2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing

A Probabilistic Analysis of a Leader Election Protocol for Virtual Traffic Lights

Negin Fathollahnejad[1], Raul Barbosa[2], Johan Karlsson[1]
[1]Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden
[2]CISUC, Department of Informatics Engineering, University of Coimbra, Portugal

***Abstract*—Vehicle-to-vehicle communication systems support diverse cooperative applications such as virtual traffic lights. However, in order to harness the potential benefits of those applications, one must address the challenges faced by distributed algorithms in environments based on unreliable wireless communications. In this paper we address the problem of leader election among the nodes of a cooperative system, under the assumption that the network is unreliable and any number of messages may be lost, and that the number of participating nodes is initially unknown. It is known from the literature that a deterministic solution to this problem does not exist. Nevertheless, one may devise probabilistic solutions that provide arbitrarily high probability of success. In the proposed solution, nodes are enhanced with a local oracle that provides minimal information on the state of the system. Such local oracles, even if unreliable, are shown to increase the probability of success in reaching consensus on the result of the leader election.**

Keywords*—*distributed algorithms; consensus; leader election; probabilistic analysis; unknown networks; communication failures.

I. Introduction

Recent advances in wireless technologies for vehicle-to-vehicle communications have made it possible to develop cooperative applications that can improve traffic safety and fuel efficiency of road vehicles. An example of such applications is the virtual traffic light systems (or VTLs). The concept of a VTL is introduced by Ferreira et al. as a self-organizing traffic control system that allows the road vehicles passing an intersection to implement the function of a traffic light without the presence of a roadside infrastructure [1].

The development of automotive cooperative systems comprises a lot of challenges one of which is to provide reliable and efficient means for the cooperating vehicles to make coordinated decisions. For example, in a VTL, vehicles approaching the intersection interact via wireless communication in order to elect a VTL leader. The VTL leader is a vehicle that agrees to serve as a temporary traffic controller for the intersection. After having served as VTL leader for short while, say one or two minutes, the vehicle resigns and another vehicle is elected VTL leader. Clearly, if the vehicles in and around the intersection would disagree on who the VTL leader is, serious accidents may occur.

Agreeing on the identity of a leader in a distributed system is a variant of the well-known *consensus* problem. We know from previous research, e.g. [2], that there is no deterministic solution to the consensus problem for systems that can experience an unlimited number of messages losses during the execution of a distributed consensus algorithm. Since it is unrealistic to assume an upper bound on the number of message losses in a cooperative automotive system, see e.g. [4], we are interested in finding good probabilistic solutions to consensus problems in such systems. We have previously studied this problem under the simplifying assumption that all nodes executing a consensus algorithm know n, the number of participating nodes, a priori, i.e., before the execution of the algorithm begins. We introduced the term *1-of-n* selection as an alternative name for the consensus problem under this assumption, and proposed a family of algorithms that provide approximate (i.e. non-perfect) solutions to the *1-of-n* selection problem [3].

In this paper, we address the problem of consensus in the context of VTL leader election for systems where the set of participating nodes initially is unknown to all nodes. We denote the consensus problem under this assumption as the *1-of-** selection problem, where the $*$ symbolizes the fact that n is initially unknown to all participating nodes. This problem is also called Consensus with Unknown Participants (CUP) [4]. The CUP problem is fundamental to the problem of bootstrapping self-organized networks where there is no central authority to initialize each node with the necessary information of the system.

We propose a VTL leader election protocol that uses a *1-of-** selection algorithm as its core logic. There are two possible outcomes of the protocol for a node: it either decides on a leader or decides to abort. We investigate the behaviour of the protocol for three different selection algorithms called the *optimistic**, the *pessimistic** and the *moderately pessimistic** algorithm. The names of these algorithms refer to the type of decision criterion that the algorithm uses. The algorithms are derived from our previous work on *1-of-n* selection [5], where we proposed three decision criteria called the *optimistic*, *pessimistic* and *moderately pessimistic* decision criterion. Since we assume that an arbitrary number of messages may be lost, the execution of the leader election protocol may lead to disagreement among the nodes in the system. Our results show that the probability of disagreement varies significantly for the different decision criteria.

We consider a system of an unknown number of nodes communicating over unreliable links where any number of messages can be lost among the nodes. Each node (or process[1]) that participates in a leader election protocol executes the selection algorithm for R rounds of message exchange. At the end of round R, depending on the specified decision criterion[2] for the algorithm, each process either decides to *select a leader*

[1]We refer to the computing process of a node as process. In this paper, we use the terms of node and process interchangeably

[2]A decision criterion refers to the logical expressions based on which a process makes a decision which depends on the amount of information a process has obtained from the system.

or it decides to *abort* due to lack of information. Disagreement occurs if some processes decide to select a value, while the remaining processes decide to abort.

We consider two types of processes; *aborting* processes and *non-aborting* processes. We call a process an *aborting* process if after the execution of the selection algorithm, it decides to abort. Likewise, we define a process as *live* or *non-aborting* if it decides to select a leader. Considering the outcomes of all processes, the selection algorithms have three main outcomes: (i) *agreement on a leader*, (ii) *agreement to abort*, (iii) *disagreement*. We have *agreement* if the participating processes in the algorithm either all decide on the same leader or they all decide to abort. We have *disagreement* if the processes decide on different leaders or if some processes decide on one leader while others decide to abort. Our analysis shows that, under our system model and failure assumptions, one cannot guarantee that the outcome of the algorithm always consists of one unique leader. As a result, we cannot guarantee safety in case of disagreement among nodes. Nevertheless, it is useful to further classify disagreement cases as *safe* and *unsafe* as follows:

Unsafe Disagreement: We have unsafe disagreement if there are ***at least*** two processes electing different *non-aborting processes* as the VTL leader, i.e., we have ***at least*** two live processes acting as leaders in the system.

Safe Disagreement: We have safe disagreement if all processes in a proper subset[3] of the system decide on a unique live process as the VTL leader, while the remaining processes either decide to abort or decide on an *aborting process* as the leader.

In order to reduce the probability of having unsafe disagreement among the processes, we propose the use of an extra component for each node in the system, called an *oracle*. The oracle is a local device attached to each node used for detecting other nodes in the system. An oracle is assumed to be unreliable and may report incorrect information, i.e., it might underestimate or overestimate the number of processes in the system. In order to account for the inaccuracy of the local oracles we use a *correction* parameter. In this paper, for simplicity we assume the same correction parameters for all processes.

The main contributions of the paper are as follows. We investigate the behaviour of a family of selection algorithms which aim to solve the problem of *1-of-$*$* selection in an unknown system and lossy links. We propose the selection algorithms as the core logic of a VTL leader election protocol. We calculate the probabilities of each outcomes of the selection algorithms as a function of the probability of message loss, using a probabilistic model checker called PRISM. We investigate how different settings of the system parameters, i.e., the number of rounds R, the oracle values, the correction parameter and the number of participants can influence the probability of having unsafe disagreement.

The rest of this paper is organized as follows: Section II presents related work on leader election algorithms for systems with unknown participants and lossy links. In Section III we explain our proposed VTL leader election protocol in more details. We formally specify our system model and failure assumptions in Section IV-A. We explain our proposed selection algorithms in Section IV-B. In Section V, we present a probabilistic analysis of the selection algorithms under different system settings and probabilities of message loss. Finally in Section VI we present some discussions and conclusions.

[3]A proper subset S^* of a set S, is a set which excludes at least one member of S.

II. Related Work

The problem of leader election has been widely investigated over the last decades, e.g. [6], [7]. In [7], the problem is addressed for a set of n nodes with unique identifiers communicating over reliable FIFO channels. It is assumed that eventually there is a unique leader elected by the nodes providing that n is fixed. In the present work, we are mainly interested in solving the leader election problem in unknown wireless networks with unreliable links where any number of messages can be lost due to communication failures.

There are several papers in the literature on the problem of leader election in dynamic networks with lossy links such as in [8]–[13]. However, most of the proposed approaches stipulate restrictions on the communication failure model of the system. For example the authors of [12] propose a protocol to solve the problem of highly available local leader election for a distributed system where the set of processes can split in disjoint subsets due to network failures. In this problem, ideally there must be one leader elected for each subset (or partition). However, due to the impossibility results for systems with unrestricted communication failures [2], the election of a unique leader can not be guaranteed. For this, the authors of [12] introduce the notion of a *stable* partition of a group of nodes connected to each other which are required to elect a leader within a bounded time.

In [9], a self-stabilizing leader election algorithm for an ad-hoc network is proposed. The suggested algorithm uses an approach based on directed acyclic graphs with the advantage of detecting partitions automatically using the TORA mechanism, i.e., the Temporally Ordered Routing Algorithm developed by Vincent Park and Scott Corson. In [14], the problem of leader election is investigated for a system of nodes which have partial knowledge about the other nodes in the system. The authors of [14] define the necessary and sufficient conditions on the global knowledge that nodes should be provided with in order to solve the leader election problem. It is assumed that nodes are using asynchronous and reliable communication links to expand their knowledge of the network over rounds of communicating with each other. They prove that with knowing the size of the network it is possible to solve the election problem on every network whereas knowing only an upper bound on the size is not enough. In [15] the problem of eventual leader election in dynamic and unknown network is solved under a communication model called Time-Varying Communication Graph. In this model, it is assumed that the network remains connected over time.

The methods suggested in the literature to solve the problem of leader election considering the Santoro and Widmayer's impossibility results have mostly a preventive approach. However, it is possible to design algorithms to reduce the probability of failing to reach consensus as much as needed, so as to meet specific requirements on reliability and availability. This intuition has been explored to build protocols that maximize the probability of correctness by accumulating more information over a larger duration of the execution [16]. Researchers have also focused on stochastic models of verifying the probability of transitioning into an incorrect state [17].

III. The VTL Leader Election Protocol

A Virtual Traffic Light (VTL) is a self-organizing traffic control system that allows the road vehicles passing an intersection to implement the function of a traffic light without the need for a roadside infrastructure [1], [18]. Ferreira et al. in [1] propose a VTL scheme which relies on two main procedures: *leader election* and *leader handover*.

Fig. 1 shows an example of how a VTL can be established among the cars in a 4-leg intersection. Each leg of the intersection consists of a cluster of cars travelling in the same direction. In each leg, there is one car acting as the leader of the cluster[4]. It is assumed that only the cluster leaders participate in the VTL leader election protocol. The cluster leaders should reach to an agreement on one them to be the VTL leader. A VTL leader is responsible to take the role of the traffic controller for a temporarily amount of time, called the *control period*. During the control period, the VTL leader assumes a red light for itself while it broadcasts the state of the virtual traffic light to other vehicles in the intersection. At the end of the control period, the leader hands over the leadership to another vehicle provided that there exists one in the intersection (*leader handover*). Otherwise, no leadership handover is performed and the new coming vehicles to the intersection must elect a new leader. Finally the leader gives the green light to itself and passes the intersection.

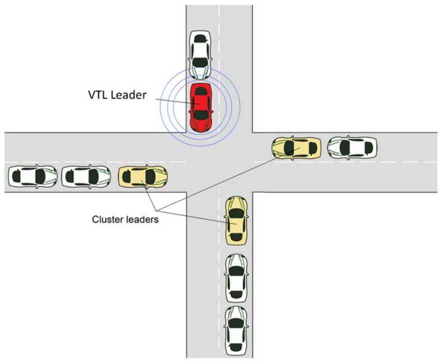


Fig. 1. Virtual Traffic Light

Fig. 2 depicts the principle of the operation of our proposed VTL leader election protocol to be run by each cluster leader in an intersection. We consider that all vehicles have access to the GPS in order to detect the approaching intersections where a VTL can be created. Moreover, we assume that all vehicles are equipped with the necessary technologies for data dissemination in a vehicular network in order to participate in a VTL system [19], e.g. the features for VANET geographical routing protocols such as beaconing and location table (See e.g. [20]).

[4]The problem of electing a cluster leader is similar but not identical to that of electing a VTL leader.

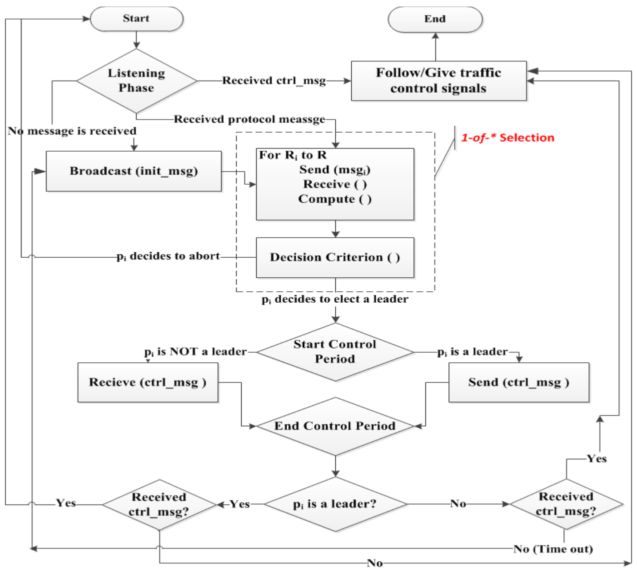


Fig. 2. Leader Election Protocol

As we see from Fig. 2, each vehicle (or process[5]) executing the VTL leader election protocol goes through a number of steps. Each computing process of a vehicle starts executing the protocol with listening to the network (*listening phase*). Depending on the outcome of the listening phase, a process either : (i) starts executing a selection algorithm to solve the problem of *1-of-** selection or (ii) it follows the traffic light orders that it receives from an existing VTL leader. Each process after participating in a selection algorithm starts a phase called the *control period*. During a control period, the elected VTL leader sends out traffic light orders to the cars present in the intersection while the other cars follow the traffic orders. In the following, we explain each step in detail.

≪Listening Phase≫ Each vehicle approaching an intersection, first enters a *listening phase* in which it listens to the network for a period of time to receive messages. The duration of the listening phase is set to a factor of R, the number of rounds of execution of the selection algorithm. A vehicle in its listening phase might receive two different types of messages: (i) the traffic control messages denoted by $ctrl\_msg$ or (ii) the VTL protocol messages. The traffic control messages are the traffic light signals sent from a VTL leader to the vehicles, i.e. a green, yellow or red light. The VTL protocol messages are the messages communicated among the vehicles during the execution of a selection algorithm to solve the *1-of-** selection problem. If a vehicle receives a traffic control message from a vehicle, it sets the sender of the messages as the VTL leader and follows its traffic orders until it passes the intersection with a green light. If a vehicle receives a protocol message from another vehicle indicating that a *1-of-** selection process is currently running, it adjusts its starting round number according to the sender's round number and joins the protocol.

A vehicle during its listening phase may receive a protocol message called the initiating message (or $init\_msg$) from

[5]The process refers to the computing process in each vehicle that is responsible to participate in the function of a VTL.

another vehicle which is sent to call for starting the execution of a selection algorithm. A process which receives an initiating message during its listening phase will start executing the selection algorithm.

[Broadcast Initiating Message]: If a vehicle does not receive any message during its listening phase, it broadcasts a protocol initiating message ($init\_msg$) to start the execution of a selection algorithm. Such a process is called an *initiator*. We assume that an initiator process will continue executing the protocol regardless of receiving acknowledgement messages from other processes or not. The processes which are in the middle of running a VTL protocol do not accept proposals for a new protocol initiation until their next possible *listening phase*.

[*1-of- Selection]:** Each vehicle that participates in the process of a *1-of-** selection executes a selection algorithm in order to reach an agreement on a leader with other vehicles. Each process in each round executes the three steps of *send*, *receive* and *compute* messages. We explain the details of each step in Section IV-A. All vehicles are assumed to run the algorithm in exactly R rounds of message exchange which is fixed at the design time. For a vehicle which joins the VTL protocol at a later round, we assume that it has failed to send/receive messages during the missed rounds due to communication failures. Similarly, we can assume that the messages sent and received from a process which leaves the protocol before the end of round R are lost.

[Decision Criterion]: After R rounds, each process executes the function of a decision criterion based on which it either decides to select a leader or it decides to abort. A process which decides to abort should start the protocol from the beginning by running the listening phase. A process which decides on selecting a leader will start the *control period* phase.

$\ll$**Control Period**$\gg$ During the control period, a process which has selected another process as the leader should wait to receive traffic control messages from the selected leader. The process that has selected itself as the leader must send traffic control messages (denoted by $ctrl\_msg$) to the network. A non-aborting process, which is not a leader listens to the traffic control messages until it receives the green light to pass the intersection[6]. A non-aborting process which has not selected itself as the leader and has not received any $ctrl\_msg$ until the end of the control period, concludes that the corresponding elected leader is either not alive or all messages it sent are lost. Therefore, it broadcasts a request for creating a new VTL by sending initiating messages. If the process which has selected itself as the leader receives $ctrl\_msg$ from another process(or other processes) during the control period, for safety reasons due to the presence of multiple leaders in the system, it must abort and start the leader election protocol from the beginning.

In the following, we describe our system model and failure assumptions. We then explain the details on the selection algorithm as the core logic of the leader election protocol.

[6]In the current specification of the protocol, we assume that once a vehicle receives traffic control messages during its control period it will eventually receive the green light signal and passes the intersection.

IV. Protocol Description

A. System Model and Failure Assumptions

We consider a system of a set of finite number of processes where the number and the identities of the processes are initially unknown to all processes. We assume that there can be any number of processes in this set, i.e. there is no bound on the size of the set. Processes are assumed to have unique identifiers, i.e. there are no two processes with the same identifier. In other words, the set of all existing processes and their identities is known (e.g. in a VTL we know all the existing license plates for vehicles), however the processes initially do not know which subset of the existing processes is participating in the protocol.

We formally consider a set of processes denoted by $\mathcal{S} = \{p_1, p_2, \ldots\}$ that execute a round-based algorithm to reach agreement on one process among themselves as the leader. The processes execute R rounds of message exchange where in each round they send, receive and compute messages. It is assumed that the participating processes in the system have synchronized clocks relying on the existing clock synchronization methods for autonomous distributed applications such as in [21]. However, in a vehicular network, as all vehicles have access to the global positioning system, they can synchronize their clocks using the global information they acquire from the Global Positioning System (GPS) [22].

We assume that any number of messages can be lost during the execution of the algorithm. For example, a message sent by a process p_i may be received by all, a subset or none of other processes in the system. For simplicity, we assume that the processes are fault-free. Note, however, that a send omission failure of a process is equivalent to the loss of all messages sent by that process, and that a receive omission failure is equivalent to the cases where only one process fails to receive a message.

We consider that each process is equipped with a local oracle to detect the other participating processes in the system. The oracles are responsible to provide their corresponding processes with an approximation of the number of processes in the system. We assume that the oracles are unreliable in the sense that they may underestimate or overestimate the actual number of processes in the system. We define an oracle as *correct* if it reports the actual number of processes in the system while an *incorrect* oracle reports a different number from the actual number of processes in the system. For example, in a system of n processes a correct oracle reports $o_i = n$ while an incorrect oracle reports a different value $o_i \neq n$. In order to account for the unreliability of the oracles, we consider a *correction parameter* (denoted by c). In this paper, we assume that the correction parameter is set at the design time and is the same for all processes. If $c = 1$, it means that each process counts the value provided by its oracle as the actual number of processes in the system, i.e. it is assumed that the oracles are always correct. If $c < 1$, it is assumed that the oracles overestimate the number of processes in the system. Finally, if $c > 1$, it is assumed that the oracles underestimate the number of processes.

B. The Selection Algorithms

Alg. 1 shows the pseudocode of a generic algorithm for the proposed selection algorithms, i.e. the *optimistic**, the *pessimistic** and the *moderately pessimistic** selection algorithms. Each participating process executes the selection algorithm for R rounds of message exchange ($R \geq 1$). In each round, each process *sends*, *receives* and *computes* messages. The message of a process contains its proposed value (or *ranking value*[7]) and its *view* of the system (Π_i), i.e. $msg_i = \{proposed_i, \Pi_i\}$. The *view* of a process is the set of processes' identities which are known to that process. Initially, a process is only aware of its own identity in the system. So, at the first round the view of process p_i is $\Pi_i = \{p_i\}$ and its proposed value is its own identity, i.e. $proposed_i = p_i$. Over the rounds, the processes extend their views and update their proposed values according to the information they receive from other processes. A process always updates its proposed value to the highest proposed ranking value it received from the system. Finally, the process with the highest ranking value must be elected as the VTL leader.

Algorithm 1 Generic Selection Algorithm for p_i

```
1: $msg_i \leftarrow \{proposed_i, \Pi_i\}$;
2: for 1 to $R$ do
3:     begin_round
4:       Send ($msg_i$);
5:       Receive ();
6:       Compute ($msg_i$);
7:     end_round;
8: end for
9: Decision_Algorithm();
```

During the *Send* phase, a process p_i broadcasts its message (i.e. msg_i) to the network. Note that some of the receivers may not receive this message due to communication failures. Then, in the *Receive* phase, each process listens to the network to receive messages from other processes. At the end of each round, each process runs the *Compute* phase in order to update its message based on the information it received so far. Finally at the end of round R, a process executes a decision algorithm in order to either decide on selecting a leader or to abort due to the lack of information.

In the design of the *1-of-n* selection algorithm in [3], we proposed decision algorithms based on a primary decision condition for a process as follows: a process decides on selecting a value if it has a *complete view* of the system. In [3], we defined the view of process p_i as *complete* if it contained the information of all n processes in the system. Such a definition was based on the simplifying assumption that the set of processes is previously known to all processes, i.e. n is known. However, in the design of the selection algorithms it is assumed that n is initially unknown. Therefore, we define a new concept called *relatively complete*, denoted by $complete_r$: The view of process p_i is *relatively complete* if the number of processes in its view set (denoted by m_i) is greater than or equal to the factor of its oracle value (o_i) and the *correction* parameter (c), i.e. $m_i \geq c * o_i$. Similarly, we define the view of p_i *relatively incomplete* if $m_i < c * o_i$, denoted by $incomplete_r$.

[7]The ranking value depends on a number of parameters, such as the cluster leaders physical proximity to the intersection, its speed and the size of its cluster, or its driving direction. The procedure for selecting the ranking value is an important and elaborate part of an LEP, but its design is beyond the scope of this paper. In this paper, for simplicity, we assign the identity of a process as its ranking value.

Algorithm 2 Compute (msg_i) for p_i: *Optimistic**

```
1: for all $p_j$ such that $p_i$ has received $msg_j$ do
2:     if $proposed_i < proposed_j$ then
3:         $proposed_i \leftarrow proposed_j$;
4:     end if
5:     $\Pi_i \leftarrow \Pi_i \bigcup \Pi_j$;
6: end for
7: $msg_i \leftarrow \{proposed_i, \Pi_i\}$;
```

Algorithm 3 Decision_Algorithm() for p_i: *Optimistic**

```
1: $o_i \leftarrow p_i$ query its oracle
2: $m_i \leftarrow$ size of $\Pi_i$
3: $c \leftarrow$ correction parameter
4: if $m_i < c * o_i$ then  //  $\Pi_i$ is $incomplete_r$
5:     abort;
6: else                    //  $\Pi_i$ is $complete_r$
7:     $p_i$ selects $proposed_i$;
8: end if
```

Alg. 2 shows the psuedocode of the *compute* phase for the *optimistic** selection algorithm. Based on Alg. 2, process p_i which received a message from process p_j ($msg_j = \{p_j, \Pi_j\}$), updates its view to the union of its current view set (i.e. Π_i) and the view vector it received from process p_j (i.e. Π_j). Process p_i also updates $proposed_i$ to $proposed_j$ if $proposed_j$ is larger than $proposed_i$.

Alg. 3 shows the decision algorithm for the *optimistic** algorithm which is run by each process at the end of round R. According to Alg. 3, each process as p_i, at the end of round R, first queries its local oracle in order to receive the estimated number of processes in the system (i.e. o_i). Based on Alg. 3 there are two possible outcomes for a process: decide to *select its proposed value* or to *abort*. Process p_i decides to *select* $proposed_i$ if its view is *relatively complete*, otherwise it decides to *abort*.

Algorithm 4 Compute (msg_i) for p_i: *Pessimistic**

```
1: $m_i \leftarrow$ size of $\Pi_i$
2: for all $p_j$ such that $p_i$ has received $msg_j$ do
3:     if $r \neq R$ then
4:         if $proposed_i < proposed_j$ then
5:             $proposed_i \leftarrow proposed_j$;
6:         end if
7:         $\Pi_i \leftarrow \Pi_i \bigcup \Pi_j$;
8:     end if
9:     if $m_j \geq c * o_i$ then  // $\Pi_j$ is $complete_r$
10:        $C_i[j] \leftarrow 1$;
11:    end if
12: end for
13: $msg_i \leftarrow \{proposed_i, \Pi_i\}$;
```

Alg. 4 shows the compute phase for the *pessimistic** algorithm. At the end of each round *except for the last round*, all processes update their proposed value and their view vector. Additionally, each process at the end of *all* rounds, updates a

Algorithm 5 Decision_Algorithm() for p_i: *Pessimistic**

```
1: if $\Pi_i$ is $complete_r$ then // $m_i \geq c * o_i$
2:     if $C_i$ is $complete_r$ then // $C_i[j] = 1$ for all $p_j$
3:         $p_i$ selects $proposed_i$;
4:     else
5:         abort;
6:     end if
7: else
8:     abort;
9: end if
```

local bit-vector called the *confirmation vector* which is denoted by C_i for process p_i. The confirmation vector, C_i, is a local vector for process p_i which is used to indicate whether p_i has received a relatively complete view from a process p_j or not. For example, process p_i sets $C_i[j]$ to 1, if it has received a message from p_j indicating that v_j is *relatively complete* according to the oracle value of p_i, i.e. $m_j \geq c * o_i$.

Alg. 5 shows the description of the *pessimistic** algorithm. Process p_i with a *relatively incomplete* view vector, at the end of round R must decide to abort. On the other hand, process p_i with a *relatively complete* view pessimistically assumes that other processes do not have *relatively complete* views unless they confirm this at some point during the R rounds of execution. If process p_i does not receive such confirmations from all processes that it knows, it must decide to abort.

Algorithm 6 Compute(msg_i) for p_i: *Moderately Pessimistic**

```
1: for all $p_j$ such that $p_i$ has received $msg_j$ do
2:     if $r \neq R$ then
3:         if $proposed_i < proposed_j$ then
4:             $proposed_i \leftarrow proposed_j$;
5:         end if
6:         $\Pi_i \leftarrow \Pi_i \bigcup \Pi_j$;
7:     end if
8: end for
9: $msg_i \leftarrow \{proposed_i, \Pi_i\}$;
```

Algorithm 7 Decision_Algorithm() for p_i: *Moderately Pessimistic**

```
1: if $\Pi_i$ is $complete_r$ then
2:     if receives some $incomplete_r$ view in round $R$ then
3:         abort;
4:     else
5:         $p_i$ selects $proposed_i$;
6:     end if
7: else
8:     abort;
9: end if
```

Alg. 6 shows the description of the compute phase for the *moderately pessimistic** algorithm. When a process p_i receives a message from a process p_j, if $proposed_j > proposed_i$ it updates $proposed_i$ to $proposed_j$. Process p_i updates its proposed value and its view vector at the end of all rounds except for the last round (i.e., round R).

Alg. 7 shows the description of the *moderately pessimistic** algorithm. Process p_i running the *moderately pessimistic** algorithm decides to abort if its view is *relatively incomplete*. Otherwise it checks the second **if** statement given at line 2 (See Alg. 7). If p_i at round R, receives a message from a process p_j indicating that v_j is *relatively incomplete* (i.e. $m_j \geq c * o_i$), process p_i must abort, otherwise it selects its $proposed_i$. Process p_i disregards the lost messages in the last round and optimistically assumes a *relatively complete* view for the senders of the lost messages.

As mentioned before, we have three main possible outcomes for the selection algorithms: (i) *agreement on a value*, (ii) *agreement to abort*, (iii) *disagreement*. We have *agreement on a value* if at the end of the algorithm, *all* processes decide on the same value (i.e. same process identity). We have *agreement on abort* if *all* processes decide to abort. We have *disagreement* if some processes decide to abort and some processes decide to select a value. Also, as we described before, we classify *disagreement* into two main categories: *safe* and *unsafe* disagreement.

V. Probabilistic Analysis

In this section, we present several graphs showing how the probability of each outcome of the selection algorithms varies for different configurations of the system parameters, i.e. the number of rounds of the algorithm (R), the *correction* parameter (c), the values reported by the local oracles, the choice of the selection algorithm and the probability of message loss (Q). We calculate the probability of each outcome of the algorithm using a probabilistic model checker called PRISM [23]. We calculate the probabilities of agreement on the identity of a process (`AG`), agreement on abort (`AB`), *safe* disagreement (`SD`) and *unsafe* disagreement (`UD`). The sum of the probabilities of *safe* and *unsafe* disagreement implies the probability of disagreement and is denoted by the `DG` label.

We model a system of processes executing a selection algorithm by creating discrete-time Markov chain (dtmc) models of the system [24]. The dtmc class allows the specification of both the deterministic and the probabilistic transitions. For the probabilistic transitions, the choice of the next state is determined by a discrete probability distribution. Due to the problem of state space explosion for large systems modelled using model checkers [25], we are only able to perform exact verification of the selection algorithm for a system of at most three participating processes. For systems with larger number of processes, PRISM can estimate the outcome using simulation, with a defined tolerance and interval of confidence. Our results in the following includes both systems of three and four processes. Therefore, in order to keep the uniformity of the results for different systems, we compute all results using probabilistic simulation of PRISM.

A. Specification of properties

In order to analyse the PRISM models, it is necessary to identify the properties of the system model to be evaluated by the tool. We specify each outcome of the selection algorithms as properties. In order to define the properties, we use PCTL temporal logics which are used by PRISM property specification language. In the following, we present a formal description of each property. The set of participating processes in the algorithm is shown by S. We denote a proper subset of the system by S^*. S^* is a proper subset of S if it excludes at least one member of S, i.e. $S^* \subset S$, $S^* \neq \emptyset$.

Agreement on a leader: We have agreement on a leader if *all* processes in the system decide on the same leader.

$$\forall p_i \quad (p_i \in S : p_i \ selects \ p_l \ , \ p_l \in S) \tag{1}$$

Agreement on abort: We have agreement on abort if *all* processes decide to abort.

$$\forall p_i \quad (p_i \in S : p_i \ decides \ to \ abort) \tag{2}$$

Disagreement: We have disagreement if the processes in a proper subset of the system (i.e. S^*) decide on selecting a leader and others decide to abort.

$$\begin{aligned}\exists p_i \quad (p_i \in S^* \ : \ p_i \ selects \ p_x \ , \ p_x \in S) \quad \wedge \\ \exists p_j \quad (p_j \in S - S^* \ : \ p_j \ decides \ to \ abort)\end{aligned} \tag{3}$$

Safe disagreement: We have safe disagreement if the processes in a proper subset (S^*) of the system (S) , decide on a *live* process as their leader, while the remaining processes either decide to abort or decide on an *aborting* process as their leader.

$$\begin{aligned}\exists p_i \quad (p_i \in S^* \ : \ p_i \ selects \ p_l \ , \ p_l \in S \ , \ p_l \quad is \ live) \quad \wedge \\ \forall p_j \quad (\quad (p_j \in S - S^* \ : \ p_j \ decides \ to \ abort) \ \vee \\ (p_j \ selects \ p_x \ , \ p_x \in S \ , \ p_x \ decides \ to \ abort) \quad)\end{aligned} \tag{4}$$

Unsafe disagreement: We have unsafe disagreement if there are at least two processes deciding on two different *live* processes as their leaders, i.e. we have at least two live leader in the system.

$$\begin{aligned}\exists p_i, \exists p_j, \exists p_x, \exists p_y \quad (p_i, p_j, p_x, p_y \in S \wedge (p_i \neq p_j) \wedge (p_x \neq p_y) : \\ (p_i \ selects \ p_x, \ p_x \quad is \ live) \wedge (p_j \ selects \ p_y, \ p_y \ is \ live)\end{aligned} \tag{5}$$

In the following, we present some interesting results from the probabilistic verification of the selection algorithms against the specified properties.

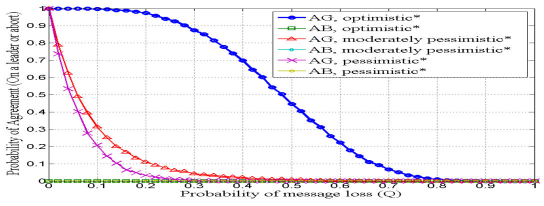
B. Varying the decision algorithm

Fig. 3 shows a comparison of the probability of each outcome of the *1-of-** selection algorithm for a system of four processes with unreliable oracles ($o_i = i$ for process p_i) as a function of Q, using different decision algorithms. As we see from the results, the *optimistic** algorithm shows the highest probabilities of agreement on a leader while it shows the highest probabilities of unsafe disagreement and the lowest safe disagreement probabilities. The *pessimistic** and the *moderately pessimistic** approach show lower probabilities of unsafe disagreement while they show higher probabilities of safe disagreement and agreements on aborts.

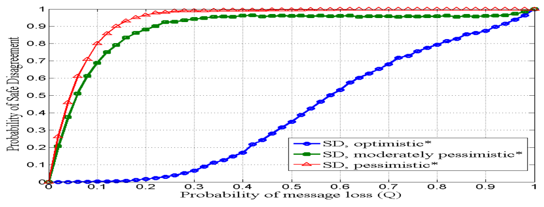
C. Varying the oracle values (o)

We present the results for different systems when the local oracles of some processes are underestimating the correct number of participants and assuming $c = 1$ [8]. We show how the probabilities of each outcome of the selection algorithms varies for different combinations of the oracle values, provided that we consider fixed values of R and c. For a system of n participants, assuming that the oracles do not overestimate the number of processes in the system (i.e. $o \leq n$), we have n^n number of possible combinations of the oracles values. For example, if there are four processes participating in the algorithm, the oracle value of each process can have four different values (i.e. $o_i \in \{1, 2, 3, 4\}$). So, we have $4^4 = 256$ possible combinations of the oracle values. Here, we only present the results for some interesting combinations of the oracle values.

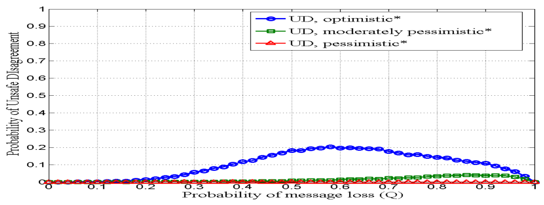
[8]We do not consider the settings of the system where the oracles are overestimating the number of participants since we know that with overestimating oracles and assuming $c \geq 1$, the processes will always abort.



(a) Agreement



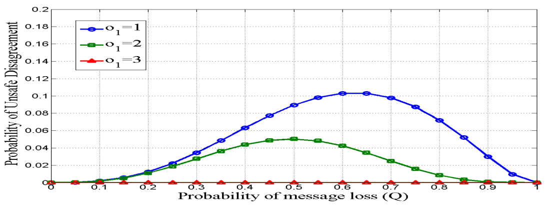
(b) Safe Disagreement



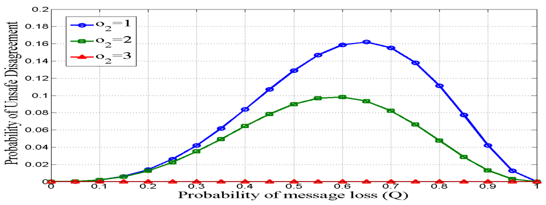
(c) Unsafe Disagreement

Fig. 3. A comparison of the probability of ***unsafe*** disagreement for the ***optimistic**** algorithm for a system of four processes as a function of Q varying the decision algorithm. $R = 2$, $o_i = i$ and $c = 1$.

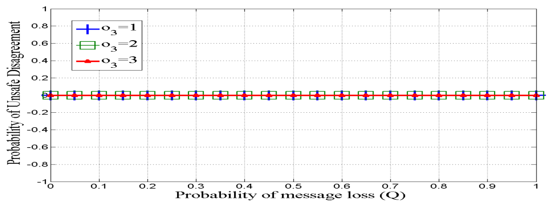
Fig. 4 shows the probability of unsafe disagreement for a system of three processes running the *optimistic** algorithm for two rounds. The results shown pertain the cases where the oracle values of two of the processes are correct while varying the oracle value of the third process. Based on the the maximum decision function the process with the highest ranking value must be elected as the leader. Therefore, in a system of three processes with the identities of p_1, p_2 and p_3, process p_3 has the highest ranking value and must be elected as the leader by all processes. Fig. 4(a) shows the results for three combinations of the oracle values where process p_2 and p_3 have always correct local oracles and the oracle value of process p_1 shows the values of 1, 2 and 3. As we see from the results, the closer the value of o_1 becomes to the correct value (i.e. 3), we have lower probabilities of unsafe disagreement. We have zero probabilities of `UD` for $o_1 = 3$ since all oracles for all processes show the correct value. We can observe the same trend in Fig. 4(b), for the cases where we vary the oracle value of process p_2. However, the `UD` probabilities are slightly higher for the cases where the oracle value of p_2 is incorrect



(a) $o_2 = 3$, $o_3 = 3$



(b) $o_1 = 3$, $o_3 = 3$



(c) $o_1 = 3$, $o_2 = 3$

Fig. 4. A comparison of the probability of ***unsafe*** disagreement for the ***optimistic**** algorithm for a system of three processes as a function of Q varying the oracle values. $R = 2$ and $c = 1$.

compare to the cases where the oracle value of p_1 is incorrect. This is because of the decision function, i.e. the maximum value. In the results shown in Fig. 4(b), for the cases where the p_2 only has p_1 and itself in its view set and $o_2 = 1$ or $o_2 = 2$, it decides on selecting itself while p_1 and p_3 either select p_3 or they abort, i.e. there are at least two cases of UD . On the other hand, for the setting shown in Fig. 4(a), when the view of p_1 shows $\{p_1, p_2\}$, for $o_1 = 1$ or 2, it decides to select p_2 while p_2 and p_3 with correct oracles either decide to abort or to select p_3. This means that p_1 selects an aborting process (p_2) as the leader that results in SD .

As we see in Fig. 4(c), for all values of o_3 and any probability of message loss (Q), provided that the oracle values of p_1 and p_2 are correct, the probability of unsafe disagreement is always zero. This is because using the *optimistic** algorithm, p_1 and p_2 with correct oracles will decide on selecting a leader only if they have the complete view of the system. So, p_1 and p_2 will only select a leader if they satisfy the decision condition, i.e. $m_1 \geq 3$ for p_1 and $m_2 \geq 3$ for p_2. This requires that they both have p_3 in their view set and in such a case as p_3 has the highest ranking value it will be selected by p_1 and p_2. On the other hand, process p_3 always includes its own identity in its view set which is the highest ranking value in the system. So, regardless of what information p_3 obtains from the system or which value its oracle provides, if it satisfies the decision condition (i.e. $m_3 \geq c * o_3$) it will always select itself as the leader. Therefore, there is no such a case that two or more processes decide on different leaders, i.e. there is not case of unsafe disagreement.

We can show that using the *moderately pessimistic** algorithm, we have the same trend as the ones for the *optimistic** algorithm. However, using the *moderately pessimistic** algorithm, we have lower probabilities of unsafe disagreement in general compared to the *optimistic** algorithm for the same system setting. This is because of the decision criterion defined for each algorithm which results in higher probabilities of agreement on abort for the *moderately pessimistic** algorithm compared to the *optimistic** one.

Based on our results for the same setting as given in Fig. 4 but for the *pessimistic** algorithm[9], the probability of UD is always zero for all values of Q, i.e. all disagreement cases are safe. This is because of the specified decision criterion for the *pessimistic** algorithm, where a process p_i satisfying the condition $m_i \geq \ c * o_i$ must also satisfy the following condition: All processes in p_i's view set such as process p_j must have at least $c * o_i$ number of processes in their view (i.e. $m_j \geq c * o_i$). We can show that in a system executing the *pessimistic** algorithm for any number of rounds, if there is only one process with an incorrect oracle value, the probability of unsafe disagreement is zero for all values of Q.

D. *Varying the correction parameter (c)*

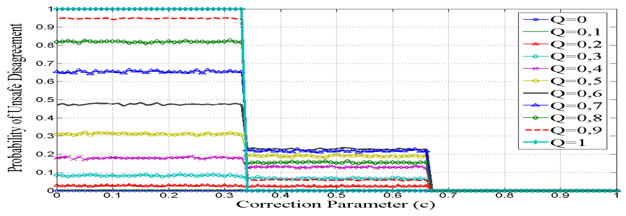
In this section, we show how different values of the *correction* parameter (c) affect the behaviour of the selection algorithm. For this, we assume that the local oracles of all processes show the correct number of the participants while we vary the value of c. The value of c is the same for all participating processes. As we explained in Section IV-A, using the *correction* parameter, a process intends to compensate for the unreliability of its local oracle.

Fig. 5 shows the probability of UD for a system of three participants executing the three selection algorithms for two rounds. In the given graphs, the x-axis shows the variant of the value of the correction parameter. As we see from the results, the UD probabilities as a function of c for different values of Q is a step function. From the given results in Fig. 5, we see that for all algorithms, the probability of UD declines considerably first at $c = 0.33$ and then at $c = 0.66$. We can show that for a system of four processes, for all algorithms, the probability of UD shows sudden decreases at $c = 0.25$, $c = 0.50$ and $c = 0.75$.

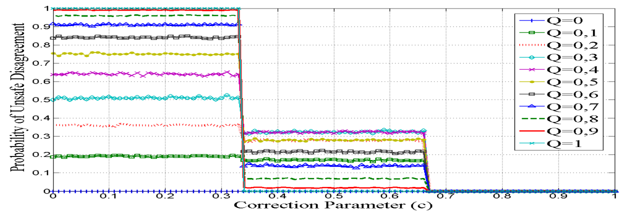
Finally, for a system of n participants with correct oracles, the curve for the probability of UD as a function of c and varying values of Q is a decreasing step function[10]. In other words, we have the same probability of UD for certain intervals of c and value of Q. We can show that for a system of n participants, the interval with the highest probability of UD is $0 \leq c \leq (1/n)$. The next interval with the second highest UD probability is $(1/n) < c \leq (2/n)$ and the third, the forth, .. n^{th} are respectively: $(2/n) < c \leq (3/n)$, $(3/n) < c \leq (4/n)$, .. $(n-1/n) < c \leq (n/n)$. This is because of the decision condition $m_i \geq c * o_i$. Since we assume correct oracles, we always have $o_i = n$. Therefore, the initial condition for a process to decide is to have at least $c * n$ number of processes

[9]Due to the lack of space, we only show a limited number of our results in this paper.

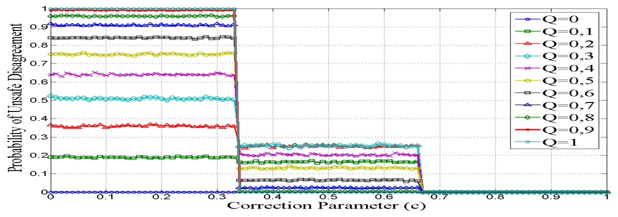
[10]Step function (or staircase function) is a function on the real numbers which can be written as a finite linear combination of indicator functions of intervals.



(a) *Optimistic**

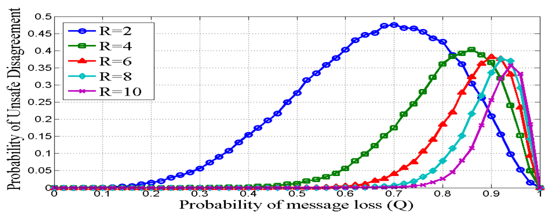


(b) *Moderately pessimistic**

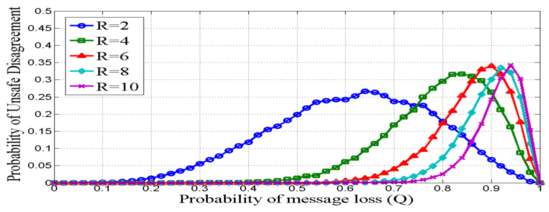


(c) *Pessimistic**

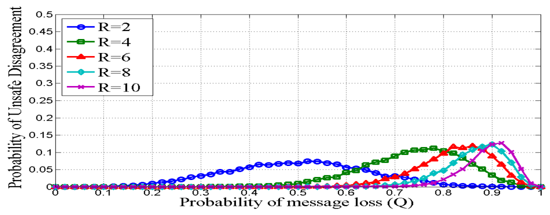
Fig. 5. The probability of ***unsafe*** disagreement for the three selection algorithms as a function of c for a system of three participants with varying Q and $R = 2$, $o_i = 3$



(a) *Optimistic**



(b) *Moderately pessimistic**



(c) *Pessimistic**

Fig. 6. A comparison of the probability of ***unsafe*** disagreement for the three selection algorithms as a function of Q for a system of four participants with varying R and $c = 0.5$, $o_i = 4$

in the system. If $c * n = 1$ (i.e. $c = 1/n$), all processes will always satisfy the first decision condition since all processes have at least their own identity in their view set. So, for $c \leq (1/n)$, we have the highest probabilities of UD . Similarly, for $c = 2/n$, the primary condition for a process to decide will be $m_i \geq (2/n) * n$, i.e. the process must have at least two processes in its view. With higher number of processes in the view of a process, it is less probable to make wrong decisions and therefore the probability of UD is lower for the intervals with the values of c closer to 1.

From the given results in 5, we also see that the probabilities of UD for different values of Q does not keep the same trend in different ranges of the value of c. For example, in Fig 5(a), for $0 \leq c \leq 0.25$ the probabilities of UD increase for larger values of Q while for the interval $0.26 \leq c \leq 0.51$ the probability of UD is the highest for $Q = 0.7$ while it is zero for $Q = 1$. This is because, with higher values of the probability of messages loss it is more probable for the processes to decide to abort due to lack of information and therefore lower probabilities of unsafe disagreement are expected. However, from the results in the interval $0.26 \leq c \leq 0.51$, the probability of UD does not decrease continuously with increasing the value of Q, i.e. the probability of UD as a function of Q shows a peak for the second and third intervals of the value of c. For a system of three processes, the UD probabilities show the same behaviour for the *moderately pessimistic** and *pessimistic** decision criteria when $0 \leq c \leq 0.25$, while for larger values of $c \geq 0.25$, the *pessimistic** approach shows a lower probabilities of UD compared to the *moderately pessimistic** approach.

E. Varying the number of rounds (R)

Fig. 6 shows a comparison of the probability of UD for a system of four participants with correct oracles and correction parameter of $c = 0.5$ executing the selection algorithms for different number of rounds. As we see from the results, increasing the number of rounds, does not show the same influence on the probability of UD for all algorithms. For the *optimistic** algorithm, increasing R results in lower UD probabilities in general while for the *pessimistic** and *moderately pessimistic** algorithms, with increasing the number of rounds we have higher peaks of UD probabilities. For all algorithms, with increasing the value of R, the peaks of the probabilities of UD occur at larger values of Q.

VI. Conclusion and Future Work

This paper describes the design and analysis of a family of distributed algorithms that aims to solve the problem of leader election among an initially unknown set of nodes, in the presence of unrestricted omission failures. In the proposed algorithms, each node is augmented with a local oracle that estimates, with a given confidence, the number of participating nodes in the leader election protocol. We consider a broad range of system configurations in the probabilistic analysis. The results show that under unrestricted communication failures and unreliable oracles, safety cannot be guaranteed if we wish to guarantee that some leaders are elected. Nevertheless, the probability of unsafe outcomes may be reduced through an adequate choice of the system parameters (i.e., R and c) as well as the right choice of the selection algorithm. Some of the main conclusions from the probabilistic analysis of the selection algorithm are as follows:

We have the highest probabilities of UD when the local oracles of the majority of the processes show the smallest possible value which is 1, i.e., the oracle of a process detects no other processes in the system. In such a system a process p_i satisfies the decision condition regardless of the number of processes in its view (since $c * o_i \leq 1$ and $m \geq 1$) which results in high probabilities of UD . Therefore, there is a need to stipulate an alternative decision condition for such cases. For example, if the oracle value of a process p_i shows the minimum value ($o_i = 1$) while process p_i could communicate with at least one other process in the system, it must abort and start over the leader election protocol. On the other hand, if process p_i is unable to communicate with any other processes in the system (i.e. $\Pi_i = \{p_i\}$) and its local oracle reports the value of one (i.e. $o_i = 1$), it decides to select itself. Such an approach also prevents the problem of starvation of a single vehicle in an intersection waiting for a green light.

Our results show that if there is only one process with an incorrect oracle value and that process has the highest ranking value, we have zero probability of unsafe disagreement (See for example Fig. 4(c)). This observation is helpful in introducing the policies for assigning ranking values to different processes. For example, in a VTL scenario, the vehicle with the most unreliable local oracle (due to the exposure to the NOLS conditions, e.g. high rise buildings, big truck.), or the vehicle with the highest speed or distance from an intersection should receive the highest ranking value. This vehicle will select itself as the VTL leader and therefore will order the red light to its own lane. Such a policy is in contrary to the one proposed in [1] where the closest vehicle to the intersection must be elected as the leader.

We analysed the behaviour of the selection algorithms for small systems of three and four participants as the core logic of a VTL leader election protocol. In a VTL, it is assumed that the participating vehicles are the cluster leaders of each lane of the corresponding intersection. Such an assumption drastically reduces the number of participants in a VTL leader election protocol to the number of the lanes in an intersection. Nevertheless, a probabilistic analysis of the selection algorithm for systems with larger number of participants is necessary to understand the behaviour of the selection algorithms in larger systems.

References

[1] M. Ferreira, R. Fernandes, H. Conceição, W. Viriyasitavat, and O. K. Tonguz, "Self-organized traffic control," in *Proceedings of the seventh ACM international workshop on VehicularAr InterNETworking*, ser. VANET '10. New York, NY, USA: ACM, 2010, pp. 85–90.

[2] N. Santoro and P. Widmayer, "Time is not a healer," in *STACS 89*, ser. Lecture Notes in Computer Science, B. Monien and R. Cori, Eds. Springer Berlin Heidelberg, 1989, vol. 349, pp. 304–313.

[3] N. Fathollahnejad, R. Pathan, E. Villani, R. Barbosa, and J. Karlsson, "On reliability analysis of leader election protocols for virtual traffic lights," in *in conjunction with the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks(DSN)*, June 2013.

[4] D. Cavin, Y. Sasson, and A. Schiper, "Consensus with unknown participants or fundamental self-organization," in *Ad-Hoc, Mobile, and Wireless Networks*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, vol. 3158, pp. 135–148.

[5] N. Fathollahnejad, E. Villani, R. Pathan, R. Barbosa, and J. Karlsson, "On probabilistic analysis of disagreement in synchronous consensus protocols," in *in Proceeding of Tenth EuropeanDependable Computing Conference (EDCC) 2014*, May 2014, pp. 23–34.

[6] H. Garcia-Molina, "Elections in a distributed computing system," *IEEE Transactions on Computers*, vol. C-31, no. 1, pp. 48–59, Jan 1982.

[7] N. A. Lynch, *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[8] S. Masum, A. Ali, and M.-y. Bhuiyan, "Asynchronous leader election in mobile ad hoc networks," in *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, vol. 2, April 2006.

[9] A. Derhab and N. Badache, "A self-stabilizing leader election algorithm in highly dynamic ad hoc mobile networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 7, pp. 926–939, 2008.

[10] S. Vasudevan, J. Kurose, and D. Towsley, "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*, Oct 2004, pp. 350–360.

[11] V. Raychoudhury, J. Cao, R. Niyogi, W. Wu, and Y. Lai, "Top-leader election in mobile ad hoc networks," *Pervasive and Mobile Computing*, vol. 13, pp. 181 – 202, 2014.

[12] C. Fetzer and F. Cristian, "A highly available local leader election service," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 603–618, 1999.

[13] N. Malpani, J. L. Welch, and N. Vaidya, "Leader election algorithms for mobile ad hoc networks," in *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, ser. DIALM '00. New York, NY, USA: ACM, 2000, pp. 96–103.

[14] J. Chalopin, E. Godard, and A. Naudin, "What do we need to know to elect in networks with unknown participants?" in *Structural Information and Communication Complexity*, ser. Lecture Notes in Computer Science, M. Halldorsson, Ed. Springer International Publishing, 2014, vol. 8576, pp. 279–294.

[15] L. Arantes, F. Greve, P. Sens, and V. Simon, "Eventual Leader Election in Evolving Mobile Networks," in *OPODIS 2013 - 17th International Conference Principles of Distributed Systems*, ser. Lecture Notes in Computer Science, vol. 8304. Nice, France: Springer, Dec. 2013, pp. 23–37.

[16] M. Raynal and F. Tronel, "Group membership failure detection: a simple protocol and its probabilistic analysis," *Distributed Systems Engineering*, vol. 6, no. 3, p. 95, 1999.

[17] H. Duggal, M. Cukier, and W. Sanders, "Probabilistic verification of a synchronous round-based consensus protocol," in *Proceeding of the Sixteenth Symposium on Reliable Distributed Systems.*, oct 1997, pp. 165 –174.

[18] O. K. Tonguz, "Biologically inspired solutions to fundamental transportation problems," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 106–115, 2011.

[19] J. B. Kenney, "Dedicated short-range communications (dsrc) standards in the united states," *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1162–1182, July 2011.

[20] J. Du, J. Masters, and M. Barth, "Lane-level positioning for in-vehicle navigation and automated vehicle location (avl) systems," in *Intelligent Transportation Systems, 2004. Proceedings. The 7th International IEEE Conference on*, Oct 2004, pp. 35–40.

[21] A. Giridhar and P. Kumar, "Distributed clock synchronization over wireless networks: Algorithms and analysis," in *Decision and Control, 2006 45th IEEE Conference on*, Dec 2006, pp. 4915–4920.

[22] A. Ebner, L. Wischhof, and H. Rohling, "Aspects of decentralized time synchronization in vehicular ad hoc networks," in *Proceedings of the 1st International Workshop on Intelligent Transportation (WIT 2004*.

[23] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: probabilistic model checking for performance and reliability analysis," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 40–45, Mar. 2009.

[24] J. R. Norris, *Markov chains*. Cambridge university press, 1998, no. 2.

[25] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, *Model Checking and the State Explosion Problem*, Berlin, Heidelberg, 2012, pp. 1–30.