trial 1	138	40	25.828
Trial 2	130	41	24.676
Trial 3	139	37	23.427
Final average passing time			24.644

It is easy to observe from the results above, when compared to 15 s, 25 s, and 35 s Periodic Traffic Light, our VTL application has 8.7%, 18.8%, and 27.4% reductions in the average passing time for vehicles.

It is also worth noting that the algorithm we adopted for traffic light signal control is very simple, as it is not the focus of this thesis. In theory, it is possible to adopt a more complicated strategy for traffic light control in order to achieve better performance.

CHAPTER 5
DISCUSSION AND OPTIMIZATION

5.1 Communication Message Overhead

In the current design of our protocol, the communication message overhead may occur when a vehicle tries to propagate a message within the group. For example, when the leader tries to propagate a Leader Heartbeat in the group through broadcasting, many duplicate Leader Heartbeats will be rebroadcasted by members even if it is not necessary.

Creating a spanning tree within the group to help in propagating messages and reducing overhead is definitely a good idea. However, as the spanning tree has been studied extensively in other works and is out of the scope of this thesis, we will not discuss such optimization. Instead, we will present two ideas for optimization using geolocation information, which we adopt from the distance-based scheme and location-based scheme in [14]. Note that in both scenarios we assume that vehicles are aware of the maximum communication distance between vehicles.

The first optimization prevents rebroadcasting if all neighbors can be reached by the sender. For example, in Figure 7, A is the leader and is periodically broadcasting a Leader Heartbeat for group maintenance. In standard settings, B, C, and D will rebroadcast the Leader Heartbeat immediately after receiving the message, which is unnecessary since all members in the group can receive the message directly from A. However, under the first optimization, B will check its neighbor list and use the location information stored for A, C, and D to determine whether C and D can hear the message from sender A directly. Therefore, B can avoid sending the Leader Heartbeat again. C and D will take similar measurements and avoid sending unnecessary messages. In this scenario, the communication messages to propagate the Leader Heartbeat are reduced from four to one.

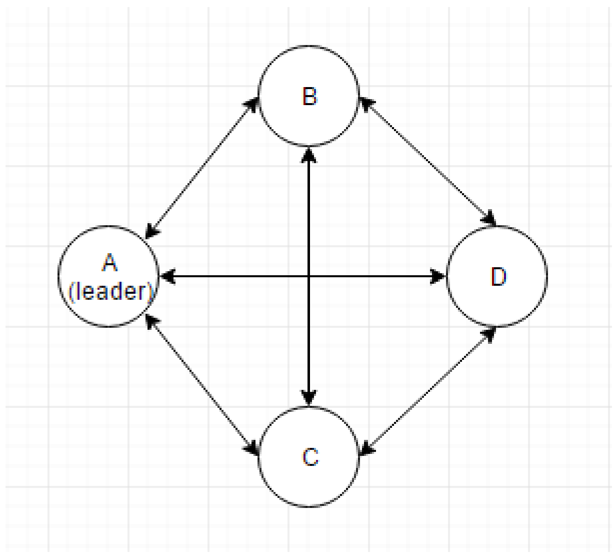


Figure 7: Scenario for First Optimization

The second optimization prevents rebroadcasting if any neighbor i, which cannot be reached by the sender, has at least another neighbor j closer to i, and j can be reached by the sender. For example, in Figure 8, A is the leader and is periodically broadcasting a Leader Heartbeat for group maintenance. In standard settings, B and C will both rebroadcast the Leader Heartbeat so that D will be able to receive the message, though one of the rebroadcasts is unnecessary. However, under the second optimization, B will check the neighbor list and realize C can receive message from A and is closer to D. Therefore, B can avoid sending the Leader Heartbeat again. In the meantime, C will check the neighbor list and realize D cannot receive the message from A and there is no other node able to receive the message from A and closer to D. Therefore, C will send the Leader Heartbeat again. For D, after receiving the message from C, it will check the neighbor list and realize that both B and C can receive the message from C. Therefore, D can avoid sending the Leader Heartbeat again. In this scenario, the communication messages to propagate the Leader Heartbeat are reduced from four to two messages.

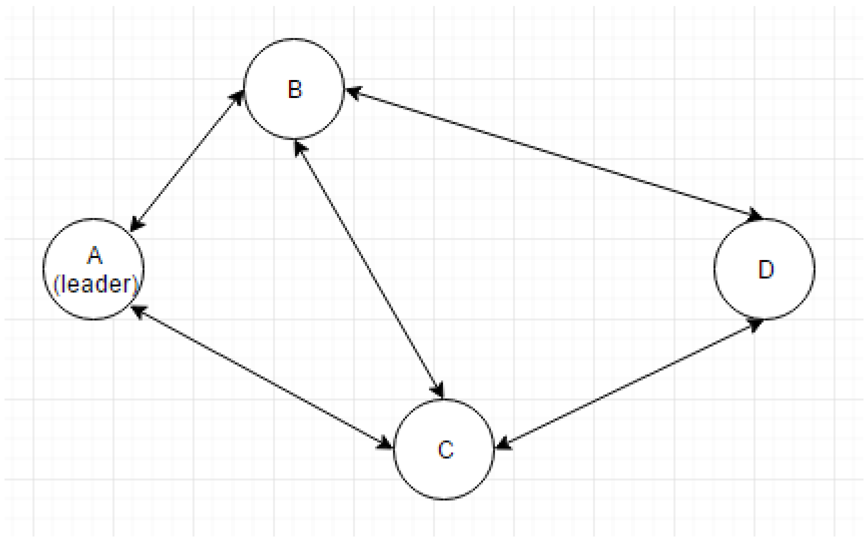


Figure 8: Scenario for Second Optimization

5.2 Neighbor List Reduction

In our Leader Election Protocol, a vehicle is considered as a neighbor of another vehicle if its BEACON message can be received. However, in dense network topology, such as downtown Manhattan, the vehicles are extremely close to each other and may have more than 30 neighbors. Since our protocol requires confirmation from every neighbor in order to continue execution, a single message drop will delay the entire election. In order to address such issue, two optimizations are proposed.

The first optimization reduces the number of neighbors by setting a predetermined maximum distance between vehicle pairs that can be considered as neighbors. For example, in downtown Manhattan, such a number could be 30 m while 50 m~100 m for a rural area. This method is extremely easy to implement, but may introduce unnecessary group partitions once in a while.

The second optimization reduces the number of neighbors by selectively erasing vehicle pairs from each other's neighbor list. For example, if vehicles A and C both find that there is another vehicle B right in between them, then both A and C can remove each other from their own neighbor list. This method is relatively difficult to implement and may result in increase in group diameter, but it is effective in preventing group partition.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this thesis, leader election and group management are studied in the context of VANET. By adopting the idea from [13] and making improvements, a new Leader Election Protocol is proposed. It takes advantage of two properties of the VANET: the overhearing property of the wireless network reduces unnecessary communication messages; and location information allows vehicles to closely track the status of neighbors as well as perform leader handover. Furthermore, we present a set of group management protocols, in which solutions for conflict, group partition, and group merging are provided. Evaluation has shown that the Leader Election Protocol is able to achieve high efficiency (< 1 s for Leader Election) and high accuracy, while the group management protocols provide high efficiency (< 0.5 s for Group Merging and Leader Handover) and high stability.

6.2 Future Work

The next step of this study is to deploy the algorithms in real-world devices to test its performance. It does not have to be on vehicles, for example, android-based robots will be able to utilize the algorithm for leader election and share important data among group members.

In addition, as a continuity of the study, it would be interesting to extend the algorithms to include shared memory management. Once the concept of shared memory is integrated into the protocol, it will be very convenient to implement a VTL application on top of it, which is essentially creating a multi-read single-write instance.

Last but not least, as the technology advances, there are more and more applications that can be implemented in the field of VANET. For example, an electronic brake light alerts a driver when cars in front brake suddenly, even if sight of the brake light is obscured by other vehicles. A supervisor vehicle, like a police vehicle, may be allowed to access any driver’s public information without the need to stop the driver’s vehicle. Our hope is that, with the technology studied in this thesis, some day these applications will become available in our daily lives.

REFERENCES

[1] Hartenstein, H., and Laberteaux, K. P. (2008). A tutorial survey on vehicular ad hoc networks. *Communications Magazine, IEEE*, *46*(6), pp. 164-171.

[2] Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, *32*(2), pp. 374-382.

[3] Santoro, N., and Widmayer, P. (2007). Agreement in synchronous networks with ubiquitous faults. *Theoretical Computer Science*, *384*(2), pp. 232-249.

[4] Schmid, U., Weiss, B., and Keidar, I. (2009). Impossibility results and lower bounds for consensus under link failures. *SIAM Journal on Computing*, *38*(5), pp. 1912-1951.

[5] Biely, M., Schmid, U., and Weiss, B. (2011). Synchronous consensus under hybrid process and link failures. *Theoretical Computer Science*, *412*(40), pp. 5602-5630.

[6] Gradinescu, V., Gorgorin, C., Diaconescu, R., Cristea, V., and Iftode, L. (2007, April). Adaptive traffic lights using car-to-car communication. In *Proceedings of the IEEE Vehicular Technology Conference*, pp. 21-25.

[7] Ferreira, M., Fernandes, R., Conceição, H., Viriyasitavat, W., and Tonguz, O. K. (2010, September). Self-organized traffic control. In *7th ACM International Workshop on Vehicular Inter-Networking*, pp. 85-90.

[8] Nakamurakare, M., Viriyasitavat, W., and Tonguz, O. K. (2013). A prototype of Virtual Traffic Lights on android-based smartphones. In *Proceedings IEEE SECON*, pp. 236-238.

[9] Neudecker, T., An, N., and Hartenstein, H. (2013, December). Verification and evaluation of fail-safe virtual traffic light applications. In *Vehicular Networking Conference (VNC)*, pp. 158-165.

[10] Holzmann, G. J. (1993). Design and validation of protocols: A tutorial. *Computer Networks and ISDN Systems*, *25*(9), pp. 981-1017.

[11] Fathollahnejad, N., Villani, E., Pathan, R., Barbosa, R., and Karlsson, J. (2013, June). On reliability analysis of leader election protocols for virtual traffic lights. In *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*, pp. 1-12.

[12] Sommer, C., Hagenauer, F., and Dressler, F. (2014, March). A networking perspective on self-organizing intersection management. In *Internet of Things (WF-*

IoT), 2014 IEEE World Forum on, pp. 230-234.

[13] Vasudevan, S., Kurose, J., and Towsley, D. (2004, October). Design and analysis of a leader election algorithm for mobile ad hoc networks. In *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference on*, pp. 350-360.

[14] Tseng, Y. C., Ni, S. Y., Chen, Y. S., and Sheu, J. P. (2002). The broadcast storm problem in a mobile ad hoc network. *Wireless Networks*, *8*(2-3), pp. 153-167.

[15] Fernandes, R., Vieira, F., and Ferreira, M. (2012, November). VNS: An integrated framework for vehicular networks simulation. In *Vehicular Networking Conference (VNC), 2012 IEEE*, pp. 195-202.

[16] Henderson, T. R., Lacage, M., Riley, G. F., Dowell, C., and Kopena, J. (2008). Network simulations with the ns-3 simulator. *SIGCOMM Demonstration*, *14*.

APPENDIX A

EXAMPLE PSEUDO CODE FOR SIMULATION

An example pseudo code for the third version of Leader Election Protocol is provided. If you are interested in the simulation source code, please contact Yixiao Nie at nie4@illinois.edu or lucasnyx37@gmail.com to request access.

```
INITIALIZATION:
Parent=-1, In_election=false, Leader=-1, Proposal=-1, ProposalId=-1, AckList={},
ChildrenList={}, NeighborList=get_neighbors(), Vid=get_my_id();
BEGIN:
If (is_source()) {
    broadcast(ELECTION(“Parent”=-1, “Sender”=Vid));
    in_election=true;
}
While(an ELECTION packet p is received) {
    AckList.add(p.Sender);
    If (!In_election) {
        In_election=true;
        Parent= p.Sender;
        broadcast(ELECTION(“Parent”= Parent, “Sender”=Vid));
    } else {
        If (Vid == p.Parent) ChildrenList.add(p.Sender);
    }

    If (AckList == NeighborList) break;
}

Proposal = get_proposal_value();
ProposalId = Vid;

While(1) {
    If (ChildrenList == < >) {
        broadcast(PROPOSAL(“Proposal”=Proposal,        “ProposalId”=ProposalId,
“Sender”=Vid);
        break;
    }
    If (a PROPOSAL packet p is received) {
        If (ChildrenList.has(p.Sender)) {
            ChildrenList.remove(p.Sender);
            If (Proposal < p.Proposal or (Proposal == p.Proposal and ProposalId <
p.ProposalId) {
                Proposal = p.Proposal;
                ProposalId = p.ProposalId;
            }
        }
    }
}

If (is_source()) {
    Leader = ProposalId;
    In_election=false;
    Broadcast(LEADER("Leader"= Leader);
}

If (In_election and a LEADER packet p is received) {
    Leader = p.Leader;
    In_election=false;
    Broadcast(LEADER("Leader"= Leader);
}
END.
```

(NOTE: here we omitted the code that sets a timeout for sending ABORT message in case of failure)