Thesis for the Degree of Doctor Of Philosophy

On the Design and Analysis of Consensus Protocols for Vehicular Ad Hoc Networks

Negin Fathollahnejad Asl

Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden 2017

On the Design and Analysis of Consensus Protocols for Vehicular Ad Hoc Networks

Negin Fathollahnejad Asl

Göteborg, Sweden 2017
ISBN: 978-91-7597-619-8

Doktorsavhandlingar vid
Chalmers Tekniska Högskola
Ny serie Nr 4300
ISSN 0346-718X

Contact Information:
Dependable Real-Time Systems Group
Division of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
Phone: +46 (0)31-772 1000
http://www.chalmers.se/cse/
Author e-mail: `negin1361@gmail.com`

Printed by Chalmers Reproservice
Göteborg, Sweden 2017

To my daughter Shahrzad

On the Design and Analysis of Consensus Protocols for Vehicular Ad Hoc Networks

Negin Fathollahnejad Asl
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract Vehicle-to-vehicle communication technologies support diverse cooperative applications for intelligent transportation systems to increase safety and fuel efficiency of road vehicles. Vehicles participating in a cooperative application are expected to make coordinated and mutually consistent decisions. To ensure consistency, it is often essential that the participating vehicles reach agreement on the data they use as a basis for these decisions. This thesis deals with the fundamental problem of reaching agreement on a value, or a set of values, in a distributed system in the presence of unrestricted communication failures. It is known from the literature that this problem is impossible to solve perfectly, i.e., no matter what algorithm we use there is always a non-zero probability of *disagreement*. Hence, our aim is to design algorithms that minimize the probability of disagreement. We propose and analyse several agreement algorithms to solve three fundamental consensus problems. These algorithms are distinguished by their *decision criterion*, which determine whether a computer should *decide on a value* or *decide to abort*. Our analyses show that the probability of disagreement depends strongly on the number of computers in the system, the number of rounds of message exchange, the choice of decision criterion, as well as the probability of message loss. We identify two types of disagreement, safe and unsafe disagreement, and show that unsafe disagreement can be avoided if all computers know the number of computers in the system.

Keywords: Agreement Algorithms; Probabilistic Analysis; Consensus; Vehicular Ad-Hoc Networks; Communication Failure; Intelligent Transportation Systems

Acknowledgments

First of all, I would gratefully and sincerely thank my supervisor Professor Johan Karlsson for his inspiring guidance, rich experience and sustained encouragement.

I especially want to thank my former colleagues and advisers, Dr. Raul Barbosa, Dr. Risat Pathan and Dr. Emilia Villani for the excellent collaboration we had in the papers as well as many fruitful discussions. I would like to show my appreciation to Professor Marina Papatriantafilou, Professor Wolfgang Ahrendt and Professor Jan Jonsson for the regular follow-up meetings to discuss the direction of my studies.

I am grateful for the financial support provided by the Swedish National Automotive Research and Innovation Programme (FFI) and VINNOVA through the DFEA2020 project.

Special thanks go to all the people at the Department of Computer Science and Engineering for providing a friendly and productive environment. I thank my dear friends in Sweden for their kind company, support and encouragements during my studies. I want to thank my family in Iran, my beloved father, mother and brother for their unwavering love, support and encouragement over the years.

Finally, I want to thank my loving family in Sweden, Jean, for always being there for me, his support and his love. And Shahrzad, our little daughter who brought joy, hope and happiness to our life.

Negin Fathollah Nejad Asl
August 21, 2017

List of Papers

Parts of the work presented in this thesis have previously been published in the following conference papers:

I **Negin Fathollahnejad**, Raul Barbosa, Johan Karlsson, *"A Probabilistic Analysis of a Leader Election Protocol for Virtual Traffic Lights"*, in *the 22nd IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 311-320*, Christchurch, New Zealand, 22-25 January, 2017.

II **Negin Fathollahnejad**, Risat Pathan, Johan Karlsson, *"On the Probability of Unsafe Disagreement in Group Formation Algorithms for Vehicular Ad hoc Networks"*, in *the 11th European Dependable Computing Conference (EDCC)*, Paris, France, September 7-11, 2015.

III **Negin Fathollahnejad**, Emilia Villani, Risat Pathan, Raul Barbosa, Johan Karlsson, *"On probabilistic analysis of disagreement in synchronous consensus protocols"*, in *the 10th European Dependable Computing Conference (EDCC)*, Newcastle, UK, May 13-16, 2014.

IV **Negin Fathollahnejad**, Emilia Villani, Risat Pathan, Raul Barbosa, Johan Karlsson, *"Probabilistic analysis of a 1-of-n selection algorithm using a moderately pessimistic decision criterion"*, in *the*

19th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), Vancouver, Canada 2-4 December, 2013.

V Emilia Villani, **Negin Fathollahnejad**, Risat Pathan, Raul Barbosa, Johan Karlsson, *"Reliability Analysis of Consensus in Cooperative Transport Systems"* in *Proceedings of the 2nd Workshop on Architecting Safety in Collaborative Mobile Systems (ASCoMS)*, September 2013, Toulouse, France.

VI **Negin Fathollahnejad**, Emilia Villani, Risat Pathan, Raul Barbosa, Johan Karlsson, *"On Reliability Analysis of Leader Election Protocols for Virtual Traffic Lights"*, in *43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSNW), pp. 1-12*, Budapest, Hungry, 24-27 June, 2013.

Acronyms

AB	Agreement to abort
AG	Agreement on a value
DG	Disagreement
GF	Group Formation
LEP	Leader Election Protocol
MP	Moderately Pessimistic
NLOS	No Line Of Sight
OP	Optimistic
PS	Pessimistic
SD	Safe Disagreement
UD	Unsafe Disagreement
UAV	Unmanned Air Vehicles
VANET	Vehicle Ad Hoc Networks
VTL	Virtual Traffic Lights

Contents

List of Figures

1 Introduction

Recent advances in wireless technologies for vehicle-to-vehicle communication have made it possible to develop cooperative applications to improve traffic safety and fuel efficiency of road vehicles. Examples of such applications include real-time traffic management systems [43], vehicle platooning [10] and virtual traffic lights (VTL) [25].

The applications for real-time traffic management systems rely on the wireless technologies for data dissemination among road vehicles. The communicated data dynamically provide the vehicles with the traffic information such as road conditions, traffic congestions and collision alerts. These systems are currently used to support safe driving by providing dynamic route scheduling, emergency message dissemination, traffic condition monitoring, etc.

Vehicle platooning is a method to increase the capacity of the roads by driving a group of vehicles in close distance to each other. The cooperative vehicles in a platoon are travelling in a single lane following a leading vehicle at the head of the lane [8]. The vehicles are fully automated relying on both local sensors, and inter-vehicle communication to coordinate speed, braking and acceleration.

The concept of a virtual traffic light (VTL) was proposed in 2010 by Ferreira et al. as a self-organizing traffic control system that allows road vehicles passing an intersection to implement the function of a traffic light without the presence of a roadside infrastructure [25]. In a VTL system, the vehicles approaching an intersection interact via wireless communications to elect a vehicle among themselves as the VTL leader. The VTL leader is responsible to take control of the traffic lights for the intersection temporarily for a short period of time.

The development of applications for automotive cooperative systems comprises many challenges, one of which is to provide reliable and efficient means for the cooperating vehicles to make coordinated decisions. Electing a leader in a VTL is an example of a coordinated decision, where it is essential for the participating vehicles to reach consensus, i.e., they must agree on the same leader. If they disagree and appoint more than one VTL leader, then the different leaders may send conflicting traffic light signals, which could lead to serious accidents.

Leader election is a distributed consensus problem. In such problems, the aim is to reach agreement on a value (or set of values) among a group of distributed computers, or nodes, that exchange messages over a communication network. Designing distributed agreement algorithms that ensures consensus in the presence of failures has proven to be a challenging task. Consensus problems has therefore attracted much attention from the distributed systems research community over last forty years.

The consensus problem was first introduced in 1978 by Gray in order to model the problem of *distributed commit*. In this problem, a group of processes are to reach agreement on whether to commit or abort a

distributed transaction [30].

Solving consensus problems is straightforward under ideal conditions when there are no *uncertainties* in the system. However, uncertainties must be considered in the design of distributed agreement algorithms, since they cannot be avoided in real systems. Node failures and communication failures are examples of such uncertainties. Another example is uncertainties related to the amount of knowledge the nodes have about the system, e.g., concerning the system size (the number of nodes in the system) and the network topology [38].

The possibilities to solve a consensus problem is also affected by the *timing* model (the level of synchrony among the nodes), and the nature of the failures. Two commonly used timing models are the (fully) *synchronous* model and the *asynchronous* model [38]. In the *synchronous* model, it is assumed that nodes execute distributed algorithms in synchronous rounds[1]. This assumption removes any uncertainty about the order in which different events occurs, such as the reception and transmission of messages. In the *asynchronous* model, processes take steps at arbitrary relative speeds and orders. There are no timing bounds on the delivery of a message in an asynchronous system.[2].

Distributed algorithms can be designed to cope with a variety of process and communication failures. These failures can be divided into two major classes: *symmetric* and *asymmetric* failures. A symmetric failure is one which is perceived in the same way by all non-faulty nodes, while an asymmetric failure is perceived differently by different non-faulty nodes in the system.

Examples of commonly used models for process failures include *value failure* (a process delivers an incorrect output that the user believe is correct), *stopping failure*, or *fail-stop failure* (a process stops delivering

[1] At each round, each process sends messages to other processes, receives messages and performs computations of the received messages. The messages sent in a round must be received at the same round by the receiving processes; otherwise they are considered to be lost. (See for example [9])

[2] The authors of [27] prove that for an asynchronous system, it is impossible to design a deterministic consensus algorithm even with only one process failure.

outputs instantaneously, without prior notification), and *timing failure* (a process delivers an output too late, or too early). A stopping failure is symmetric since all processes perceives the failure in the same way - the absence of an output. A value failure, on the other hand, can be both symmetric (all non-faulty nodes receive the same erroneous output) and asymmetric (at least two nodes receive two different outputs, of which at least one is incorrect).

Common models for communication failures include *send omissions* (the transmission of a message fails), and *receive omissions* (the reception of message fails at some, but not all, of the intended recipients). A send omission is a symmetric failure since no node receives the message, whereas a receive omission is an asymmetric failure since some, but not all, nodes receive the message.

Previous research has shown that consensus problems often are impossible to solve for both synchronous and asynchronous systems. For example, the authors of [27] prove that no deterministic algorithm can solve the agreement problem in an asynchronous system, not even in the case where only a single node exhibits a stopping failure. We also know that no algorithm can guarantee consensus in a synchronous system if there is no upper bound on the number of messages that can be lost during the execution of the algorithm. In 1989, N. Santoro and P. Widmayer showed that, in a synchronous system of n processes, any non-trivial form of agreement is impossible to solve, if $n-1$ or more messages are lost per communication round [52].

This thesis addresses the problem of designing distributed agreement algorithms for systems that rely on wireless networks for data communication. The main motivation for our work is the increasing interest in developing automotive cooperative systems to improve road safety and reduce traffic congestion. However, since our work is of a fundamental and theoretical nature, we believe it is relevant also for other applications domains, such as self-organizing systems based on robots or unmanned air vehicles.

Although agreement problems have been proved to be solvable under different system models and failure assumptions, such as in [7, 38, 48], we still lack definite answers to how such problems are best solved in distributed systems that rely on wireless networks. A main challenge in designing agreement algorithms for systems that use vehicle-to-vehicle (V2V) technology is that the communication channels in these systems can be subject to disturbances of widely different durations and magnitudes. Consequently, we cannot make any assumptions about the number of messages that can be lost during the execution of an agreement algorithm in these systems.

We have chosen to use the synchronous system model in our studies of agreement algorithms. Automotive cooperative systems are hard real-time systems, and the synchronous model provides a reasonable, but not perfect, model of such systems. Another reason for using the synchronous model is that we want to focus our studies on the impact of communication failures, and we therefore want to exclude uncertainties related to a lack of synchronization among processes in our studies.

We focus our analyses on the impact of send omissions (symmetrical message losses) and receive omission (asymmetric message losses), since they are likely to be the dominating type of communication failures in V2V networks. Since we know it is impossible to design a distributed agreement algorithm that can guarantee consensus in the presence of an arbitrary number of message losses, our work focuses entirely on probabilistic analysis of agreement algorithms.

The thesis addresses three fundamental consensus problems, which we call *1-of-n selection*, *1-of-* selection* and *group formation*. We propose and analyse several agreement algorithms to solve these problems. An important feature of our algorithms is that they allow a process to abort, i.e., a process may choose not to decide on a value. The decision whether to abort or to decide on a value is made by a decision criteria executed as the final step in all our algorithms. A key contribution of the thesis is that we investigate how different decision criteria affects the probability

of disagreement, in the presence of any number of message losses, for both send and receive omissions.

The thesis specifically addresses the following research questions:

- How do we calculate the probability of disagreement among the nodes of a cooperative system in the presence of an arbitrary number of message losses?
- How does the algorithm's decision criterion influence the probability of disagreement?
- How does the system model and the communication failure model affect the outcome of a consensus algorithm?
- How does the number of rounds of execution and the number of participating nodes in the algorithm affect the probability of disagreement?

As already mentioned, we design and analyse three main families of synchronous round-based consensus algorithms with the aim of solving three consensus problems: *1-of-n* selection, group formation and the *1-of-** selection. In the following sections, we briefly explain each of these problems.

1.1 The *1-of-n* Selection Problem

In the *1-of-n selection* problem, n nodes (or processes) are to reach agreement on one value. Each node proposes one value, and the processes must agree to select one of the proposed values, in the presence of an unrestricted number of communication failures. We propose a family of round-based consensus algorithms to solve the problem of *1-of-n selection*. We refer to these algorithms as *1-of-n selection* algorithms. These algorithms can be used as the core logic of a protocol for leader election in a cooperative application.

We consider a system of n processes which execute the algorithm in a fixed number of rounds, denoted by R. After R rounds of message exchange, each process decides on an outcome. There are two main outcomes for a process: it decides either to *select a value*, or to *abort*.

The decision of a process depends on the *decision criterion* specified for the *1-of-n* algorithm. A decision criterion contains a set of logical expressions which decide whether a process should select a value or abort due to the lack of information concerning the status of other processes. We introduce three different decision criteria for the *1-of-n* selection algorithms, called the *optimistic*, *pessimistic* and the *moderately pessimistic* decision criterion. These decision criteria are described in detail in Chapter 4.

Considering the outcomes of all processes in the system, a *1-of-n* algorithm can have three main outcomes, namely (i) *agreement on a value*, (ii) *agreement on abort* and (iii) *disagreement*. *Agreement* implies that all nodes select the same value, or all nodes decide to abort, while *disagreement* happens if some nodes decide to select a value and others decide to abort.

In the design of the *1-of-n* selection algorithm, we restrict ourselves to algorithms where the nodes never decide on a value unless they have access to the proposed values from all participating nodes. Thus, we assume that each node knows the number and the identity of the participating nodes, i.e. n is known. This simplifying assumption ensures that even in the presence of an arbitrary number of lost messages, two processes never select different values. Therefore, the only kind of disagreement that can occur is when some processes decide to select the same value and the remaining processes decide to abort.

1.2 The Group Formation Problem

In a self-organized cooperative applications, such as a virtual traffic light (VTL), it is unrealistic to assume that all nodes initially know the ex-

act set of nodes that are involved in the decision process. This is due to several factors, such as the high mobility of the nodes in such environments, or the existence of massive communication failures which can result in partitioning the system into several networks. For example, in a VTL system, a vehicle approaching an intersection may not be aware of all other vehicles that approach the intersection [35]. Therefore, another consensus problem that needs to be solved for cooperative applications is the problem of reaching agreement on *a set of nodes* that are involved in bootstrapping the application. We call this problem the group formation problem, since it involves agreeing on the group of nodes that are the initial participants, or members, of a cooperative application.

We propose a *group formation* algorithm that aims to solve the problem of reaching agreement on a group of nodes' identities in a synchronous system with unbounded number of message losses. We know that due to the Santoro and Widmayer's impossibility result, it is impossible to design a group formation algorithm that can guarantee agreement on a group of nodes under the given communication failure model.

Our proposed group formation algorithm has two outcomes at the process level. Each process that executes the algorithm will either decide to *select a group* (i.e. a set of nodes), or decide to *abort*. At the system level, i.e., when we consider the outcomes of all participating nodes, the algorithm have three main outcomes: (i) *agreement on a set of nodes*, (ii) *agreement to abort*, (iii) *disagreement*. We categorize *disagreement* in two classes: *unsafe disagreement* and *safe disagreement*. In case of *safe* disagreement, one subset of the nodes decides on the same set of nodes while the remaining nodes decide to abort. In case of *unsafe* disagreement, at least two different subsets of the nodes decide on different sets.

In order to reduce the probability of *unsafe disagreement*, we suggest different decision criteria for the group formation algorithms which rely on the use of an extra component, called an *oracle*. The oracles are local devices attached to each process, and they are responsible for providing

each process with an estimation of the number of processes (nodes) in the system [3]. The decision of a process executing a group formation algorithm depends on its *view*[4] of the system after R rounds of message exchange, as well as the system size estimated by the oracle.

1.3 The *1-of-** Selection Problem

Finally, 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) [11]. 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 family of consensus algorithms to solve the problem of *1-of-** selection to be used as the core logic of a leader election protocol in a self-organized cooperative system such as a VTL application. The goal of these algorithms is that a group of processes agree on the identity of one of the processes among themselves to be the leader of the system.

We assume that the processes are initially unaware of the number and the identities of other processes in the system. However, similar to the design of the group formation algorithm, we consider the existence of a local oracle for each node to have an estimation of the number of participants in the algorithm.

We propose a family of round-based consensus algorithms with three different decision criteria to solve the *1-of-** selection problem, called

[3] We assume that the oracles are unreliable and may underestimate or overestimate the actual number of the nodes in the system.

[4] We define the view of a process as the set of processes it sees in the system either directly by receiving messages from them or indirectly through the views of other processes.

the *optimistic**, *pessimistic** and the *moderately pessimistic** decision criterion[5]. Depending on the choice of the decision criterion, each process executing the algorithm either decides on *electing a leader* or *abort*.

At the system level, we can have three different outcomes of the consensus algorithm: *agreement on a value*, *agreement to abort* and *disagreement*. We have agreement among the processes if all processes decide on the same leader or if all processes decide to abort. Disagreement occurs if some processes decide on electing a leader and some decide to abort. We define two classes of disagreement cases: *safe disagreement* and *unsafe disagreement*. However, we introduce different definitions of the safe and unsafe disagreement cases compare to the ones we introduced for the group formation algorithms. The new classification of the disagreement cases rely on the introduction of two different types of processes in the system: a *live* process and an *aborting* process. A *live* process is a process which, at the end of the algorithm, has decided to elect a leader while an *aborting* process is a process which has decided to abort.

Unsafe disagreement occurs if at least two processes decide on two different live processes as their leaders. We have *safe disagreement* if the processes in a proper subset of the system[6], 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.

1.4 Thesis Structure

The remainder of the thesis is organized as follows. In Chapter 2, we discuss related work. In Chapter 3, we provide an overview the main design principles of a virtual traffic light. In Chapter 4, we present the design and analysis of the *1-of-n selection* algorithms. The design and the analysis the group formation algorithms is presented in Chapter 5. Chapter 6 presents the design and analysis of the *1-of-* selection* algo-

[5]Details on the description of each of these decision criteria are given in Chapter 6

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

rithms, and explains how they can be used as the core logic of a VTL leader election algorithm. In Chapter 7, we provide discussions and elaborate on two problems that require further research. Finally, we present our conclusions and suggestions for future work in Chapter 8.

2

Related Work

The consensus problem has been investigated widely in the area of distributed computing and is proved to be solvable under different failure assumptions such as in [7, 38, 46, 48]. However, most previous research were based on different classes of node failures only with assuming reliable communication links among the nodes. For simplifications, the communication failures have been mostly associated to the failure of the nodes rather than being investigated explicitly as an independent phenomenon, e.g. [27, 37].

Such simplifying assumptions are unrealistic and may lead to incorrect characterizations of a system. For example, if in a system the loss of a single message due to a transient communication failure is inscribed to a faulty behaviour of the sending or receiving process, we may reach to

incorrect conclusions such as considering the entire nodes in the system to be faulty.

Some other researchers use perception-based hybrid failure models in which the sender-caused link faults are considered as the failure of the sending node while the link fault term is used to denote the receiver-caused faults [55]. Such failure models might also lead to undesirable conclusions for a system.

On the other hand, for the systems based on highly unpredictable wireless environments, it is important to consider the communication failures explicitly in order to assure a specific degree of dependability and safety of critical distributed systems.

In this work, our goal is to address the consensus problem for synchronous systems which are subjected to unrestricted communication failures. In such a model, the communications failures can occur on any link at any time while there are no limitations on the number or pattern of the lost messages. Our failure model is based on the model introduced by Santoro and Widmayer in [52] denoted as the *transmission fault model*.

In [52], Santoro and Widmayer show that any non-trivial form of agreement is impossible to solve if $n-1$ or more messages can be lost per communication round in a system of n processes. The given impossibility result is a generalization of the results presented by Akkoyulunu et al. [2] and the results given in [31]. The authors of [31] show that there is no deterministic solution to the consensus problem between two processes with unreliable communication links.

There are a large number of methods suggested in literature to circumvent the impossibility result in synchronous consensus systems with dynamic omission faults. In [53], Santoro and Widmayer provide an extensive map of possible and impossible computations for a synchronous system of n processors in the presence of transmission faults. The objective of a computation is defined as either to compute a non-constant function on all inputs, or to reach an agreement among the processes on a value. The authors in [53] characterize the maximum number of

transmission faults per clock cycle that can be tolerated for the computation of arbitrary or specific functions, with several types of faults. Later in [54], Santoro and Widmayer define bounds on the number of dynamic faults and express the connectivity requirements to achieve any non-trivial agreement.

The authors of [15] propose the use of collision detectors augmented to the nodes in an unreliable wireless network where the messages can be lost due to collisions, electromagnetic interferences or other anomalies. They introduce a new classification of the collision detectors to circumvent the Santoro and Widmayer's impossibility result. The collision detectors are classified based on their ability to detect actual collisions and their ability to report only actual collisions (no false positives), so called the completeness and accuracy. Then, for each class of collision-detector they show how to solve consensus and provide matching lower bounds.

In [38], a lower bound of $\frac{1}{r+1}$ is introduced for the probability of disagreement for any *r-round* algorithm to solve the randomized coordinated attack problem in the presence of unbounded number of message losses. The coordinated attack problem is the challenge of coordinating an action by processes communicating over an unreliable link. In the randomized version of the problem the processes flip coins to decide what to do.

The authors of [4] solve the consensus problem for sparse MANETs[1] with density values larger than DTN (*Delay tolerant networking*). These MANETs are assumed to keep distinct subsets of nodes connected at distinct intervals.

In [1], Afek et al. employed randomization techniques to solve the *k-consensus* problem in the presence of communication failures. In a *k-consensus* problem, at least k processes among n processes decide on the same value such that $k > n/2$. They show that the safety properties of consensus (i.e., validity and agreement) are ensured in the presence of

[1]The mobile ad hoc networks or MANETs are self-configuring, infrastructure-less networks of mobile devices which are connected using wireless channels.

even unrestricted communication failures. But, in order to satisfy the liveness property of the consensus algorithm the number of faults in a round should be restricted. The approach proposed in this thesis does not restrict the number of faults in each round, while assuring liveness, at the expense of having a certain probability of disagreement.

Most of the suggested methods in the literature take a preventive approach toward the impossibility result for consensus in the presence of unrestricted communication failures. The given techniques mostly rely on restricting the communication failure patterns or limiting the number of failures in a round.

Nevertheless, it is possible to design protocols that have a low probability of failing to reach consensus, 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 [49].

The authors in [49], perform a probabilistic analysis of a group membership algorithm to show that it is possible to tune the algorithm parameters so as to reduce the probability of false group failure detections. They consider an asynchronous system of a finite set of n processes using a reliable FIFO channel. The processes are assumed to be faulty. Based on the FLP impossibility result [27] for asynchronous systems, it is not possible to design a deterministic consensus algorithm for such a system even with only one process failure. Moreover, due to the FLP impossibility result, it is not possible to ensure both the liveness and the safety for such a protocol and therefore, they have considered a protocol that trades liveness against safety.

The authors in [20], present a new approach in the probabilistic verification of synchronous consensus protocols. They make stochastic assumptions about the system and failure models, and verify the probability of transition into an incorrect state for a simple variant of the Byzantine Generals (BG) protocol in the presence of failures. In their analysis, they consider the settings of system parameter for which the probability

of having valid valued consensus is at least 0.999999. According to their results, varying the probability of omission failure (denoted by q), the maximum permissible value of q that allows the consensus value to be achieved is 0.00417.

Our goal is to design decision making algorithms to run on top of a simple consensus protocol with the main purpose of minimizing the probability of failing to reach consensus. We evaluate and compare the effectiveness of different decision algorithms by means of using probabilistic model checking tools as well as deriving closed-form expressions to calculate the probability of disagreement among processes. Our results may be applied also for on-line verification and adaptation to cope with variable probabilities of communication failures.

Our work focuses on the probabilistic analysis of round-based consensus protocols in which processes communicate in rounds of message exchange in order to decide on a consistent output [48]. Our system model is inspired by a general computational model named as the *heard-of* model. This model is introduced by Schiper and Charron-Bost [13] and is used to specify systems with any type of benign failures.

3

Virtual Traffic Lights

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 [25], [56]. Ferreira et al. in [25] propose a VTL scheme which relies on two main procedures: *leader election* and *leader handover*.

Leader election is the process of electing a vehicle as the VTL leader among a group of vehicles that are about to pass an intersection. The VTL leader takes upon itself to serve as a traffic controller for the intersection for a limited amount of time, called the *control period*. During the control period, the VTL leader assumes a red light for itself while it broadcasts traffic light signals to other vehicles in the intersection. At the end of the control period, the vehicle hands over the leadership to

another vehicle, and then waits to receive a green light from the new leader (*leader handover*). If no other vehicle is present in the intersection at the end of the control period, the leader announces its resignation as the controller and then gives a green light to itself. Vehicles approaching an intersection where no leader is present must execute a leader election protocol to appoint a new VTL leader.

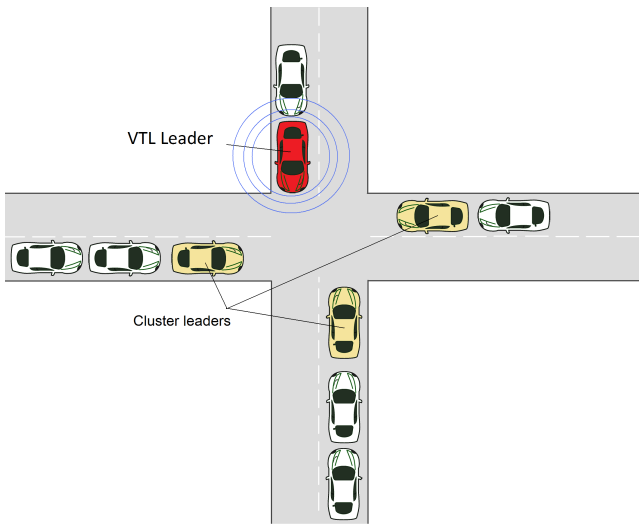


Figure 3.1: *An example of virtual traffic lights*

Fig. 3.1 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[1]. It is assumed that only the cluster leaders participate in the VTL leader election protocol. Leader election involves the execution of an agreement algorithm whose purpose is to ensure that the participating nodes (the cluster leaders) reach consensus on which car they appoint as VTL leader.

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

A vehicle approaching an intersection must first determine if a leader is present. If so, it simply follows the orders sent by the leader. If the leader orders a red light for the lane in which the vehicle is travelling, it must stop and be prepared to participate in a leader handover operation and assume the role as leader. In case an approaching vehicle does not detect a leader, it must announce its presence by broadcasting an invitation to other vehicles to participate in a leader election procedure.

The VTL system described in [25] is based on the assumptions that all road vehicles are equipped with a GPS system and use the same digital roadmap. To allow the identification of intersections where VTLs can be created, the authors propose to use beaconing features provided by VANET geographical routing protocols. They assume that each vehicle maintains a real-time database of the location of all vehicles in its vicinity. This database is constantly updated through the reception of new beacons. The GPS devices are assumed to have an accuracy of 10-20 meters, which is sufficient to identify intersections where a VTL can be created.

The introduction of virtual traffic lights is likely to bring several potential benefits to society. As highlighted by the authors of [25], these benefits include improved traffic safety, mitigation of traffic congestion, and significant cost savings for construction and maintenance of roads. The positive impact on traffic safety is obvious. The introduction of virtual traffic lights can potentially bring traffic control to all road junctions across the globe at a very low price.

To determine the current ratio of road junctions equipped with conventional traffic lights in the United States, the authors of [25] conducted a study of junctions in 3138 of the nation's 3234 counties, which showed that 0.5% of the junctions are equipped with traffic lights. For urban areas, they report a corresponding ratio of 24% for the five boroughs of New York City, and 25% for the down town area of Dublin, Ireland. The corresponding ratio for Portugal's second city Porto is reported to be 16%. These numbers serve as an indication of the potential for virtual

traffic lights to improve traffic safety by bringing traffic control to all intersections and junction.

While conventional traffic lights may have a negative impact on traffic flow, virtual traffic lights promises to reduce and mitigate traffic congestion. The authors of [25] present a simulation study of how the traffic flow of all roads in the city of Porto would be affected by the introduction of VTL technology. It shows that VTL:s would increase traffic flow with around 20% for low traffic densities, and with more than 60% for high traffic densities, compared to the current situation, where 328 out of 2000 of the city's junctions are equipped with conventional traffic lights.

The authors of [25] also provide some examples of the potential for cost reductions based on data taken from [45]. For example, the cost of installing a traffic light in the United States ranges from $50.000 to $200.000 depending on the complexity of the intersection, and the annual average cost of operating a traffic light is in the order of $3000. Hence, they estimate that the total deployment cost of all traffic lights in the U.S. represent a value of $33 billion, while the annual total cost of maintaining traffic lights is in order of $780 millions. (Note that the cost figures mentioned in this paragraph dates back to 2007 when [45] was published.)

Thus, the authors of [25] provide compelling arguments for the benefits of introducing virtual traffic lights. However, the development of VTL technology comprises many technical and practical challenges. Clearly, a general adoption of VTL technology is not feasible until a significant number of vehicles are equipped with V2V communication capabilities. In addition, the application-level protocols for VTL technology must be developed and accepted by standardization organizations and national authorities across the globe. To speed up the adoption of V2V technology, authorities in the United States, Europe and Japan are currently considering to introduce legislation that makes it mandatory to equip road vehicles with V2V communication capabilities.

From a technical perspective, it is still an open question whether existing and emerging vehicle-to-vehicle (V2V) and vehicle-to-infrastructure

(V2I) communication standards provide adequate bandwidth, reliability and security for implementing virtual traffic lights. Standards for Dedicated Short-Range Communications (DSRC) for automotive systems are currently being developed by the Institute of Electrical and Electronic Engineers (IEEE) in the United States, by the European Telecommunications Standards Institute (ETSI), and by the Association of Radio Industries and Businesses (ARIB) in Japan. The IEEE standard, known as IEEE 1609 [33], provides communication range of up to 1km with transmission rates of 3Mbps to 27Mbps for vehicles travelling in velocities up to 260 km/h.

Although such transmission rates would be sufficient for implementing VTLs, the DSRC technology may not provide sufficient connectivity around intersections in urban areas where transmissions can be blocked by environmental conditions such as high-rise buildings, trees, mountains, et cetera; so called no-line-of-sight (NLOS) conditions. Although one study [44] shows that NLOS conditions are not necessarily a show stopper for VTLs, we cannot exclude that NLOS conditions in conjunction with extreme weather, malicious adversaries, or adverse physical disturbances (like solar flares), can cause conditions where the execution of leader election protocol for a VTL is affected by massive communication failures.

This thesis addresses a fundamental challenge in the design of leader election and other distributed agreement algorithms for systems that rely on wireless data communication, namely the design of agreement algorithms that exhibit a low probability of disagreement in the presence of massive communication failures. It has not been my ambition to provide a complete solution for the leader election problem in virtual traffic lights and other emerging wireless automotive application. We merely use the virtual traffic light application as an example to illustrate the practical relevance of our work.

In the following three chapters, we propose and analyse several distributed agreements algorithms that aim to solve three fundamental agree-

ment problems. We elaborate on the system model, failure models and the simplifying assumptions used in our analyses.

4

1-of-n Selection

We propose and investigate a family of synchronous round-based consensus algorithms to solve the problem of selecting one value among n proposed values, called the *1-of-n* selection problem. In this problem, each process in a system of n processes, initially proposes one value. After one or several rounds of communication, where the processes exchange information about the proposed values, each process either decides to *select a value*, which has to be one of the proposed values, or decides to *abort*. We call our proposed consensus algorithms the *1-of-n* selection algorithms.

We analyse the behaviour of the *1-of-n* selection algorithms in the presence of an arbitrary number of message losses. We consider two different scenarios for a lost message: (i) when all the intended receivers of

the message fail to receive the message (*symmetric message loss*), and (ii) when only a subset of the intended receivers fail to receive the message (*asymmetric message loss*). We know from previous research that under the given communication failure assumptions, it is impossible to construct an algorithm which guarantees consensus among the participating processes in the algorithm. Therefore, our focus is to design simple decision making algorithms which result in the probability of disagreement as low as possible.

We show that the probability of disagreement in general depends on three main parameters: i) the number of processes in the system, ii) the number of rounds of message exchange and iii) the probability of message loss. In addition, it also depends on the decision criterion that determines whether a process should abort or select a value. We propose three different decision criteria to be run on top of a *1-of-n* selection algorithm. Each decision criterion consists of logical expressions which decide whether a process should select a value or abort due to a lack of information concerning the status of other processes.

At the system level, there are three possible outcomes for a *1-of-n* selection algorithm:

Agreement on a value : If *all* processes decide to select a value.

Agreement to abort : If *all* processes decide to abort the decision making due to the lack of information.

Disagreement : If *some* processes (at least one process) decide to *select a value* while the remaining processes (at least one process) decide to *abort*.

We restrict ourselves to the algorithms where the participating processes never decide on a value unless they have access to the proposed values from all other processes. This condition requires that each process should know the exact set of participating nodes, i.e. each process must know the *number* and the *identity* of other processes in the system.

Such a simplifying assumption ensures that there are no two processes which decide on selecting different values. As a result, the only kind of disagreement that can occur in a *1-of-n* selection algorithm is when some processes decide to select the *same* value, while the remaining processes decide to abort, i.e. all disagreement cases are *safe*.

In this Chapter we specifically address the following research questions for synchronous algorithms that solve the *1-of-n* selection problem:

- **How do we calculate the probability of disagreement in the presence of an arbitrary number of lost messages?**
- **How does the algorithm's decision criterion influence the probability of disagreement?**
- **How does the communication failure model (symmetric vs. asymmetric failures) influence the the probability of disagreement?**

In Section 4.1, we describe our system model, failure assumptions and the *1-of-n* selection algorithms in detail. We present our three proposed decision algorithms called the *optimistic*, the *pessimistic* and the *moderately pessimistic* decision criterion in Section 4.1.2. To have a better understanding of the influence of each decision criterion on the probability of disagreement, in Section 4.2, we show a comparison of the outcomes of the *1-of-2* selection algorithm using each decision criterion for two rounds of execution. In Section 4.3, we present a probabilistic analysis of the *1-of-n* selection algorithms using the three decision criteria, for two different communication failure models, i.e. the *symmetric* and *asymmetric* communication failure models. We present closed-form expressions to calculate the probability of disagreement in the presence of *symmetric* communication failures in Section 4.3.1. In Section 4.3.2, we present our results of the probabilistic analysis of the algorithm under the asymmetric communication failure model. For the asymmetric failure model, we use the PRISM model checker. The details on the PRISM

models we used are given both in Section 4.3.2 and Appendix A.2 and A.3. In Section 4.5, we discuss our main observations from the probabilistic analysis of the *1-of-n* selection algorithms. Finally, in Section 4.5, we present the chapter conclusions.

4.1 Protocol Description

We assume a synchronous system of n processes executing a *1-of-n* selection algorithm for R rounds. The processes are indexed respectively with their identifiers as: $\Pi = \{p_1, .., p_n\}$. Each process p_i initially knows the exact set of Π, i.e. the number and the identity of others. We assume that the processes are fully connected to one another via point-to-point links with which they exchange messages.

Our system model is based on the classical round-based computational model which was first introduced by Dwork et al. [21] and has been used in the solutions for the agreement problem in synchronous message-passing systems (See for example [13], [57], [18]). Based on the given round-based model, all processes execute the algorithm for R rounds (R is fixed at the design time). In each round, each process broadcasts a message and waits to receive messages from other processes. Then at the end of each round, each process computes and updates its new state according to the received messages. We define three steps for a round: *send, receive* and *compute*. We assume that a message received by a process in a round has been sent at the same round. A message which is not received in a round is considered to be discarded (or lost) [13].

Our failure model is inspired by the model introduced by Santoro and Widmayer in [52] denoted as the *transmission fault model*. We consider failures as message losses that can occur on any communication link at any time during the execution of the algorithm. We assume no restrictions on the number, timing or pattern of the lost messages. We consider two different scenarios for a lost message: (i) when all the intended receivers of the message fail to receive the message (*symmetric* message

loss), and (ii) when only a subset of the intended receivers fail to receive the message (*asymmetric* message loss).

For simplicity, we assume that the processes are fault-free. Note, however, that a send omission failure of a process is equivalent to a symmetric message loss, and that a receive omission failure is equivalent to an asymmetric message loss where only one process fails to receive a message.

4.1.1 The *1-of-n* selection algorithms

Alg. 1 shows the pseudocode of the *1-of-n* selection algorithms executed by each process $p_i \in \Pi = \{p_1, \ldots, p_n\}$. Each of the processes as p_i initially constructs a message (denoted as msg_i). A message of a process contains a proposed value ($proposed_i$) and a bit-vector (v_i) of length n. Each element of v_i represents the view of process p_i from the proposed value of a process in the system. Initially, $v_i[i] = 1$ and $\forall j$, $v_i[j] = 0$, i.e. at this point p_i has not received any message from other processes in the system.

We define v_i as *complete* if all of its elements are set to 1. Similarly, we define v_i as *incomplete* if at least one of its elements is 0.

Algorithm 1 Generic algorithm for *1-of-n* selection (p_i)

```
1: msg_i ← {proposed_i, v_i};
2: for r = 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();
```

After initialization, each process iterates the send, receive and compute phases for R rounds. These phases work as follows:

- **send (msg_j):** Process $p_i \in \Pi$ broadcasts msg_i to all other processes.
- **receive( ):** Process p_i receives messages from the other $n-1$ processes. If p_i does not receive a message from process p_j within a bounded time, it assumes that message to be lost.
- **compute (msg_j):** Each process p_i at the end of each round performs the computation phase algorithm and updates its local state and its message accordingly.

After R rounds of execution, each process executes the decision algorithm. At the end of the execution of the algorithm each process either decides to *select a value* or decides to *abort*.

4.1.2 The Decision Criteria

We propose three different decision criteria for the consensus algorithm given in Alg. 1; namely the *optimistic* decision criterion, *pessimistic* decision criterion and the *moderately pessimistic* decision criterion.

Algorithm 2 compute (msg_i): optimistic criterion (p_i)

```
1: ∀p_j ∈ Π − {p_i}
2: if p_i received msg_j then
3:     if proposed_i < proposed_j then
4:         proposed_i ← proposed_j;
5:     end if
6:     ∀p_k ∈ Π − {p_i, p_j}
7:     if (v_j[k] = 1 and v_i[k] = 0) then
8:         v_i[k] ← 1;
9:     end if
10:    v_i[j] ← 1;
11: end if
12: msg_i ← {proposed_i, v_i};
```

Alg. 2 shows the compute phase for a process p_i executing the optimistic criterion. If the process p_i receives a message from p_j with a

larger proposed value, i.e., $proposed_j > proposed_i$, process p_i must update $proposed_i$ to $proposed_j$. Also, each process p_i updates its v_i vector at the end of each round as follows: For all the elements of v_j that are set to 1, process p_i sets the corresponding elements in its view (v_i) to 1 (If it is not already 1).

Algorithm 3 decision_algorithm(), optimistic criterion (p_i)

```
1: if v_i is complete then
2:     p_i selects proposed_i;
3: else
4:     abort;
5: end if
```

Alg. 3 shows the description of the *optimistic* decision criterion. Executing the optimistic decision criterion, if at the end of the R^{th} round, the view of a process p_i is *complete*, p_i must select its $proposed_i$ as the highest value. Indeed a process with complete view optimistically assumes that all the other processes have also complete views and select a value. A process with an incomplete view at the end of the R^{th} round decides to abort.

Alg. 4 shows the compute phase for the pessimistic decision criterion. At the end of all rounds except for the last round, process p_i updates its proposed value and its view vector in the same way as in the compute phase of the optimistic criterion. In addition at the end of all rounds p_i updates its C_i, its confirmation vector, by definition.

Alg. 5 shows the description of the pessimistic decision criterion. A process p_i with an *incomplete* v_i at the end of the execution of the algorithm decides to abort while a process p_i with a *complete* view pessimistically assumes that other processes do not have complete views unless they confirmed this at some point during R rounds of execution. If process p_i does not receive such confirmations from all processes it decides to abort. We define a vector of size n for each process p_i, called C_i, in order to keep a record of the processes who have sent a confirmation to p_i indicating that their view is complete. Initially, $\forall j\ C_i[j] = 0$. When $(C_i)_j$ is set

Algorithm 4 compute (msg_i): pessimistic criterion (p_i)

```
1: ∀p_j ∈ Π − {p_i}
2: if p_i received msg_j then
3:     if r ≠ R then
4:         if proposed_i < proposed_j then
5:             proposed_i ← proposed_j;
6:         end if
7:         ∀p_k ∈ Π − {p_i, p_j}
8:         if (v_j[k] = 1 and v_i[k] = 0) then
9:             v_i[k] ← 1;
10:        end if
11:        v_i[j] ← 1;
12:    end if
13:    if v_j is complete then
14:        C_i[j] ← 1;
15:    end if
16: end if
17: ∀j ∈ {1..n}
18: if v_i[j] = 1 then
19:    C_i[i] ← 1;
20: end if
21: msg_i ← {proposed_i, v_i};
```

Algorithm 5 decision_algorithm(), pessimistic criterion (p_i)

```
1: if v_i is complete then
2:     if C_i is complete then
3:         p_i selects proposed_i;
4:     else
5:         abort;
6:     end if
7: else
8:     abort;
9: end if
```

to 1, it means that p_i has received a message from p_j showing that v_j is complete. C_i is complete if all of its elements are set to 1.

Algorithm 6 compute (msg_i): moderately pessimistic criterion (p_i)

```
1: ∀p_j ∈ Π − {p_i}
2: if p_i received msg_j then
3:     if r ≠ R then
4:         if proposed_i < proposed_j then
5:             proposed_i ← proposed_j;
6:         end if
7:         ∀p_k ∈ Π \ {p_i, p_j}
8:         if (v_j[k] = 1 and v_i[k] = 0) then
9:             v_i[k] ← 1;
10:        end if
11:    end if
12: end if
13: ∀j ∈ {1..n}
14: if v_i[j] = 1 then
15:    C_i[i] ← 1;
16: end if
17: msg_i ← {proposed_i, v_i};
```

Alg. 6 shows the description of the compute phase defined for the moderately pessimistic decision criterion. 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 also updates its v_i vector as follows: for all the elements of v_j that are set to 1, p_i sets the corresponding elements in v_i to 1 (if it is not already 1). 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 decision criterion. A process p_i executing the moderately pessimistic decision criterion decides to abort if its view is 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 incomplete, process p_i must abort, otherwise it selects its $proposed_i$ as the highest value. Pro-

cess p_i disregards the lost messages in the last round and optimistically assumes a complete view for the senders of the lost messages.

Algorithm 7 decision_algorithm(), moderately pessimistic criterion (p_i)

```
1: if v_i is complete then
2:     if receives some incomplete 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
```

4.2 Analysis of a Simple System

In this section, we compare the outcomes of each of the three given decision criteria for a *1-of-2* selection algorithm using two rounds of message exchange ($n = 2$ and $R = 2$). Our aim is to provide an intuitive understanding of how the choice of a decision criterion influences the probability of the three possible outcomes of the *1-of-n* selection algorithms, i.e., *agreement on a value*, *agreement on abort* and *disagreement*.

We assume that two processes which are called p_1 and p_2, respectively propose *11* and *12* as their initial values. The objective of the algorithm is to reach agreement on the highest value proposed by any of the two processes, i.e., the value *12* in this example.

Each process sends its message using a point-to-point link to the other process. Therefore, in a *1-of-2* selection algorithm in two rounds of execution each of the two processes sends two messages, one in each round, and as a result there are 4 exchanged messages among processes in total. Since we assume unbounded number of communication failures, for a *1-of-2* selection algorithm with $R = 2$ rounds, we have $2^4 = 16$ permutations of lost and successful messages. As a result, we have sixteen

cases of execution of the algorithm.

Table 4.1 illustrates the sixteen cases of execution of a *1-of-2* selection algorithm in two rounds. The columns denoted $T1_2$ refer to the transmissions from p_1 to p_2, while columns denoted $T2_1$ refers to the transmissions from p_2 to p_1. A successful message transmission is marked as 'OK' while a transmission failure is marked as 'Fail'. The columns denoted msg_1 and msg_2 state the content of the messages received by each process in the corresponding round. msg_1 (resp. msg_2) is the message received by p_2 (resp. p_1) from p_1 (resp. p_2). As explained in Section 4.1.1, msg_i consists of v_i, the view vector of process p_i and the value proposed by p_i ($proposed_i$). The content of a message is given within square brackets. The view vector is given with curly brackets. As an example, $[\{0\}, 11]$ indicates that the view vector[1] is $\{0\}$ (this shows that the process has not yet received a message from the other process) while the proposed value is 11. '[–]' denotes the loss of a message due to a transmission failure.

Table 4.2 shows the outcomes of the three decision criteria for the 16 cases shown in Table 4.1. As we can see in Table 4.2 there are 9 cases of agreement on a value, 6 cases of disagreement and one case of agreement to abort for the optimistic decision criterion. For the moderately pessimistic decision criterion there are 4 cases of agreement on a value, 4 cases of disagreement and 8 cases of agreement to abort. In the case of the pessimistic decision criterion there is one case of agreement on a value, 2 cases of disagreement, and 13 cases of agreement to abort.

Now we explain some of the cases shown in Table 4.2 in detail. For the pessimistic decision criterion, Case 2 and 3 result in disagreement with losing only one message. In Case 2, $T2_1$ fails. This means that p_1 does not receive the confirmation that p_2 has a complete view which implies that C_1 in Alg.4is incomplete. Therefore, p_1 decides to abort, while p_2 selects the value 12. On the other hand, the other two decision criteria result in an agreement on 12. For the optimistic criterion, the condition

[1]Note that the view vector actually consists of two bits. However, one of the bits represents the process's view of its own value and this bit is always set to 1. For simplicity, we have omitted this bit in the example.

Table 4.1: *Possible executions for a $n = 2$, $R = 2$ system*
$msg_i = [v_i, proposed_i]$

Case	*round* 1				*round* 2			
	$T1_2$	$T2_1$	msg_1	msg_2	$T1_2$	$T2_1$	msg_1	msg_2
1	OK	OK	[{0},11]	[{0},12]	OK	OK	[{1},12]	[{1},12]
2	OK	OK	[{0},11]	[{0},12]	OK	Fail	[{1},12]	[–]
3	OK	OK	[{0},11]	[{0},12]	Fail	OK	[–]	[{1},12]
4	OK	OK	[{0},11]	[{0},12]	Fail	Fail	[–]	[–]
5	OK	Fail	[{0},11]	[–]	OK	OK	[{0},11]	[{1},12]
6	OK	Fail	[{0},11]	[–]	OK	Fail	[{0},11]	[–]
7	OK	Fail	[{0},11]	[–]	Fail	OK	[–]	[{1},12]
8	OK	Fail	[{0},11]	[–]	Fail	Fail	[–]	[–]
9	Fail	OK	[–]	[{0},12]	OK	OK	[{1},12]	[{0},12]
10	Fail	OK	[–]	[{0},12]	OK	Fail	[{1},12]	[–]
11	Fail	OK	[–]	[{0},12]	Fail	OK	[–]	[{0},12]
12	Fail	OK	[–]	[{0},12]	Fail	Fail	[–]	[–]
13	Fail	Fail	[–]	[–]	OK	OK	[{0},11]	[{0},12]
14	Fail	Fail	[–]	[–]	OK	Fail	[{0},11]	[–]
15	Fail	Fail	[–]	[–]	Fail	OK	[–]	[{0},12]
16	Fail	Fail	[–]	[–]	Fail	Fail	[–]	[–]

Table 4.2: *Outcome of the 1-of-2 selection algorithm with $R = 2$*
AG :agreement DG : disagreement

Case	Optimistic	Moderately Pessimistic	Pessimistic
1	AG (12)	AG (12)	AG (12)
2	AG (12)	AG (12)	DG
3	AG (12)	AG (12)	DG
4	AG (12)	AG (12)	AG (abort)
5	AG (12)	AG (abort)	AG (abort)
6	DG	AG (abort)	AG (abort)
7	AG (12)	DG	AG (abort)
8	DG	DG	AG (abort)
9	AG (12)	AG (abort)	AG (abort)
10	AG (12)	DG	AG (abort)
11	DG	AG (abort)	AG (abort)
12	DG	DG	AG (abort)
13	AG (12)	AG (abort)	AG (abort)
14	DG	AG (abort)	AG (abort)
15	DG	AG (abort)	AG (abort)
16	AG (abort)	AG (abort)	AG (abort)

of having a complete view (**if** condition at line 1 in Alg. 2) is satisfied for both processes and therefore they both select 12. In case of the moderately pessimistic decision criterion, p_1 does not receive a message that indicates the incomplete view of the other process (**if** condition at line 2 in Alg. 5), so it selects 12. Case 3 is similar to Case 2, but with p_1 and p_2 swapped.

Similar to Case 2 and 3, in Case 4 both the optimistic and moderately pessimistic decision criteria result in an agreement on 12. However, in case of the pessimistic criterion as both messages are lost in the second round, process p_1 and p_2 both abort because they do not receive the confirmation that the other process has a complete view (**if** condition at line 2 in Alg. 3).

In Case 7, $T2_1$ in the first round and $T1_2$ in the second round fail. For the optimistic decision criterion, both v_1 and v_2 are complete and therefore both p_1 and p_2 select 12. For the pessimistic decision criterion p_1 aborts because it has not received msg_2 in the first round. According to the compute phase of the pessimistic criterion, p_1 does not update v_1 at the end of round R and as a result v_1 remains incomplete (**if** condition at line 1 in Alg. 3 is not satisfied). On the other hand process p_2 receives msg_1 in round one and therefore completes v_2. However in the second round p_2 does not receive msg_1 from process p_1, which means p_2 does not have a confirmation that p_1 has a complete view. Based on the second **if** condition in Alg. 3 p_2 also decides to abort (i.e., outcome is agreement to abort). The moderately pessimistic decision criterion in Case 7 results in disagreement. p_1 aborts (for the same reason as for the pessimistic criterion), while p_2 selects 12 (the first **if** condition in Alg. 5 holds for p_2). On the other hand, in round 2, $T1_2$ is failed which means that p_2 receives no information from p_1 indicating whether v_1 is incomplete or not (the second **if** condition at line 2 in Alg. 5 does not hold). Case 10 is similar to Case 7 with p_1 and p_2 swapped.

In Case 8, all messages sent to p_1 are lost. So process p_1, with an incomplete view, decides to abort for all decision criteria. Process p_2

receives msg_1 in round one and completes v_2. Therefore in case of the optimistic decision criterion p_2 selects a value which implies disagreement. On the other hand, for the pessimistic decision criterion, p_2 aborts although it has a complete view. This is because p_2 never receives a message from p_1 indicating that v_1 is complete. This leads to an agreement on abort. For the moderately pessimistic decision criterion, according to the **if** condition in Alg. 5 at line 2, p_2 decides to select a value that results in disagreement. Case 12 is the same as Case 8, with p_1 and p_2 swapped.

Case 5 and 9 are similar in the sense that for both cases one message is lost in the first round ($T2_1$ in Case 5 and $T1_2$ fail), while there are no message losses in the first round. For the case of the optimistic decision criterion since both v_1 and v_2 are complete both p_1 and p_2 select the value 12. However, for the other two decision criteria, as processes do not update their views at the end of the second round (round R in Alg. 1), both p_1 and p_2 have incomplete views and therefore decide to abort.

Looking at Table 4.2 it is easy to see that the optimistic decision criterion has a large number of cases that results in agreement on a value while there are only two cases of disagreement for the pessimistic criterion, as in most cases it will result in an abort. Moreover, the pessimistic criterion results in disagreement if only one message is lost, in contrast to the optimistic decision criterion for which at least two messages must be lost in order to have disagreement among the processes. We can also see that the number of cases of agreement, abort and disagreement for the moderately pessimistic criterion lies in between the corresponding numbers for the optimistic and pessimistic decision criteria. This example gives an intuitive explanation of the impact of the different decision criteria on the outcome of the *1-of-n* selection algorithms.

4.3 Probabilistic Analysis

In this chapter, we present an analysis of the behaviour of the *1-of-n* selection algorithms in the presence of *symmetric* and *asymmetric* com-

munication failures. We analyse the algorithm executing each of the three given decision criteria and under different settings of the system parameters, i.e. n, R and the probability of message loss.

In Section 4.3.1, we begin with presenting several graphs showing how the probability of each outcome of the algorithm varies for different configurations of a system under symmetric communication failures. Then we present the details on the derivation of the closed-form expressions that we propose to calculate the probability of each outcome of the *1-of-n* selection algorithms. In Section 4.3.2, we present an analysis of the behaviour of the *1-of-n* selection algorithms under asymmetric communication failures using a probabilistic model checking tool called PRISM.

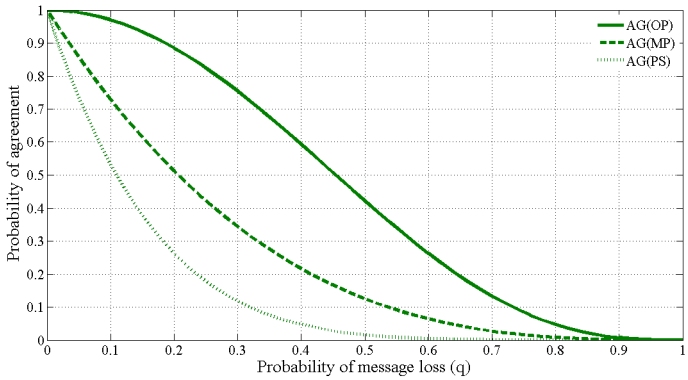
4.3.1 Analysis for Symmetric Failures

In this section, we present several graphs showing how the probabilities ($P_{\texttt{AG}}$, $P_{\texttt{AB}}$ and $P_{\texttt{DG}}$) of the outcomes of the algorithm varies for different decision criteria. In order to analyse the behaviour of the algorithm, we vary the main system parameters: the number of processes (n), the number of rounds (R) and the probability of message loss (q).

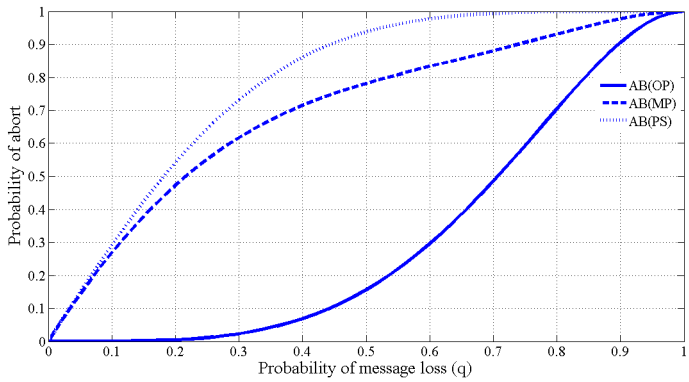
We denote the outcomes as `AG` , `AB` and `DG` respectively for *agreement on a value*, *agreement to abort* and *disagreement*. Also, we denote each decision criterion with OP, MP and PS respectively for the optimistic, the moderately pessimistic and the pessimistic decision criterion.

Observations

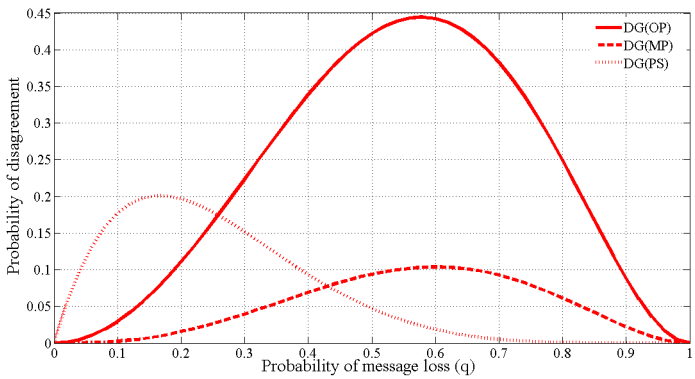
Fig. 4.1 shows the probabilities of the three outcomes of the three decision criteria for a *1-of-3* selection algorithm with 2 rounds of execution as a function of q. As we see in Fig. 4.1(a) for any value of q, the optimistic decision criterion has the highest probability of agreement while the pessimistic one has the lowest. On the other hand, the pessimistic decision criterion has the highest probability of abort compared to the other two decision criteria (See Fig. 4.1(b)). For all decision criteria the



(a) The probability of agreement on a value



(b) The probability of agreement to abort



(c) The probability of disagreement

Figure 4.1: *Probability of an outcome for 1-of-3 selection algorithm for $R = 2$ with the symmetric failure model.*

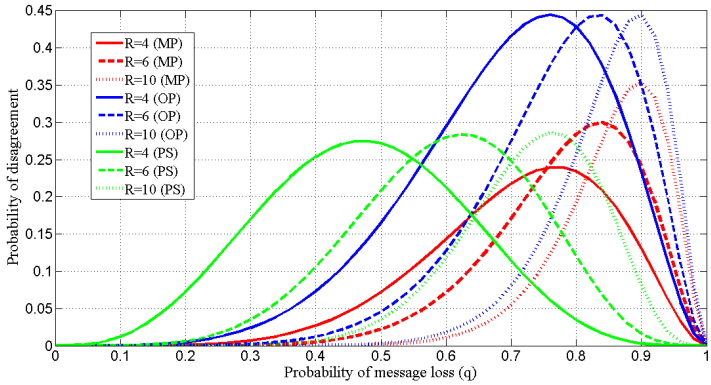


Figure 4.2: *Probability of disagreement for 1-of-3 selection algorithm ($R = 4, 6, 10$) for different decision criteria with the symmetric failures.*

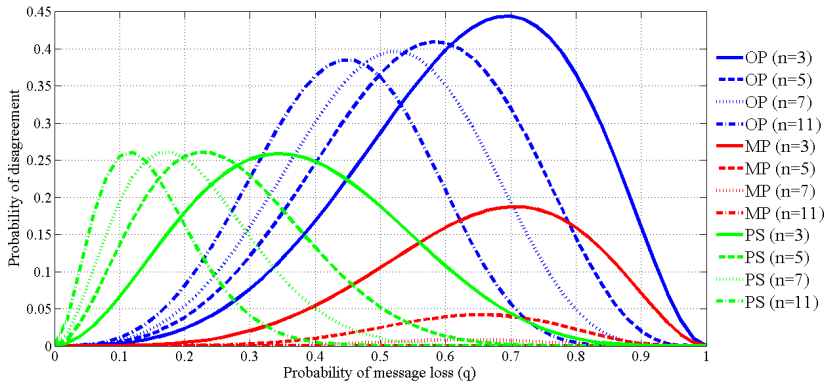


Figure 4.3: *Probability of disagreement for 1-of-n selection algorithms ($n = 3, 5, 4, 11$ and $R = 3$) for different decision criteria with the symmetric failures.*

probability of disagreement shows a distinct peak. The maximum probability of disagreement varies significantly for different decision criteria. The optimistic decision criterion has the highest maximum probability of

disagreement, which is around 44% and occurs at $q \approx 0.6$. The peak of the curve for the pessimistic decision criterion is 20% and occurs at the value of $q \approx 0.16$. The smallest maximum probability of disagreement occurs for the moderately pessimistic decision criterion (See Fig. 4.1(c)).

Fig. 4.2 shows how the probability of disagreement varies for a *1-of-3* selection algorithm when the number of rounds is 4, 6 and 10. As we see, with increasing R, the maximum probability of disagreement remains around the same value for the optimistic and the pessimistic decision criteria, while it increases for the moderately pessimistic decision criterion as R increases. It is interesting to note that the MP criterion has a higher maximum probability of disagreement compared to the PS criterion when the number rounds are 6 and 10. This shows that it is an advantage to keep the number of rounds low when using the MP criterion. For all decision criteria, we see that the peak of the curves moves to the right along the x-axis as we increase R. This implies that the P_{AG} value increases as we increase the number rounds, for all decision criteria. This also means that the value of P_{AB} decreases as we increase R.

Fig. 4.3 shows how the probability of disagreement varies for a system executing the *1-of-n* selection algorithms with $n = 3, 5, 7$ and 11 and $R = 3$. Here we see that the peaks of the curves become lower for the OP and MP criteria as we increase the number of processes in the system. In contrast, the peaks remain more or less constant for the PS criterion when the number of processes increases. For all decision criteria, the peaks move to the left along the x-axis when the number of processes increases. This implies that the P_{AG} value increases, and the value of P_{AB} decreases, for all decision criteria when the number of processes increases. Fig. 4.3 also shows that the probability of disagreement becomes quite low for the MP criterion for systems with 7 or more processes.

Closed-form Expressions

In this section, we present our approach in the derivation of closed-form expressions to count the number of different possibilities that all processes

agree or disagree. Assuming the occurrence of symmetric communication failures only, the number of different possibilities for each outcome, should be counted out of the total $2^{n \cdot R}$ possibilities[2].

We compute the number of executions of the *1-of-n* selection algorithms with respect to the number and pattern of the lost messages, which result in all processes select the same value (*agreement on a value*), all processes decide to abort (*agreement on abort*) or the cases in which some processes select the same value and some decide to abort (*disagreement*). In particular, we compute `AG`, `AB`, and `DG` for a given n and R, where

- `AG` is the total number of executions that all processes select the same value.
- `AB` is the total number of executions that all processes decide to abort.
- `DG` is the total number of executions that some processes select a value while others decide to abort.

In addition, by considering the probability of a message loss, q, we also compute

- $P_{\texttt{AG}}$: the probability of agreement on a value;
- $P_{\texttt{AB}}$: the probability of agreement to abort; and
- $P_{\texttt{DG}}$: the probability of disagreement.

for a system of n processes where $P_{\texttt{AG}} + P_{\texttt{AB}} + P_{\texttt{DG}} = 1$.

According to Algorithm 1, each process executes the send operation in each round. Each send operation can either be successful (all processes receive the broadcast message) or unsuccessful (no process receives the message). As a result, executing a *1-of-n* selection algorithm, at the end of the R^{th} round, there are $2^{n \cdot R}$ possible combinations of the views of the n processes. For large values of n and R the number of different possible executions of the algorithm can be exponential. For such big systems, it can be computationally prohibitive to calculate `AG`, `AB` and `DG` , when $n \cdot R$ is very large (e.g., $n = 20$ cars execute the consensus algorithm for $R = 5$ rounds in a road intersection). Therefore, finding an

[2]The messages sent from n number of processes can be lost or delivered in any round among R rounds of execution.

efficient way to compute AG, AB, and DG , is a non-trivial and challenging problem. Our endeavour in the following sections (Section 4.3.1, 4.3.1 and 4.3.1) is to address this problem and efficiently compute AG, AB, and DG for the optimistic, the pessimistic and the moderately pessimistic decision criteria. We show that it is possible to compute AG, AB and DG in linear time by conducting elegant analysis of each decision criterion for the *1-of-n* selection algorithms. The following propositions are useful to show how we derive the closed-form expressions in the following sections. For the ease of reading we leave the proofs to the Appendix. A.1.

Proposition 4.1. *Two or more processes fail to send their messages in all the* $1 \dots K$ *rounds, if and only if, all* n *processes have incomplete views at the end of the* K^{th} *round.*

Proposition 4.2. *All processes have complete views at the end of the* K^{th} *round, if and only if, each process successfully broadcasts its message in at least one of the* K *rounds.*

Proposition 4.3. *One process has the complete view and the remaining* $(n-1)$ *processes have incomplete views at the end of* K^{th} *round, if and only if, exactly one process fails to broadcast its message in all of the* K *rounds.*

In the following sections, we use Proposition 4.1—4.3 in order to derive expressions to calculate AG, AB and DG for each decision criterion. In addition to the propositions, for the ease of presentation, we define predicates C_0, C_1, C_2 and C_3 each representing an **if** condition given in Algorithms 2, 4 and 7. Table 4.3 shows the conditions under which the given predicates can be *true* or *false* for a given process p_i executing a *1-of-n* selection algorithm.

Optimistic Decision Criterion

In this section, first we derive closed-form expressions to calculate AG, AB and DG . Then, we derive the expressions to calculate $P_{\texttt{AG}}$, $P_{\texttt{AB}}$ and $P_{\texttt{DG}}$

Table 4.3: *Useful predicates for describing the decision making algorithms*

Predicate		Condition
C_0	*true*	p_i has a complete view at the end of round R.
C_0	*false*	p_i has an incomplete view at the end of round R.
C_1	*true*	p_i has a complete view at the end of round $R-1$.
C_1	*false*	p_i has an incomplete view at the end of round $R-1$.
C_2	*true*	At round R, process p_i received all messages from all processes, which all indicate the complete views of their senders.
C_2	*false*	At round R, process p_i has lost some messages or has received some messages from processes with incomplete view.
C_3	*false*	Either p_i receives no message during the last round R or any message received by the process during the last round is sent from a process with complete view.

for the *1-of-n* selection algorithm with the optimistic decision criterion. We know from the the description of the optimistic decision criterion given in Algorithm 2 that, if the view of a process is complete (i.e., C_0 is *true*) at the end of the R^{th} round, the process selects a value; otherwise, it must decide to abort.

Finding closed-form expressions for $P_{\texttt{AG}}$ To derive the closed-form expression to calculate $P_{\texttt{AG}}$, first, we determine $\texttt{AG}$, the number of cases that the views of all processes are complete at the end of the R^{th} round (i.e., C_0 is *true* for all processes).

According to Proposition 4.2, the views of all of the n processes are complete, if and only if each process successfully sends its message at least once during R rounds. Considering that each send operation in a round, can either be successful or unsuccessful, there are $\sum_{i=1}^{R}\binom{R}{i}$ possible executions of the algorithm in which at least once in R rounds, the message broadcast by a process is successfully delivered to the other processes. Since there are n processes in a system, we can calculate the total number of executions that result in an agreement on a value among n processes in R rounds using the given formula in 4.1.

$$\texttt{AG} \;=\; \left(\sum_{i=1}^{R}\binom{R}{i}\right)^{n} = (2^R-1)^n \tag{4.1}$$

The formula given in 4.2 calculates the probability that all processes reach to an agreement on a value when the probability of losing a message broadcast is q. In the given expression, i send operations of each process are successful with the probability of $1-q$ while $R-i$ send operations are unsuccessful for each i in Eq. (4.1) with the probability of q. Note that i starts from one which means that we assume each process broadcasts its message at least once with the probability of 1. So the probability that all of the n processes decide to select a value (i.e., reach to an agreement on a value) is given as follows:

$$P_{\texttt{AG}} = \left(\sum_{i=1}^{R} \binom{R}{i} \cdot (1-q)^i \cdot q^{R-i} \right)^n = (1-q^R)^n \tag{4.2}$$

Finding closed-form expressions for $P_{\texttt{AB}}$ To calculate AB , we need to find the number of ways in which the view of each of the n processes is incomplete at the end of the R^{th} round (i.e., C_0 is *false* for all processes). According to Proposition 4.1, C_0 is *false* for all processes if and only if at least two processes fail to broadcast their messages during all R rounds. If there are i processes that fail to broadcast their message in all R rounds, where $2 \leq i \leq n$, then each of the remaining $n-i$ processes successfully broadcast their message in at least one of the R rounds. For a given i provided that $2 \leq i \leq n$, there are $\left(\sum_{j=1}^{R} \binom{R}{j} \right)^{n-i} = (2^R-1)^{n-i}$ possible cases in which all $n-i$ processes successfully broadcast their message in at least one of the R rounds. Moreover, i number of processes can be selected out of n processes in $\binom{n}{i}$ number of ways. Therefore, the closed-form expression to calculate the number of cases that all processes decide to abort is given in 4.3:

$$\texttt{AB} = \sum_{i=2}^{n} \binom{n}{i} \cdot (2^R-1)^{n-i} \tag{4.3}$$

Given that the probability of a message loss is q, the probability that exactly i processes fail to send in all R rounds is $(q^R)^i$. On the other hand, the probability that all $n-i$ processes successfully broadcast their message in at least one of the R rounds is $(\sum_{j=1}^{R} \binom{R}{j} \cdot (1-q)^j \cdot q^{R-j})^{n-i} = (1-q^R)^{n-i}$, where $2 \leq i \leq n$. Consequently, the closed-form expression to calculate the probability that all processes agree to abort is given in 4.4:

$$P_{\texttt{AB}} = \sum_{i=2}^{n} \binom{n}{i} \cdot (q^R)^i \cdot (1-q^R)^{n-i} \tag{4.4}$$

Finding closed-form expressions for $P_{\texttt{DG}}$ We know that the total number of the cases of executions of a *1-of-n* selection algorithm in R rounds assuming symmetric communication failures is $\texttt{AG} + \texttt{AB} + \texttt{DG} = 2^{n \cdot R}$. So we can calculate the value of $\texttt{DG}$ using the formula given in 4.5.

$$\texttt{DG} = 2^{n \cdot R} - \texttt{AG} - \texttt{AB} \tag{4.5}$$

where $\texttt{AG}$ and $\texttt{AB}$ are computed in Eq. (4.1) and Eq. (4.3), respectively. Similarly, the probability of disagreement can be calculated from the formula 4.6:

$$P_{\texttt{DG}} = 1 - P_{\texttt{AG}} - P_{\texttt{AB}} \tag{4.6}$$

where $P_{\texttt{AG}}$ and $P_{\texttt{AB}}$ are computed in Eq. (4.2) and Eq. (4.4), respectively. This completes the analysis of the optimistic decision criteria. In the next section, we present the analysis for the pessimistic decision criterion.

Pessimistic Decision Criterion

In this section, we present the closed-form expressions to compute $P_{\texttt{DG}}$ and $P_{\texttt{AG}}$ for a system of n processes executing the *1-of-n* selection algo-

rithm with the pessimistic decision criterion when the probability of a symmetric message loss is q. According to the description of the pessimistic decision criterion in Algorithm 4, and the specified predicates in Table 4.3, if C_1 and C_2 are *true* for a process it can decide to select a value.

Finding closed-form expressions for $P_{\mathtt{DG}}$ To derive closed-form expressions to calculate the probability of disagreement for the case of the pessimistic decision criterion, we define Lemma 4.4 proved in Appendix. A.1.4 as below:

Lemma 4.4. *In order to have disagreement among processes, it is necessary that **all** processes have complete views at the end of round $R-1$.*

There are two possible ways that the views of all the processes can be complete at the end of round r, where $1 \leq r \leq R-1$:

Case I When *all* processes have incomplete views at the end of round $r-1$ but they *all* have complete views after round r.

Case II When exactly $n-1$ processes have incomplete views by the end of round $r-1$ and *all* processes have complete views after round r.

We can show that other than these two cases, there is no other case that disagreement may occur after r rounds ($1 \leq r \leq R-1$). Given that the views of all the processes are complete at round r, to compute the probability of disagreement, we have to consider that *exactly* one process, say process p_x, receives complete views from *all* other processes while others do not[3]. We derive the closed-form expression to compute the probability of disagreement for the pessimistic decision criterion with analysing Case I and Case II. More details on the analysis of these cases are given in the Appendix. A.1.4.

[3]Note that more than one process receive confirmation from all other processes if and only if each process receives confirmation from all other processes (i.e., all processes decide to select a value).

Combining the probabilities for Case I and Case II, we can calculate the probability of disagreement for the pessimistic decision criterion as follows:

$$
\begin{aligned}
P_{\mathtt{DG}} &= P_{\mathtt{DG}\ CaseI} + P_{\mathtt{DG}\ CaseII} \qquad (4.7)\\
&= (1-q)^n \cdot q^{(R-1)} \cdot (1-q^{R-1})^{n-1} \cdot n \\
&+ \sum_{r=2}^{R-1} \sum_{i=2}^{n} \binom{n}{i} (1-q^{r-1})^{n-i} \cdot q^{(r-1)\cdot i} \cdot (1-q)^i \cdot n \cdot q^{(R-r)} \cdot (1-q^{R-r})^{n-1} \\
&+ \sum_{r=2}^{R-1} n \cdot q^{r-1} \cdot (1-q^{r-1})^{n-1} \cdot (1-q) \cdot (n-1) \cdot q^{(R-r)} \cdot (1-q^{R-r})^{n-2}
\end{aligned}
$$

Finding closed-form expressions for $P_{\mathtt{AG}}$ Agreement on a value occurs if all processes decide to select a value. According to the description of the pessimistic decision criterion given in Alg. 4, a process p_i decides to select a value if its view is complete by the end of round $R-1$ and if it receives confirmation messages from all other processes at some point during R rounds of execution. The crucial observation is that having complete view by each of the processes at the end of round $R-1$ is a necessary condition for agreement. There are two possible ways the views of all the processes can be complete at the end of round r, where $1 \leq r \leq R-1$:

Case I' All processes have incomplete views by the end of round $r-1$ and all processes have complete views by the end of round r.

Case II' Exactly $n-1$ processes have incomplete views by the end of round $r-1$, then at the end of round r all processes have complete views.

Notice that other than these two cases, there is no other execution that agreement on a value may occur. Given that the views of all processes are complete at round r, to compute the probability of agreement on a value,

we have to consider that each process receives complete messages from all other processes. We derive the closed-form expression to calculate the probability of agreement on a value, $P_{\mathtt{AG}}$, for the pessimistic decision criterion with analysing the two cases, Case I' and Case II'.
Eq. 4.8 and 4.9 show the expressions for calculating the probability of agreement on a value among processes when they meet the conditions for Case I' and Case II', respectively. Details on the analyses of these cases are given in Appendix. A.1.5.

$$P_{\mathtt{AG}CaseI'} = (1-q)^n \cdot (1-q^{R-1})^n \quad + \quad \sum_{r=2}^{R-1}\sum_{i=2}^{n}\binom{n}{i}(1-q^{r-1})^{n-i} \cdot q^{(r-1)\cdot i} \cdot (1-q)^i \cdot (1-q^{(R-r)})^n \tag{4.8}$$

$$P_{\mathtt{AG}CaseII'} = \sum_{r=2}^{R-1} n \cdot q^{r-1} \cdot (1-q^{r-1})^{n-1} \cdot (1-q) \cdot (1-q^{R-r})^{n-1} \tag{4.9}$$

Combining the expressions for Case I' and Case II', the probability of agreement on a value among processes executing the pessimistic decision criterion is computed as follows:

$$\begin{aligned} P_{\mathtt{AG}} &= P_{\mathtt{AG}CaseI'} + P_{\mathtt{AG}CaseII'} = \\ &(1-q)^n \cdot (1-q^{R-1})^n \quad + \\ &\sum_{r=2}^{R-1}\sum_{i=2}^{n}\binom{n}{i}(1-q^{r-1})^{n-i} \cdot q^{(r-1)\cdot i} \cdot (1-q)^i \cdot (1-q^{(R-r)})^n \quad + \\ &\sum_{r=2}^{R-1} n \cdot q^{r-1} \cdot (1-q^{r-1})^{n-1} \cdot (1-q) \cdot (1-q^{R-r})^{n-1} \end{aligned} \tag{4.10}$$

It is not difficult to see that the above equation can be computed in polynomial time. The probability of abort is $P_{\mathtt{AB}} = 1 - P_{\mathtt{DG}} - P_{\mathtt{AG}}$ where $P_{\mathtt{DG}}$ and $P_{\mathtt{AG}}$ are computed in Eq. (4.7) and Eq. (4.10), respectively.

Moderately Pessimistic Decision Criterion

Following, we present the closed-form expressions to compute $P_{\texttt{DG}}$ and $P_{\texttt{AB}}$ for the *1-of-n* selection algorithm executing the moderately pessimistic decision criterion given in Alg. 7, assuming symmetric message losses with the probability of q.

Finding closed-form expressions for $P_{\texttt{DG}}$ We have disagreement among processes, if some processes decide to select a value while the others decide to abort. We assume a set of processes as Π_x that all decide to select a value, while all other processes as p_k in $\Pi - \Pi_x$ decide to abort. Based on Alg. 7, there are two conditions for a process p_x to decide to select a value. The first condition for p_x is to have a complete view by the end of round $R-1$. The second condition is that p_x should not receive any message from any process indicating that the sender's view is incomplete at round R.

Lemma 4.5. *The set Π_x consists of exactly one process p_x which has a complete view at the end of round $R-1$ while all other processes have incomplete views at this point*

We derive the closed-form expression for calculating $P_{\texttt{DG}}$ based on Lemma. 4.5 which is proved using contradiction. For details on the proof for Lemma. 4.5 see Appendix. A.1.6.

According to the definition of the moderately pessimistic decision criterion, on condition for p_x to decide to select a value is that it must not receive a message from a process with incomplete view of the system, in round R. In this case, all other $n-1$ processes have incomplete views. Eq. (4.11) shows the closed-form expression to calculate the probability of disagreement for the moderately pessimistic decision criterion.

$$P_{\texttt{DG}} \quad = \quad n \cdot q^{R-1} \cdot (1 - q^{R-1})^{n-1} \cdot q^{n-1} \tag{4.11}$$

We explain Eq. (4.11) as follows. We proved that there is exactly one process p_x in the set Π_x which has a complete view at round $R-1$. So, none of the other $n-1$ processes received the message broadcast from p_x in any of the $R-1$ rounds. This means that the message sent from p_x has been lost in all $R-1$ rounds with the probability of q^{R-1}, and all messages sent from all other $n-1$ processes have been delivered at least once in $R-1$ rounds with the probability of $(1-q^{R-1})^{n-1}$.

Based on Alg. 7, a process p_x with a complete view, decides to select a value if it did not receive any message from a process with an incomplete view. On the other hand, since the assumption is that the view of all $n-1$ processes are incomplete at the end of round $R-1$, process p_x decides to select a value if it receives no message from any of the $n-1$ processes at round R. So, q^{n-1} refers to the probability that all messages sent from the $n-1$ processes in $\Pi - \Pi_x$ are lost in round R. Finally, as the process p_x can be selected in n possible ways from n processes we multiply the expression by n (See Eq. (4.11)).

Finding closed-form expressions for $P_{\texttt{AB}}$ Based on Alg. 7, we divide the execution cases which result in agreement to abort among processes in two cases as below:

Case (a) All processes have incomplete views at the end of round $R-1$.

Case (b) There is a set of processes called as Π_x that all processes in this set have the complete view of the system by the end of round $R-1$, but in round R they receive incomplete views from all or some of the processes in $\Pi - \Pi_x$.

We calculate the probability of agreement to abort, with calculating the probabilities of each of the given cases, Case (a) and Case (b). More details on our approach can be found in Appendix. A.1.7. We show that the probability of having agreement to abort when all processes have incomplete views at the end of round $R-1$ is calculated using the given expression in 4.12.

$$P_{\text{AB } Case(a)} = \sum_{i=2}^{n} \binom{n}{i} \cdot (q^{R-1})^i \cdot (1 - q^{R-1})^{n-i} \tag{4.12}$$

Eq. A.21 shows the probability that all processes meet the given condition in Case (b) and decide to abort.

$$P_{\text{AB } Case(b)} = n \cdot (1 - q^{R-1})^{n-1} \cdot q^{R-1} \cdot (1 - q^{n-1}) \tag{4.13}$$

Finally, in Eq. 4.14 the probability of reaching to an agreement to abort among processes executing the *1-of-n* selection algorithm with the moderately pessimistic decision criterion is given as the sum of the probabilities of two cases, Case (a) and Case (b).

$$P_{\text{AB}} = P_{\text{AB } Case(a)} + P_{\text{AB } Case(b)} = \tag{4.14}$$
$$n \cdot (1 - q^{R-1})^{n-1} \cdot q^{R-1} \cdot (1 - q^{n-1}) + \sum_{i=2}^{n} \binom{n}{i} (1 - q^{R-1})^{n-i} \cdot (q^{R-1})^i$$

Eq. 4.15 shows how we can calculate the probability of agreement on a value, considering that we have derived the closed-form expressions to calculate the probability of disagreement and abort.

$$P_{\text{AG}} = 1 - P_{\text{DG}} - P_{\text{AB}} \tag{4.15}$$

4.3.2 Analysis for Asymmetric Failures

In this chapter, we analyse the probabilistic behaviour of the *1-of-n* selection algorithms for asymmetric failures using a probabilistic model checking tool called PRISM [36]. PRISM is a probabilistic model checker which is widely used in formal modelling and verification of the protocols

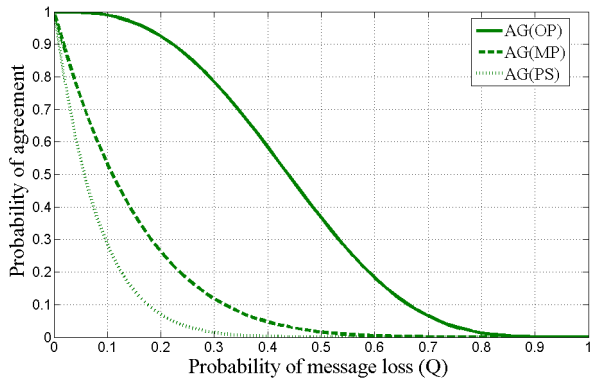
with probabilistic, non-deterministic and real-time characteristics, such as the protocols for wireless systems.

We present a number of graphs showing the behaviour of the *1-of-n* selection algorithms for different system parameters. Then, we explain in detail, the PRISM model for a system of three processes executing the algorithm in R rounds. Our model comprises the three decision criteria, described in Section 4.1.2.

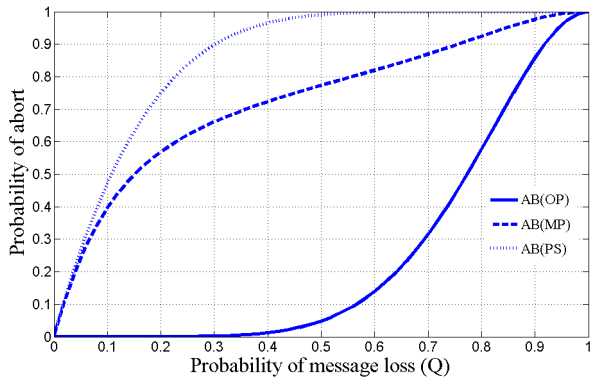
Observations

In this section, we present our results of the behaviour of the *1-of-n* selection algorithms in the presence of asymmetric communication failures. We show how the probability of each outcome of the algorithm (P_{AG}, P_{AB} and P_{DG}) varies for different configurations of the system. We vary the main system parameters: the number of processes (n), the number of rounds (R) and the probability of asymmetric message loss (Q). As we defined in Section 4.1, Q is the probability of a communication failure when only a subset of the intended receivers fail to receive the message. As in Section 4.3.1, we denote the three outcomes of the algorithm as `AG` , `AB` and `DG` respectively for *agreement on a value*, *agreement to abort* and *disagreement*. Also, OP, MP and PS are used to denote the optimistic, the moderately pessimistic and the pessimistic decision criterion.

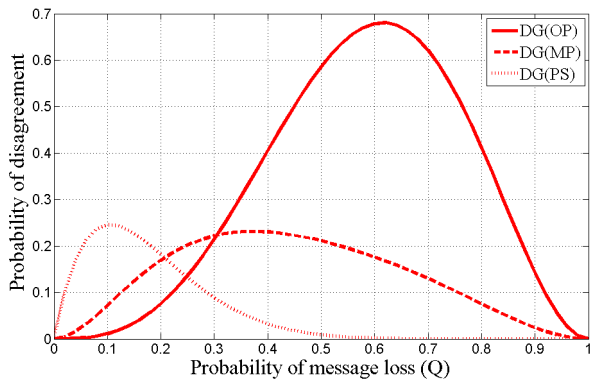
Fig. 4.4 shows the probabilities of the outcomes of the three decision criteria for a *1-of-3* selection algorithm with $R = 2$ as a function of Q. As we see in Fig. 4.4(a) for any value of Q, the optimistic decision criterion has the highest values of P_{AG} among the three decision criteria, while the pessimistic decision criterion has the lowest. The curve for the MP criterion lies between the two other curves rather close to the PS curve, which motivates the name moderately pessimistic. When it comes to the probability of agreement on abort, P_{AB}, shown in Fig.4.4(b), the behavior is as expected the reverse. Here P_{AB} is always higher for the PS criterion compared to other two criteria. Again, the curve for the MP criterion lies between the other two curves. As we see in Fig. 4.4(c), for all decision



(a) The probability of agreement on a value



(b) The probability of agreement to abort



(c) The probability of disagreement

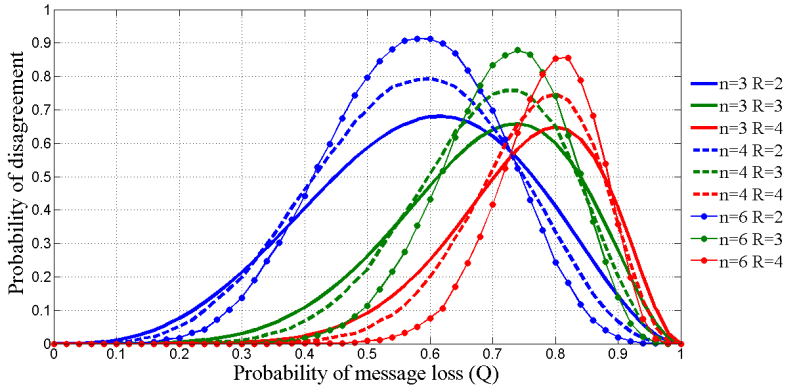
Figure 4.4: *Probability of an outcome for 1-of-3 selection algorithm for $R = 2$ with asymmetric failures.*

criteria the probability of disagreement shows a distinct peak. However, the maximum probability of disagreement varies significantly for different decision criteria. The peak value of P_{DG} for the OP criterion is as high as 0.68, while it is slightly less than 0.25 for the other two criteria. We also see that the peak for the OP criterion occurs for a significantly higher value of Q, around 0.62, compared to the other peaks. For the PS criterion the peak lies on the left side of the x-axis at $Q \approx 0.1$. Overall we see the same trend in the behavior of the three decision criteria here as for the symmetric failures in Fig. 4.1. The main difference between the two failure models is that the peaks of P_{DG} are significantly higher for asymmetric failures.

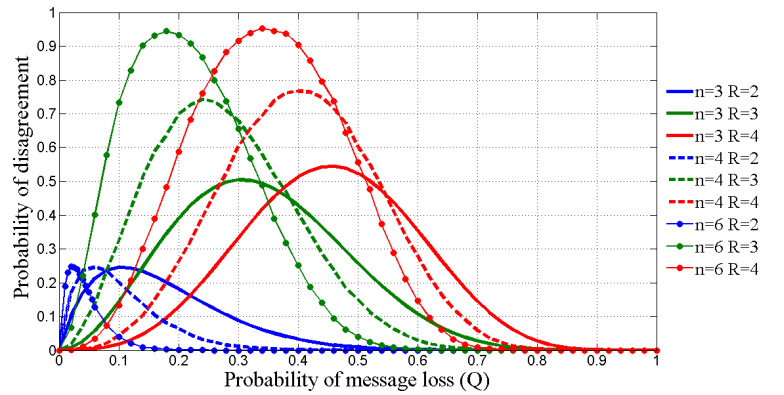
Fig. 4.5 illustrates how the probability of disagreement is affected by varying n, the number of processes, and R, the number of rounds. In this figure the curve for P_{DG} is shown for three different system configurations ($n = 3, 4, 6$) each running the *1-of-n* selection algorithms in two, three and four rounds of execution. The three given sub graphs 4.5(a), 4.5(b) and 4.5(c) show respectively the results for the three decision criteria.

Assuming a fixed number of processes, if we increase R, a process has more chance to complete its view and as a result the probability of agreement among processes increases. Consequently the peak of P_{DG} moves to the right on the x-axis and is thus achieved for larger values of Q. For the OP criterion, the maximum value of disagreement decreases slightly with increasing R, but for the PS and MP decision criteria, it increases significantly. We observe that for all decision criteria, increasing R does not guarantee lower values for P_{DG} if the probability of message loss (Q) cannot be limited.

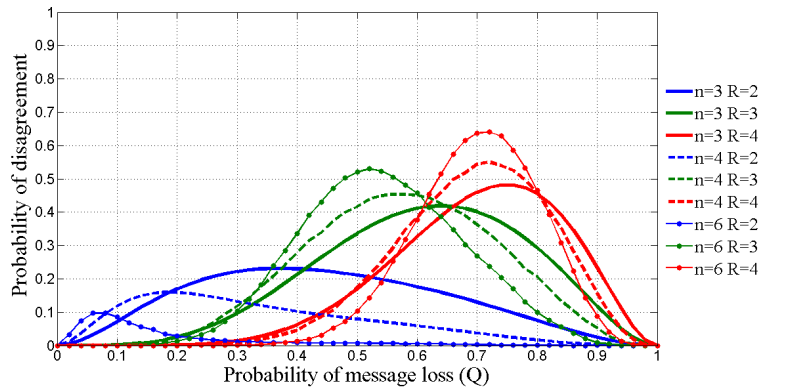
Varying n affects the probability of disagreement in a different way. As we see in Fig. 4.5(a) for the OP criterion, when we increase n, the maximum P_{DG} increases significantly (For $n = 6$ the probability of disagreement becomes larger than 80%). However, with increasing n the peak of disagreement does not significantly move to the left or right side of the x-axis.



(a) Optimistic decision criterion

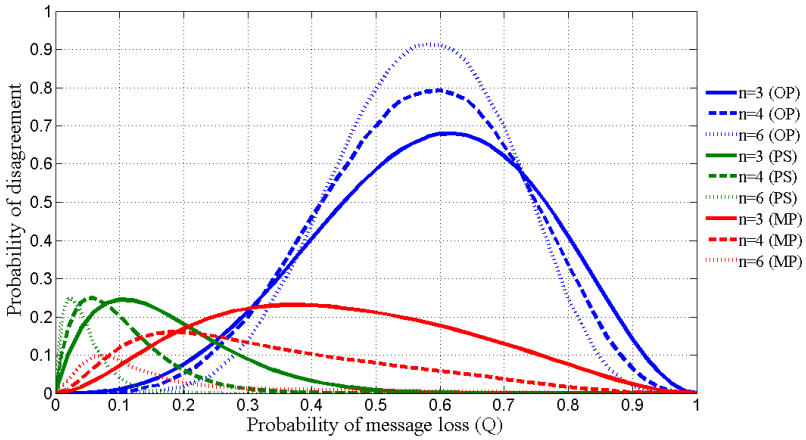


(b) Pessimistic decision criterion

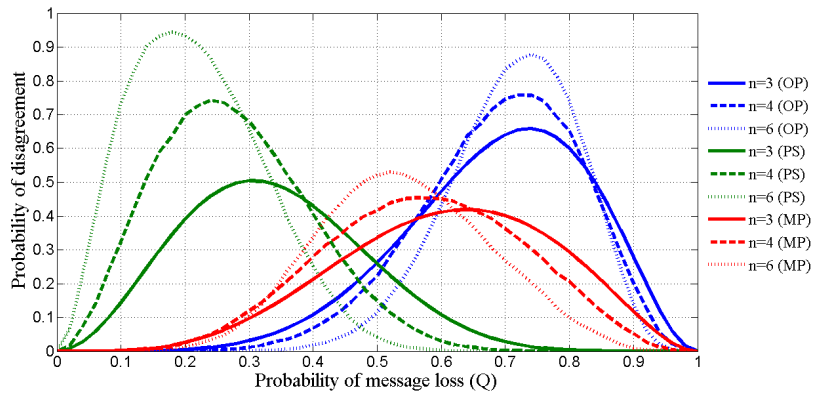


(c) Moderately pessimistic decision criterion

Figure 4.5: *Probability of disagreement for 1-of-n selection algorithms for* $(n = 3, 4, 6)$ *with* $R = 2, 3, 4$ *with asymmetric failures.*



(a) $R = 2$



(b) $R = 3$

Figure 4.6: *Probability of disagreement for the 1-of-n selection algorithms for ($n = 3, 4, 6$, $R = 2$ and $R = 3$) using different decision criteria under asymmetric failures.*

As it is shown in Fig. 4.5(b) and Fig. 4.5(c), for the PS and MP decision criteria, we distinguish two different behaviours depending on the choice of the R value. For $R = 2$, to have either the outcome `AG` or `DG` , all the messages should be successfully transmitted in the first round. Increasing n, the agreement region is reduced because it is less likely that a process in a system of many processes completes its view in the first round. As a consequence, with increasing n the curve of disagreement moves to the left w.r.t. the x-axis for $R = 2$.

For the PS criterion, when we increase n, the maximum probability of disagreement remains around the same value, that is 0.25. However, for the MP criterion, it declines significantly (not larger than 10% for $n = 6$). For $R > 2$, with increasing n the maximum probability of disagreement increases significantly for both the pessimistic and the moderately pessimistic decision criteria.

Fig. 4.6(a) shows a comparison of the three decision criteria for systems of three, four and six processes executing the *1-of-n* selection algorithm in two rounds. For any number of processes, the OP criterion has the highest peak of the probability of disagreement. For a given application with fixed n, if Q is unknown, the probability of disagreement can be minimized by adopting the moderately pessimistic decision criterion with $R = 2$. If the range of Q can be estimated, for values of Q lower than a specified threshold, the minimum disagreement is achieved by the OP criterion, while for higher values of Q the PS criterion has the lowest values of P_{DG}. For example, if $n = 3$, this threshold is around $Q = 0.24$, while for $n = 6$ it is around $Q = 0.15$.

Fig. 4.6(b) corresponds to the same settings of the system as Fig. 4.6(a) except for the number of rounds which is $R = 3$. As we see from the results, for $R = 3$, the OP criterion does not show highest peaks for P_{DG}, unlike the case for $R = 2$. Indeed, with varying R from 2 to 3 for systems of $n = 4$ and $n = 6$ processes, the PS criterion shows the highest P_{DG} compared to the other two decision criteria. Fig 4.6(b) shows that for a given number of processes ($n = 3$, 4 and 6), the MP criterion shows the

lowest maximum probability of disagreement.

From Fig. 4.6 we again see that increasing R does not necessarily result in lower probabilities of disagreement. For example, increasing R from 2 to 3, the pessimistic decision criterion shows much higher probabilities of disagreement.

PRISM Model for *1-of-3* Selection Algorithm

Model checking is to automatically verify a system model against its specified properties. System model refers to a mathematical computation model of a system. We model an abstract description of a system as a finite-state machine which presents the state transition model of the system using a formal modelling language. The properties of a system are also specified formally using a mathematical modelling language, such as temporal logic formulas. In case of probabilistic model checking, the state transition model contains stochastic behaviour. The probabilistic model checker performs the reachability analysis of the transition system. Additionally, it calculates the likelihood of reaching the states using numerical methods.

PRISM supports four different classes of models: discrete time markov chain (`dtmc`); markov decision process (`mdp`), continuous time markov chain (`ctmc`), and probabilistic timed automata (`pa`). As the consensus algorithm does not require the modelling of time intervals, among the supported models, we do not use `ctmc` and `pa`. Both `dmtc` and `mdp` allow the specification of the deterministic and the probabilistic transitions. For the probabilistic transitions, the choice of the next state is determined by a discrete probability distribution. The difference between `mdp` and `dmtc` is that `mdp` also allows the specification of non-deterministic transitions which are not associated with any probability distribution, while `dmtc` does not. Our algorithm does not require the use of non-deterministic transition. Therefore, we define the models of the *1-of-n* selection algorithms as `dmtc`.

A PRISM model consists of a set of modules representing different

components of a system model. In our model we define the processes as modules. With assuming asymmetric communication failures among the processes, a message sent from a process may be received or lost by a process independently from the other processes. This means that the number of states and transitions of a process's module depends on the number of processes in the network that is denoted as `N` in our PRISM models. We can show that generally, the process's module is composed of `N+3` transitions and `N+3` states. Fig. 4.7 illustrates the module of a process (p_i) in a system of three processes ($\{p_i, p_j, p_k\}$) executing the *1-of-n* selection algorithms under asymmetric communication failures. Table 4.4 describes the given states and transitions for process p_i.

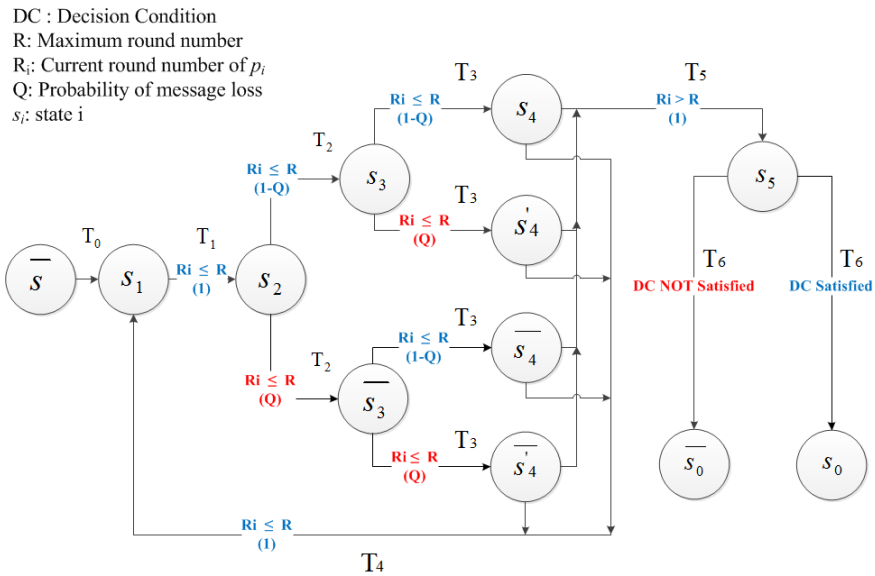


Figure 4.7: *The conceptual model of a process module p_i executing the 1-of-n selection algorithms for a system of three processes.*

a process p_i moves from its initial state ($\bar{s}$) to state s_1. Then it moves to state s_1 by broadcasting its message to the other processes with

Table 4.4: *Transition descriptions of the PRISM module of a process p_i in a system of three processes, $\{p_i, p_j, p_k\}$, executing a 1-of-n selection algorithm*

Transition	From	To	Prob.	Description
T_0	$\bar{s}$	s_1	1	p_i moves from its initial state to state s_1
T_1: $R_i \leq R$	s_1	s_2	1	p_i broadcasts its message
T_2: $R_i \leq R$	s_2	s_3	$1-Q$	p_i receives the message sent by p_j
T_2: $R_i \leq R$	s_2	$\bar{s_3}$	Q	p_i does NOT receive the message sent by p_j
T_3: $R_i \leq R$	s_3	s_4	$1-Q$	p_i receives the messages sent by p_k and p_j
T_3: $R_i \leq R$	s_3	s'_4	Q	p_i does NOT receive the message sent by p_k but received from p_j
T_3: $R_i \leq R$	$\bar{s_3}$	$\bar{s_4}$	$1-Q$	p_i receives message sent by p_k but does not receive from p_j
T_3: $R_i \leq R$	$\bar{s_3}$	$\bar{s'_4}$	Q	p_i does NOT receive messages neither from p_k nor from p_j
T_4: $R_i \leq R$	s_4, s'_4, $\bar{s_4}$, $\bar{s'_4}$	s_1	1	Not the last round: p_i continues to send, receive and compute messages
T_5: $R_i > R$	s_4, s'_4, $\bar{s_4}$, $\bar{s'_4}$	s_5	1	After the last round p_i computes the messages it received in round R
T_6: decision condition satisfied	s_5	s_0	1	p_i decides to elect p_i
T_6: decision condition Not satisfied	s_6	$\bar{s_0}$	1	p_i decides to abort

transition T_1. In the sequence, T_2 and T_3 are probabilistic transitions that model the choice between receiving (with probability of $1 - Q$) or losing (with probability of Q) the messages sent from other two processes (i.e. p_j and p_k). At any round except for the last one, the compute phase is performed by transition $T4$. In the last round, T_5 fires instead of T_4 and performs the round computation. At state s_5, with transition T_6, process p_i executes the given decision criterion algorithm and makes a decision (i.e., either to select a value or to abort) and moves to the final state, state s_0. We can show that for systems with larger number of processes, the probabilistic transitions, which are T_2 and T_3 for a system with three processes, are repeated $N - 1$ times.

We formally specify our probabilistic models based on the PRISM textual modelling language. Our PRISM model is composed of three parts: *declarations*, *modules* and *expressions*. Declarations contain the list of constant values and global variables. As mentioned before we model a process as a module. The message exchange among processes is modelled using global variables that are written/read by the modules. Also the synchronization among processes is achieved using a global variable (called `token`). We use expressions in order to avoid repetitions of the code in the module definition. Expressions are also used in defining the decision criteria. In AppendixA.2 we explain our PRISM model of the *1-of-n* selection algorithms for a system of three processes in detail.

4.4 Discussions

In general, the *optimistic* decision criterion shows a lower probability of disagreement compared to the *pessimistic* one when the probability of message loss is less than 30% to 70%. On the other hand, the *optimistic* decision criterion has a higher maximum probability of disagreement compared to the *pessimistic* criterion. We show that the outcome of the *moderately pessimistic* decision criterion generally lies in between the two other decision criteria.

The *moderately pessimistic* decision criterion shows the lowest peaks of the probability of disagreement, higher probability of agreement compared to the *pessimistic* decision criterion and lower probability of abort compared to the *optimistic* one.

According to the definition of the decision criteria, the *pessimistic* decision criterion could intuitively be chosen as the safest decision criterion among the three decision criteria. This is because using the *pessimistic* approach, a process with a complete view decides to select a value if and only if it has received the confirmations from all other processes that they also have the complete view of the system. However, we see from the results that after quite low values of the probability of message loss, if we increase the probability of message loss, the *pessimistic* criterion often results in abort. This means that even though the *pessimistic* approach shows lower probabilities of disagreement compared to the *moderately pessimistic* one, it actually results in more cases of abort which defer the process of selection.

Our results show that for asymmetric communication failures, all decision criteria show higher probabilities of disagreement compared to the symmetric failures. As the asymmetric failure model is more realistic for the applications with wireless communication, to have a better understanding of the behaviour of the *1-of-n* selection algorithms, we need an analytical analysis of the asymmetric failure model. To this end, our future work will involve deriving closed-form expressions to calculate the probabilities of each outcome of the algorithm under asymmetric failure assumptions.

Our observations also indicate that for all decision criteria, we have higher probabilities of disagreement under the asymmetric failure model compared to the symmetric failure model. Since we know that the asymmetric failure model is a more realistic model for systems with unbounded communication failures than the symmetric failure model, in order to have a better understanding of the behaviour of the *1-of-n* selection algorithms, we need to have an analytical analysis of the asymmetric failure

model. This motivates our future work to derive closed-form expressions to calculate the exact probabilities of each outcome of the *1-of-n* selection algorithms with asymmetric failure model. Deriving closed-form expressions for the asymmetric failure model is also necessary to reach our ultimate goal to design adaptive consensus protocols equipped with an on-line mechanism to monitor the main system parameters and select the best decision criterion on-line.

4.5 Chapter Conclusions

We present a probabilistic analysis of a family of *1-of-n* selection algorithms that aim to reach agreement among a set of n processes in the presence of unrestricted communication failures. We propose three different decision criteria for the *1-of-n* selection algorithms, called the *optimistic*, the *moderately pessimistic* and the *pessimistic* decision criterion.

Our results show that the choice of decision criterion significantly influences the probability of each outcome of the algorithms. However, we cannot claim that one decision criterion is better than the others in general. The choice depends on the size of the system, the quality of the communication links, and the performance requirements.

We can conclude that for all decision criteria increasing the number of rounds of execution results in higher probabilities of agreement and lower probabilities of aborts. For all decision criteria assuming a fixed value for R, if we increase the number of participating processes, n, we have higher probabilities of abort and lower probabilities of agreement.

Our observation motivates a future work to investigate the possibility of monitoring system parameters and selecting the best decision criterion on-line, by using the closed-form expressions for calculating the probability of disagreement.

A key feature of the *1-of-n* selection algorithms is that they ensure *safety*, i.e., all processes that select a value will select the same value. However, the above safety argument relies on the assumption that the

processes have the same view of the set of processes participating in the protocol, i.e., all processes know n. The problem of reaching agreement on this set of processes suffers from the same fundamental limitations as other types of consensus problems. Therefore, in the following chapters, we extend the reliability analysis of the *1-of-n* selection algorithm to include situations where the processes do not have the same view of the set of processes that participate in the protocol.

5

Group Formation

In this chapter, we address a specific agreement problem related to the design of self-organizing wireless systems, namely that of reaching agreement on the set of nodes (or processes) that are involved in bootstrapping a cooperative application. We call this problem the *group formation* problem, since it involves forming the group of nodes that are the initial participants, or members, of a cooperative application. For example, in a VTL application, bootstrapping the application occurs when one or more vehicles are approaching an intersection where no vehicles are present, or one where the vehicles in the vicinity of the intersection not yet have established a VTL. In order to bootstrap the application, the prospective VTL members must execute a group formation algorithm. Note that the group formation problem is different from the problem of *group member-*

ship. Group membership is the problem of *forming* and *maintenance* of a set of processes in a system and is first introduced in [17]. In a group membership problem, the processes who may dynamically leave and join the system must agree on the current set of the processes in the system, i.e. the departure and joining of the processes must be consistently detected by the members in the presence of communication delays and process failures.

However, in a group formation problem, the *maintenance* stage of a group membership is excluded. We assume a synchronous system of an unknown number of participating processes which are communicating in a fixed number of rounds in order to reach agreement on a common set of participants. We assume that the processes are fault-free, i.e. there is no process crash while the underlying communication links are unreliable and may result in any number of message losses.

A major challenge in designing a group formation algorithm is that the participating nodes initially have no knowledge of the number of the nodes that participate in the formation of the group, or the identity of these nodes. This is in contrast to many systems using wired communication, where the number of nodes in the system and their identities are known at the design time.

The problem of reaching agreement in a distributed system with an unknown number of participants has been studied before, e.g., in [11, 12]. Most previous research consider asynchronous systems with reliable communication channels, and focus on deriving necessary and sufficient conditions under which consensus is achievable. In contrast, we consider synchronous systems with unreliable communication and no process failures. We know from previous research [52, 55] that it is impossible to construct a synchronous algorithm that can guarantee consensus if there is no upper bound on the number of messages that can be lost during the execution of the algorithm. We argue that it is unrealistic to assume such an upper bound on the number of message losses in an automotive cooperative application, and thus our work is based on the assumption

that disagreement is an unavoidable outcome of a group formation algorithm in such systems. Therefore, we are interested in investigating the ways to minimize the probability of disagreement, and to mitigate the effects of disagreement.

We propose a group formation algorithm with two outcomes at the process level. Each process that executes the algorithm will either decide on a group (i.e. a set of nodes), or decide to abort. At the system level, i.e., when we consider the outcomes of all participating nodes, the algorithm have three main outcomes: (i) agreement on a set of nodes, (ii) agreement to abort, (iii) *disagreement*.

We categorize *disagreement* in two classes: *unsafe disagreement* and *safe disagreement*. In case of *safe* disagreement, one subset of the nodes decides on the same set of nodes while other nodes abort. In case of *unsafe* disagreement, at least two different subsets of the nodes decide on different sets. In order to reduce the probability of having *unsafe* disagreement, we propose a decision algorithm for the group formation algorithm. Our proposed decision algorithm relies on the use of an extra component, called an *oracle*. The oracles are local devices attached to each process and are used for detecting processes in the system. At the end of round R, a process that satisfies the decision criterion decides on a set and otherwise it aborts.

We perform a probabilistic analysis of the group formation algorithm using PRISM model checking tool [36]. We calculate the probability of each outcome of the algorithm for different configurations of a system and various qualities of the underlying network (i.e., the probability of message loss). Moreover, we derive generic closed-form expressions to compute the numbers of each outcome of the group formation algorithm as a function of the size of the system, n). Our results show that the probability of *unsafe disagreements* can be reduced using our proposed decision algorithm for different system settings.

The remainder of this Chapter is organized as follows. In Section 5.2, we formally present our system model, failure assumptions, and the group

formation algorithm. Section 5.3.1 presents an analytical analysis of the group formation algorithm. In Section 5.3.2, we present a probabilistic analysis of the group formation algorithm. We discuss the related work in Section 5.1. Finally in Section 5.4 we conclude and outline some directions for future research. An important challenge in solving the problem of group formation in a wireless system with unreliable links is the problem of network partitioning. Such a problem occurs when a system is virtually partitioned in two or more isolated (disjoint) networks due to communication failures. This is a common problem in mobile ad-hoc networks with high mobility of the nodes and unreliable communication environments [35]. In a partitioned network, the processes in each partition can only decide on a group of the processes that are present in that particular partition. As a result, in a partitioned network, the outcome of the group formation algorithm consists of more than one group of processes which can lead to unsafe situations. For example, in a VTL scenario, forming different groups isolated from one another can result in electing more than one VTL leader, i.e. we have *unsafe disagreement* among the vehicles.

There are methods suggested in the literature to predict the occurrence of a network partitioning such as in [59]. Authors in [59] propose a new characterization of group mobility based on the existing group-based movements of the mobile nodes. Such characterizations are used to provide parameters that are sufficient for network partition prediction. There is a large number of mobility models available for vehicular networks. Authors in [35] suggest an overview and a taxonomy of several mobility models available for vehicular networks with simulation. However, our goal is to investigate the reliability of a group formation algorithm under the assumption that there is no restriction on the level or pattern of the mobility of the nodes.

In order to reduce the probability of the occurrence of *unsafe disagreement* due to a partitioned network, we propose a group formation algorithm for a system of processes each augmented with a local device

called an *oracle*. The notion of an oracle is similar to the notion of the *participant detector* proposed in [11]. The participant detectors are defined as distributed oracles attached to each process to provide an initial *contact list of processes* to each process, whereas in our work, we assume that the local oracle of each process proposes a *value* to the process. This value is an estimation of the number of processes in the system[1]. So, we assume that the oracles only provide a number and not a set of processes. However for simplicity we assume that the set of processes seen by each process is always a subset of the group of processes detected by its oracle. We assume that the oracles are unreliable, i.e. they might underestimate or overestimate the actual number of processes in the system. So, in order to account for the unreliability of the oracles, we associate our proposed decision algorithm with a *correction parameter* for the oracles (For more details see Section 5.2). Later, in this Chapter, we show that the use of the distributed oracles augmented with the *correction parameter* can reduce the number of possible cases where *unsafe* disagreement can occur due to a partitioned network.

5.1 Related Work

In this chapter, we are interested in solving the problem of agreement on a set of values among the processes in a system with unrestricted communication failures. We assume that the identity and the number of participating processes is initially *unknown* to all processes.

The consensus problem in which the participants do not know a priori who the other participants are is first noted in [6] and is called *consensus with uncertain participants* (or CUP). In this problem, a process initiates the CUP protocol by proposing a finite set of nodes in its view as well as a value. Similarly each process participating in the CUP proposes a set and a value to others which are not necessarily identical to the

[1] In a VTL scenario, an oracle can be implemented as a physical equipment installed on top of each car such as a sensor or a camera.

proposed sets and values by other nodes. The authors of [6] suggest an early-deciding algorithm in a fail-stop model to solve the CUP problem. They assume that an unbounded number of participants may join, leave, or fail in the system. The underlying network is considered to be reliable and the failure detectors are perfect.

In [11], a consensus problem called *Fault-Tolerant Consensus with Unknown Participants* (or FT-CUP) is investigated. The FT-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 about the participants in the system. The authors of [11] consider an asynchronous system with reliable communication channels. Moreover, the processes are assumed to be fault-free and being equipped with participant detectors. They provide the necessary and sufficient conditions under which the consensus problem can be solved. The participant detectors are defined as distributed oracles that provide the processes with a set of initial contacts. The authors of [11] define various classes of participant detectors and finally solve the FT-CUP problem using what they call the *one sink reducibility* participant detector.

Greve and Tixeuil in [32] show that there is a trade-off between the synchrony requirement and knowledge connectivity among the processes in order to solve the CUP problem. They define the minimal synchrony requirements to solve the FT-CUP problem.

In [3], the FT-CUP is extended to a new problem called Byzantine Fault-Tolerant Consensus with Unknown Participants or BFT-CUP where the participants in the system can behave maliciously. The authors of [3] propose an algorithm to solve the BFT-CUP problem with defining the necessary and sufficient conditions for synchrony and knowledge requirements.

Some other examples of the previous works on the CUP problem include [5, 12, 60]. In most of the previous works, the CUP problem is solved with restricting the failure assumptions or with defining minimum

connectivity requirements.

In this work, we assume a wireless environment which is highly mobile and unpredictable and therefore the communication links cannot assumed to be reliable. Moreover, we consider no restrictions on the pattern or number of lost messages in a network. Our failure model is based on the model introduced by Santoro and Widmayer in [52] denoted as the *transmission fault model*. We know from the results given in [52] and [53] that any non-trivial form of agreement is impossible to solve if in a system of n processes, $(n-1)$ or more messages can be lost per communication round.

Our goal is to solve the problem of forming a group of processes which are going to participate in bootstrapping a cooperative application called the group formation problem. The problem of group formation is similar to the first step of a group membership problem where the processes in the system must form and maintain a set of processes among themselves, called a group [17]. Examples of previous works on the problem of group membership include [34, 51]. However, unlike the the group membership problem, we assume no process crashes in the system while we consider unrestricted communication failures among the processes. However, a constant failure of a process to communicate with others (which is due to message losses) can be implied as a process failure. Such a process will not be considered in the final group of processes.

5.2 Protocol Description

We consider a synchronous system consisting of a group of processes where neither the identity nor the number of the processes is initially known to any process. We formally consider a set of n processes denoted by $\mathcal{S} = \{p_1, p_2, \ldots p_n\}$ that executes a group formation algorithm with the aim of reaching agreement on a common group of processes which is a subset of $\mathcal{S}$. We propose a group formation algorithm that is based on the classical round-based computational model used by many researcher

such as in [14], [57] and [18].

Initially each process $p_i \in \mathcal{S}$ is only aware of its own identity in the system. Then the processes execute a deterministic group formation algorithm in R rounds of message exchange. 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 $(n-1)$ 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 a process, and that a receive omission failure is equivalent to the cases where only one process fails to receive a message.

We assume that all process are equipped with local oracles that are means for detecting other processes in a specific geographical area. Oracles provide their corresponding processes with an approximation of the set of processes in the system. We assume that oracles are unreliable in the sense that they may underestimate or overestimate the actual number of processes in the system.

5.2.1 The Group Formation Algorithm

Alg. 8 shows the pseudocode of the group formation algorithm that is run by each process in the system. Each process $p_i \in \mathcal{S}$ constructs a message $(msg_i = \{p_i, \Pi_i\})$ where p_i is the identity of the sender and Π_i is the set of processes that p_i currently sees in the system. The set Π_i is called the *view* of process p_i which is initially $\Pi_i = \{p_i\}$.

Algorithm 8 Generic group formation algorithm for p_i

```
msg_i ← {p_i, Π_i};
for r = 1 to R do
    begin_round
    send (msg_i);
    receive ();
    compute (msg_i);
    end_round;
end for
execute_decision_algorithm();
```

Each process executes the group formation algorithm for R rounds of message exchange ($R \geq 1$). Each round consists of three phases: *send, receive* and *compute*. During the *send* phase, each process p_i sends its message (msg_i) to all processes in its geographical vicinity by broadcasting. Note that some of the receivers may not receive this message under our assumed failure model. Then, in the *receive* phase, each process listens to the network to receive messages from other processes. Note that due to the existence of communication failures, a process may not receive some messages from other processes. Then, 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 (See Alg. 9).

Alg. 9 shows the psuedocode of the *compute* phase algorithm denoted by $\Pi_{\bigcup}$. Based on Alg. 9, a process p_i that received a message from a process p_j ($msg_j = \{p_j, \Pi_j\}$), computes the union of its current view set (Π_i) and the view set received from process p_j (Π_j) and updates Π_i accordingly.

Algorithm 9 Compute (msg_i): $\Pi_{\bigcup}$

```
1: for all p_j such that p_i has received msg_j = {p_j, Π_j} do
2:     Π_i ← Π_i ⋃ Π_j;
3: end for
4: msg_i ← {p_i, Π_i};
```

According to Alg. 8, at the end of round R, each process executes a

decision algorithm in order to decide on a set of processes (See Alg. 10). A trivial way to design the decision algorithm is that each process decides on its final view of the system at the end of round R. However, we show later that with such a design of the group formation algorithm, we can have high number of cases of disagreements among processes which lead to unsafe situations.

Alg. 10 shows our proposed decision algorithm that is run by each process at the end of R rounds of *send*, *receive* and *compute* phase. According to Alg. 10, each process p_i, at the end of round R, first queries its local oracle. The oracle of process p_i provides an estimation of the number of processes in the system which is denoted by o_i. In a system with n participants, we define an oracle as *correct* if it reports the actual number of processes in the system ($o_i = n$) while an *incorrect* oracle reports a different number from the actual number of processes in the system ($o_i \neq n$). As oracles are assumed to be unreliable, in order to account for possible inaccuracy of the oracles we consider a *correction parameter* denoted as c. We assume that the correction parameter is set at the design time and is the same for all processes[2]. 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. the oracles are always correct. If $c < 1$, then we assume that the oracles overestimate the number of processes in the system. Finally, if $c > 1$, then we consider that the oracles may underestimate the number of processes.

Based on Alg. 10 there are two possible outcomes for a process: decide to *select its set* or *abort*. a process p_i at the end of round R selects its view set (Π_i) if the number of processes in its view set (m_i) is greater than or equal to the number proposed by its oracle multiplied by the *correction* parameter c, i.e. $m_i \geq c * o_i$, otherwise it aborts. If m_i is less than $c * o_i$, then the number of nodes in p_i's view is smaller than the oracle's suggestion which is scaled by the correction parameter. Since p_i

[2]Considering different values of the *correction parameter* for different processes does not fundamentally change the analysis present in this thesis.

Algorithm 10 Decision Algorithm for p_i

$o_i \leftarrow p_i$ query its oracle
$m_i \leftarrow$ size of Π_i
$c \leftarrow$ *correction parameter*
if $m_i < c * o_i$ **then**
    abort;
else
    p_i selects Π_i;
end if

is not aware of as many processes as suggested by its oracle, p_i decides to abort; otherwise, p_i decides to select Π_i.

We assume three possible outcomes for our proposed group formation algorithm: (i) *agreement on a set* (Π), (ii) *agreement to abort*, (iii) *disagreement*. We classify *disagreement* to *safe disagreement* and *unsafe disagreement*. We have *agreement on a set* if at the end of the group formation algorithm, *all* processes decide on the same set including themselves (Π). We have *agreement on abort* if *all* processes decide to abort. We have *safe disagreement* if the processes in a subset of the system decide on that subset while the rest of the processes decide to abort. An *unsafe disagreement* occurs when the outcome of the group formation algorithm includes different non-empty sets of processes. It is evident that without considering *abort* also as a possible outcome of the decision algorithm, in cases of having disagreement the group formation algorithm always results in *unsafe* disagreement.

5.3 Probabilistic Analysis

In this Chapter, we present an analytical analysis of the group formation algorithm under the assumption that the system is partitioned into two or more isolated (disjoint) subsystems due to communication failures (See Section 5.3.1).

Following, in Section 5.3.2, we present a probabilistic analysis of the

algorithm in the presence of asymmetric communication failures using a probabilistic model checking tool. We analyse the algorithm under different configurations of the system with respect to the number of processes (n), the number of rounds (R) the *correction parameter* (c) and the probability of message loss (Q).

5.3.1 Analysis of network partitioning

An important challenge in solving the problem of group formation in a wireless system with unreliable links is the problem of network partitioning. We have a partitioned network when due to communication failures the network is virtually split in two or more isolated (disjoint) networks. This is a common problem in mobile ad-hoc networks with high mobility of the nodes and unreliable communication environments. In a group formation algorithm in case of having a partitioned network, the processes in each partition can only decide on a set of processes that are present in that particular partition. Therefore, the outcome of the group formation algorithm can consist of more than one group of processes which is unsafe. For example, in a VTL scenario, forming different groups isolated from one another can result in electing more than one VTL leader, i.e. we have *unsafe disagreement*.

There are methods suggested in the literature to predict the occurrence of network partitioning such as in [59]. Authors in [59] propose a new characterization of the group mobility based on the existing group-based movements of the mobile nodes models. Such characterizations are used to provide parameters that are sufficient to predict network partitioning.

However, our goal is to investigate the reliability of the proposed group formation algorithm under the assumption that there is no restriction on the level or the pattern of the mobility of the nodes. Therefore, we consider all possible cases a network can be partitioned due to the communication failures. We present the effectiveness of our proposed algorithm in reducing the probability of *unsafe disagreement* in a net-

work which is split into disjoint networks due to communication failures. First, we determine all possible ways a given set of n processes can be partitioned in k disjoint subsets where $1 \leq k \leq n$. Second, we count the percentage of the cases out of all these possible ways where we have *safe* and *unsafe disagreement* using our proposed algorithm. We count the number of ways a set of n processes can be partitioned into exactly k non-empty subsets using the Stirling number of second kind [50], denoted by $S(n, k)$. The total number of possible ways a set of n nodes can be partitioned in disjoint subsets is given using the well-known *Bell number* [28], denoted by B_n, as follows:

$$B_n = \sum_{k=1}^{n} S(n, k)$$

where the Stirling number $S(n, k)$ is recursively computed as follows:

$$S(n, 1) = S(n, n) = 1$$

$$S(n, k) = S(n-1, k-1) + S(n-1, k)$$

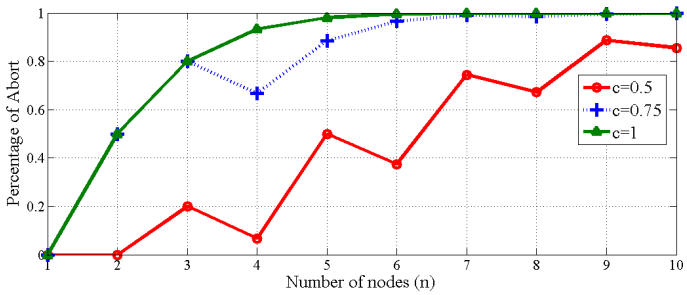
For example the Bell number for a set $\mathcal{S} = \{p_1, p_2, p_3\}$ with three elements ($n = 3$) is $B_3 = 5$. The five possible partitions are given in Table 5.1.

We coded a simple program in Matlab in order to determine all different cases at which a network of n processes is partitioned. For these cases, executing the group formation algorithm given in Alg. 8, we computed the number of cases of each outcome of the algorithm, i.e. *agreement on abort*, *unsafe disagreement* and *safe disagreement*.

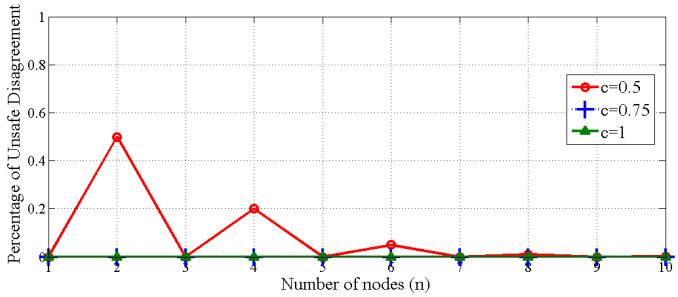
Partition 1, one subset	$\{p_1, p_2, p_3\}$
Partition 2, two subsets	$\{p_1\}\{p_2, p_3\}$
Partition 3, two subsets	$\{p_2\}\{p_1, p_3\}$
Partition 4, two subsets	$\{p_3\}\{p_1, p_2\}$
Partition 5, three subsets	$\{p_1\}\{p_2\}\{p_3\}$

Table 5.1: *$B_3 = 5$ possible ways to have a partitioned network of $n = 3$ processes.*

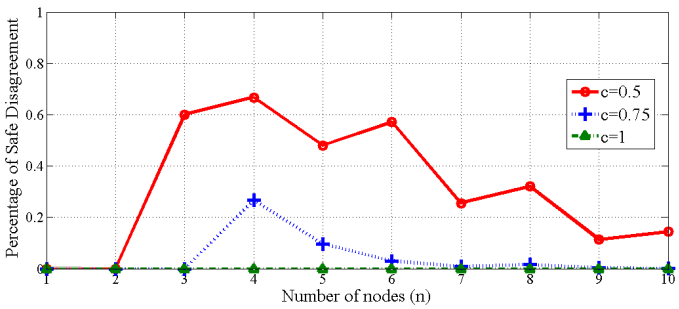
Fig. 5.1 shows the percentage of each of these outcomes for the proposed group formation algorithm considering three different values of the *correction parameter* ($c = 0.5, 0.75, 1$) and 10 values of the size of the system, i.e. $n = 1, 2, \ldots 10$. The values of the local oracle are considered to be *correct* for *all* the processes, i.e. $\forall_{p_i \in \mathcal{S}}$, we set $o_i = n$.



(a) Abort



(b) Unsafe Disagreement



(c) Safe Disagreement

Figure 5.1: *A comparison of three outcomes of the group formation algorithm varying c, the percentage of each outcome out of the total number of outcomes for a* ***partitioned*** *system of n processes with* ***correct*** *oracles ($o_i = n$).*

The percentage of different outcomes presented in Fig. 5.1 is generated by executing the decision algorithm given in Alg. 10 for all possible cases of partitioning of a system of n nodes. For example, consider that a network of three nodes $\mathcal{S} = \{p_1, p_2, p_3\}$ is partitioned to two subsets $\{p_1, p_2\}$ and $\{p_3\}$. In this case, the views of p_1 and p_2 are the same $\Pi_1 = \Pi_2 = \{p_1, p_2\}$ while process p_3 will only have itself in its view, i.e. $\Pi_3 = \{p_3\}$. We evaluate the decision criterion for each of the three processes using c and $o_i = n$. According to Alg. 10 if $c > 1$ and the oracles are *correct* then all processes will abort. On the other hand, if $c = 1$ and the oracle values are *correct* (i.e. $o_i = n$), a process p_i will decide on its view (i.e. Π_i) if it has all other processes in the network in its view ($m_i = |\Pi_i| = n$). This is because if $m_i = n$ and $c = 1$ the decision criterion given in Alg. 10 does not hold (i.e. $m_i < c * o_i$).

It is evident from Fig. 5.1 that as the correction parameter (c) increases from 0.5 to 1, the percentage of *agreement to abort* increases for most values of n. This is because as c approaches to 1, considering that oracle values are *correct* ($o_i = n$), the value of $c * o_i$ becomes closer to the accurate value (i.e. $c * n$) and as a result more processes make correct decisions. A process p_i with a *correct* oracle value will decide on Π_i if $|\Pi_i| \geq c * n$. For large values of c, the value of $c * n$ becomes larger. As in a partitioned network it is less likely that the size of Π_i is larger than or equal to $c * n$, it is more probable to have all processes deciding to abort.

Similarly, Fig. 5.1(b) shows that with increasing the correction parameter from $c = 0.5$ to $c = 1$, the percentage of *unsafe disagreement* cases decreases. As we see from Fig. 5.1(b) for $n = 4$ and correct oracles, we have 20% and 0% cases of *unsafe disagreement* respectively for $c = 0.5$ and $c = 1$.

We have *unsafe disagreement* if at least two processes decide on two *different* view sets. In a partitioned network of n processes, two processes decide on two different sets if they belong to two different partitions. Moreover, according to Alg. 10, each partition must consist of a number of processes which is greater than or equal to $c * o_i$ where $o_i = n$ in this

case. Therefore, the total number of processes in the given two partitions must be at least $(2{*}c{*}n)$. Based on the definition of network partitioning, the sum of nodes in all subsets of a partitioned network is equal to n. So, in order to have *unsafe disagreement* we must have $2 * c * n \leq n$, which implies that $c \leq 0.5$ is a *necessary* condition to have *unsafe disagreement* for a partitioned network of n processes with correct oracle values (i.e. $o_i = n$). Note that the plot for $c = 1$ and $c = 0.75$ in Fig. 5.1(b) coincides with the x-axis.

Another trend from Fig. 5.1 is that for larger systems, i.e. systems with larger number of nodes (n), the percentage of *agreement on abort* is higher while the percentage of *unsafe disagreement* cases becomes lower. This is because for larger values of n there are more possibilities of having network partitioning and there are relatively higher numbers of subsets for many of these partitions (See [28]). A relatively higher number of disjoint subsets of a partition implies relatively fewer processes in each subset. A subset with relatively smaller number of processes is more likely to have less than $(c * o_i)$ number of processes where $o_i = n$. Therefore, a process with smaller number of nodes in its view will abort. Since processes in many of the cases of network partitions abort rather than decide on the view, with increasing n the percentage of *agreement on abort* cases increases in Fig. 5.1(a) and the percentage of *unsafe disagreement* decreases (See Fig. 5.1(b)).

Fig. 5.1(c) shows the percentage of *safe disagreement* for different values of n. According to the decision criterion given in Alg. 10, a process p_i using *correct* oracle with $c = 1$ decides on its view only if $|\Pi_i| = n$, i.e. p_i sees all other processes in its view. On the other hand, by definition we have no network partitioning if a process has all the n processes in its view (See Section 5.3.1). We have *safe* disagreement, if at least one process decides to *select* and the remaining processes *abort*, which is an impossible case in a partitioned network of n processes if the oracle values are *correct* and $c = 1$. So for $c = 1$ and *correct* oracles we have no *safe* disagreement case (note that the plot for $c = 1$ in Fig. 5.1(c) coincides

with the x-axis). As the *correction parameter* increases from $c = 0.5$ to $c = 0.75$ in Fig. 5.1(c), the percentage of *safe* disagreement decreases. This is because as c becomes larger, the percentage of *abort* increases (See Fig. 5.1(a)). Therefore, the percentage of *safe* disagreement is relatively smaller for larger values of c. In summary, a trivial decision making algorithm without using an oracle will have 100% *unsafe* disagreement in case of having network partitioning. As it is shown in Fig. 5.1, a significant percentage of network partitioning cases can be safe: either all nodes *abort* or we have *safe* disagreement.

Other Pathological Cases

The analysis of network partitioning leads us to investigate whether there are other such pathological cases: *Can we have unsafe disagreement when there is no network partitioning?* We found an affirmative answer. In addition to network partitioning, *unsafe disagreement* can also occur due to *asymmetric* communication failures among the processes. In case of having an asymmetric communication failure, a subset of the intended receiving processes fail to receive a message while other processes succeed to receive.

We consider the cases where some processes have different view sets which are not disjoint. For example consider a system of four processes $\{p_1, p_2, p_3, p_4\}$ where at the end of round R, $\Pi_1 = \{p_1, p_2, p_3\}$ and $\Pi_2 = \{p_2, p_3, p_4\}$. So, the two sets of Π_1 and Π_2 are unequal but not disjoint as they both include p_2 and p_3, i.e. $\Pi_1 \cap \Pi_2 \neq \emptyset$. Note that $m_1 = m_2 = 3$ and if $o_1 = o_2 = 4$ and $c = 0.5$, then the condition $m_i > c * o_i$ holds for both p_1 and p_2 and as a result they both decide on their views, i.e. p_1 decides on Π_1 and p_2 decides on Π_2. Since $\Pi_1 \neq \Pi_2$, we have *unsafe disagreement*.

In order to have a better understanding of the behaviour of the proposed group formation algorithm, in [23] we present an analytical analysis to compute the fraction of each possible outcome of the algorithm for a system of n processes. Such fraction is computed by considering all dif-

ferent possible combinations of the views of n processes at the end of R rounds. To make our results as general as possible, e.g. independent of the number of rounds or the quality of the network, we conduct our analysis based on three following factors F1–F3:

F1: Any value of the probability of message loss is *equally* likely, i.e. $0 \leq Q \leq 1$.

F2: Each process p_i can have any possible view set at the end of round R, i.e. $|\Pi_i| \in [1, n]$.

F3: The value of o_i provided by the oracle of process p_i can have n_{upper} different possible values. The value of n_{upper} is set by the designer of the oracle.

Factor F1 is important so that our results is applicable for any quality of the network. We also know that the quality of the network may vary from one intersection to another. For example in a VTL scenario, in an intersection with no communication obstacles such as large buildings or mountains, the probability of message loss can be relatively small compared to that of a congested network in a crowded city center with lots of cars, buildings, etc.

We define Factor F2 based on Factor F1. If the quality of a network is bad (i.e., Q is large), process p_i might not be able to communicate with any process in the system due to large number of message losses. As a result, the view of process p_i may include only itself, i.e. $\Pi_i = \{p_i\}, |\Pi_i| = 1$ at the end of round R. On the other extreme, in a system with low probability of message loss, Π_i may consist of all processes in the system, i.e. $\Pi_i = \{p_1, .., p_n\}, |\Pi_i| = n$ at the end of round R.

In a VTL implementation the value of n_{upper} introduced in Factor F3 can represent the largest possible number of legs for an intersection. The performance of an oracle can vary from one intersection to another depending on the external environment. As a result, an oracle may perform differently in different situations, i.e. it may be *correct* or it may

overestimate or underestimate the number of vehicles in an intersection. Moreover, the oracle value of one vehicle in a particular intersection may be different from that of another vehicle, i.e. they might report different values to their corresponding vehicles. Factor F3 captures the applicability of our results for any possible values of the oracle.

For more details on the proposed formulas for analytical analysis of the group formation algorithm based on the above assumption we refer the interested reader to one of our previous works in [23].

5.3.2 Probabilistic analysis of disagreement

In this section, we present several graphs showing how the probability of *unsafe* disagreement varies for different configurations of the system with respect to the number of processes (n), the number of rounds (R) the *correction parameter* (c) and the probability of message loss (Q). We compute the probabilities of all four possible outcomes: *agreement on the set* Π ('AG'), *agreement on abort* ('AB'), *safe disagreement* ('SD') and *unsafe disagreement* ('UD'). The sum of probabilities of *safe* and *unsafe* disagreements is the probability of disagreement and is denoted by the 'DG' label.

In order to calculate the probabilities of each outcome of the group formation algorithm, we modelled the algorithm using Discrete Time Markov Chains (DMTCs) probabilistic models with PRISM [36]. We use probabilistic transitions in order to model the probability of loss of a message. In a probabilistic transition, the choice of the next state is determined by a discrete probability distribution. A limitation of using PRISM is that due to the problem of state space explosion, we are only able to calculate probabilities for a system of 3 processes. This is because with increasing the number of processes ($n > 3$), the number of reachable states of the model increases considerably which makes the verification process infeasible. For systems with larger number of processes, PRISM can estimate the outcome using simulation, with a defined tolerance and interval of confidence. Here, in order to keep the uniformity of the results

for different sizes of the system, we obtain all the results using probabilistic simulation of PRISM.

Fig. 5.2 shows the outcomes of the group formation algorithm **with** and **without** using an oracle for a system of 4 processes running the algorithm in two rounds, i.e. $R = 2$. In a system executing the group formation algorithm without using oracles, as there is no decision criterion defined for the processes, all processes after executing R rounds of message exchange decide on their final view set. This means that no process will *abort* and as a result, there is no case of *safe* disagreement, i.e. all disagreement cases are *unsafe*.

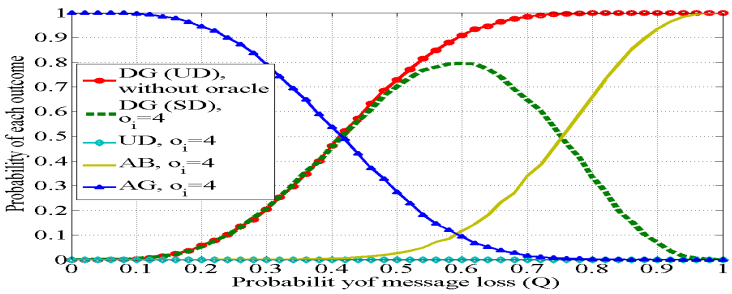
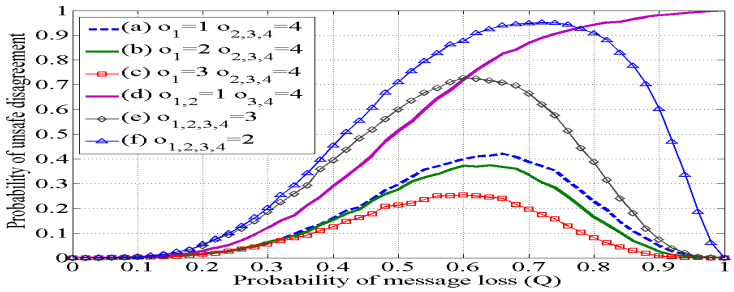


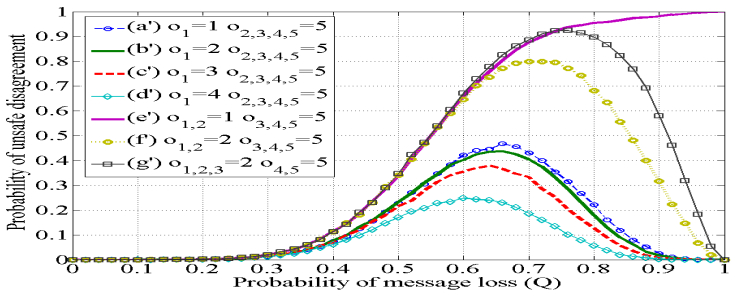
Figure 5.2: *Probability of each outcome of the group formation algorithm as a function of Q* ***with*** *and* ***without*** *using oracles, ($n = 4, R = 2$)*

As we see in Fig. 5.2, for the system with *correct* oracles ($o_i = 4$) and $c = 1$, the probability of *unsafe* disagreement ('UD') is zero for all values of Q. This means that the curve for the probability of 'DG' for this system indicates only the *safe* disagreement cases ('SD') while the curve showing the probability of 'DG' for the system **without** using oracles indicates *unsafe* disagreements only. Comparing the outcomes of the group formation algorithm with and without using oracles, we see that for $Q \leq 0.48$ the probability of 'DG' of the two systems are about the same while for $Q > 0.48$ the system with using oracle shows lower probabilities of 'DG' and positive probabilities of abort ('AB').

In the following two figures, we present our results to investigate the behaviour of the group formation algorithm for the cases where the oracle values are *incorrect*. In order to have a better understanding of the behaviour of the algorithm using *incorrect* oracles, we set the *correction* parameter to 1.



(a) $n = 4, R = 2$



(b) $n = 5, R = 2$

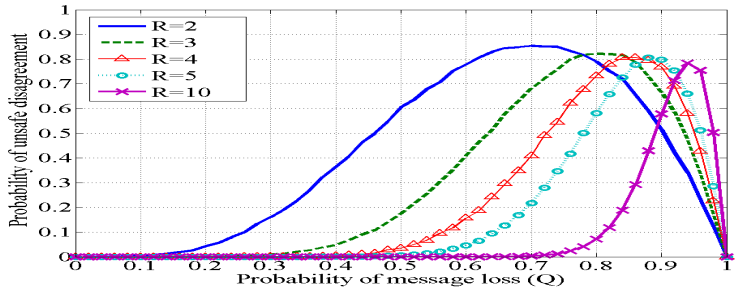
Figure 5.3: *Probability of unsafe disagreement for the group formation algorithm as a function of Q with* ***incorrect oracles****,* $c = 1$.

Fig. 5.3 shows the probability of *unsafe* disagreement for two systems of respectively $n = 4$ and $n = 5$ executing the group formation algorithm for $R = 2$ rounds using *incorrect* oracles. Fig. 5.3(a) shows 6 cases out of total 256 possible combinations of the oracle values for a system of 4 processes. The first three settings of the oracles (a, b and c), indicate the cases where one of the oracles is *incorrect* and three others are *correct*.

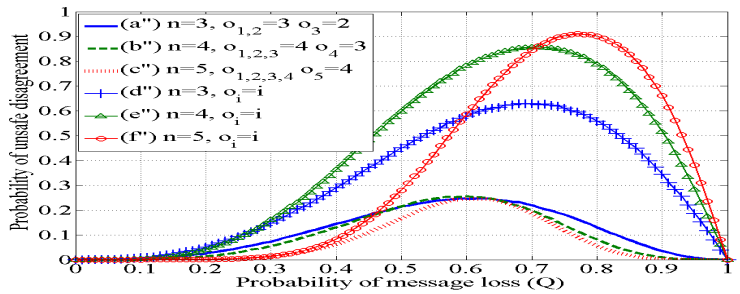
As expected, we have the lowest peak of *unsafe* disagreement for the case where only one of the oracles underestimate the actual number of processes by one. (See curve (c) in Fig. 5.3(a) where $o_1 = 3$ and the rest of the oracles are correct $(o_{2,3,4} = 4)$.) As we see from the curves shown in a and b with decreasing the oracle value of process p_1 to $o_1 = 1$ the peak of 'UD' rises upto around 0.95. We see the same trend in Fig. 5.3(a) given in the curves (a', b', c', d') for a system of $n = 5$.

Case (d) in Fig. 5.3(a) shows the probability of 'UD' for a system of $n = 4$ with a combination of the oracle values where two of the oracles have the minimum possible values $(o_{1,2} = 1)$. As we see from curve (d) the probability of *unsafe* disagreement can go upto 1 for large value of Q. We can show that for any combinations of the oracles where there are two or more than two oracles with the value of 1, the probability of *unsafe* disagreement can rise upto 1. This is because for $Q = 1$ where all messages are lost for all processes we have $\forall i, m_i = 1$, i.e. all processes have only themselves in their views. Therefore, if the oracle value is 1 for more than one process, we have more than one process which decides on a set including only itself. Therefore, we have *unsafe* disagreement among the processes. We can see the same behaviour for a system of five nodes given in curve (e') in Fig. 5.3(b). From the results given in Fig. 5.3, we can conclude that with changing the decision algorithm slightly for cases where a process p_i sees only itself $(m_i = c * o_i)$ to abort, we can have lower probabilities of *unsafe* disagreement. Such a change in the decision criterion affects the results mainly for large values of Q where many processes fail to establish communication with the rest of the network. As expected, we have higher probabilities of *unsafe* disagreement for combinations of the oracle values where more oracles are incorrect (See the curves e, f, f', g').

Fig. 5.4(a) shows the effect of increasing R, the number of rounds of execution, for a system of four nodes with *incorrect* oracles. We choose the set of oracles as $o_i = i$ where only process p_4 has the correct value $(o_4 = 4)$. As we see, increasing R results in lower probabilities of 'UD' for



(a) Varying R, $(n = 4, R = 2, 3, 4, 5, 10, o_i = i)$



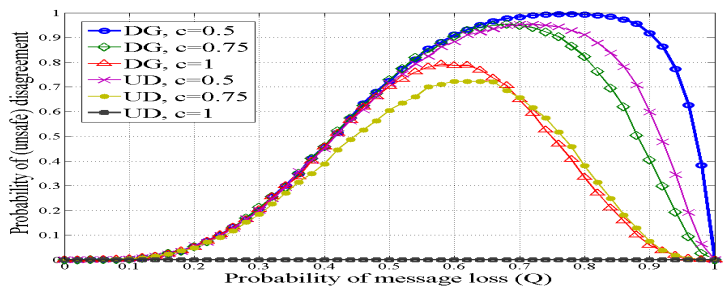
(b) Varying n, $(R = 2, n = 3, 4, 5)$

Figure 5.4: *Probability of unsafe disagreement for the group formation algorithm as a function of Q with varying n and R using* ***incorrect oracles****,* $c = 1$.

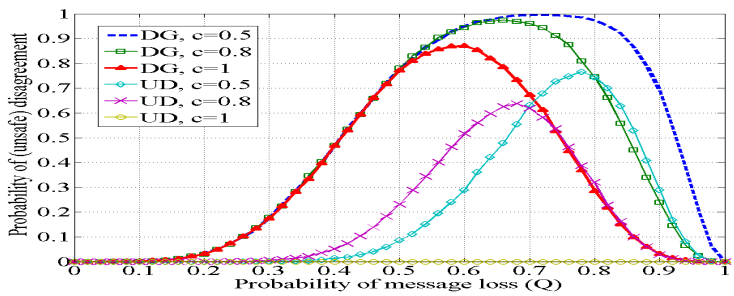
larger values of Q. This is because with more rounds, the probability of receiving the information about yet unknown participants in the network increases. However, safety-critical applications with real-time constraints may not always permit to use very large values of R.

Fig. 5.4(b) shows how the probability of 'UD' varies for different sizes of the system and different settings of the oracle values executing the algorithm in 2 rounds. The given curves in a", b" and c" are respectively for systems with 3, 4 and 5 processes and a set of oracle values where only one of them is underestimating the actual number of processes by 1. As we see from the results the peak of 'UD' is around the same for

all three systems (≈ 0.25). An interesting observation from the results in Fig. 5.4(b) is that for larger systems with only one incorrect oracle ($o_1 = n-1$), we have lower probabilities of *unsafe* disagreement for most values of Q. The given results in d", e" and f" are for the same systems but for different combinations of the oracle values ($o_i = i$). As expected, with larger n, we have higher probabilities of *unsafe* disagreement.



(a) $n = 4$



(b) $n = 5$

Figure 5.5: *Probability of (unsafe) disagreement for the group formation algorithm as a function of Q with varying c using* ***correct oracles*** *($o_i = n$), for $n = 4$ and $n = 5$ and $R = 2$.*

Fig. 5.5 shows the probabilities of total *disagreement* and *unsafe* disagreement for two systems of $n = 4$ and $n = 5$ and $R = 2$. We set the values of the oracles to the *correct* values and vary the *correction* parameter (c). From the results given in Fig. 5.5(a) and Fig. 5.5(b), we see that

for values of c larger than 0.75 and 0.8 respectively for $n = 4$ and $n = 5$, the peak of the probability of *unsafe* disagreement shows a distinct descent. This is expected as with increasing the value of c, after some point the decision criterion ($m_i >= c * o_i$) will not hold for many processes in the system and as a result many processes *abort*. Consequently, after certain values of c, we will have 100% of the probability of *abort*.

From the results shown in Fig. 5.5, we can also conclude that for $n = 4$ and $R = 2$ if the multiplication of the value of the oracle and the *correction parameter* is 3, i.e. $c * o_i = 0.75 * 4$, we have the lowest peak of the probability of *unsafe* disagreement while for $c * o_i > 0.75 * 4$ the probability of 'UD' is always zero. Similarly for a system of $n = 5$ and $R = 2$ we have the lowest peaks of 'UD' for $c * o_i = 0.8 * 5$ while for $c * o_i > 0.8$, we have 100% of aborts.

5.4 Chapter Conclusions

We have presented a group formation algorithm for bootstrapping the self-organizing cooperative automotive applications. Since these applications can experience massive communication failures, we investigated the algorithm's behaviour in the presence of an arbitrary number of messages losses. We know from previous research that it is impossible to construct a group formation algorithm that can guarantee consensus in the presence of massive communication failures. Therefore, we focused our analysis on the probability that the algorithm results in disagreement. To facilitate this analysis, we present a new classification of the disagreement cases: *unsafe* and *safe* disagreement.

The algorithm has two outcomes at the process level: a process either decides on a group or it aborts. To make this decision, the algorithm uses a local oracle that provides an unreliable estimate of the number of nodes that are present in the system. In its baseline version ($c = 1$), a node executing the algorithm will decide to abort if the value provided by its local oracle is greater than the number of participants the node

has heard from.

The main conclusions of our analysis are as follows: i) *Safe disagreement* is the only form of disagreement that occurs if the local oracles of all participants provide a correct estimate of the number of nodes in the system,

ii) *Unsafe disagreement* (i.e. when two or more nodes make inconsistent decisions about the group membership) occurs only in the cases where the local oracles provide an estimation which is less than the real number of participants,

iii) The probability of *unsafe disagreement* is high (in case it is not equal to 0) when the probability of receive omissions is large ($> 40\%$); the maximum value vary from around 25% to 98%, and occurs when the probability of a receive omission is between 50% and 95%.

In general, the probabilities of both *unsafe* and *safe* disagreement depend on three parameters: the number nodes in the systems, the number of rounds of message exchange used by the algorithm, and the accuracy of the local oracles.

6

*1-of-** Selection

In this chapter, we address the problem of reaching consensus on a value among the proposed values by the nodes of a self-organizing system with unreliable links. In a self-organizing system, the set of participating nodes is initially unknown to all nodes. In Chapter 4, we proposed a family of consensus algorithms called the *1-of-n* selection algorithms to solve the problem of selecting one value among n proposed values. The design of the *1-of-n* selection algorithms was based on the simplifying assumption that the participating nodes in the algorithm know n. For a self-organizing system, we introduce the problem of *1-of-** selection which is a variation of the problem of *1-of-n* where the $*$ symbolizes the fact that n is initially unknown to all participating nodes.

The *1-of-** selection problem is a consensus problem of type *Consen-*

sus with Unknown Participants (CUP) [11]. 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. In order to solve the *1-of-** selection problem, we propose a family of consensus algorithms called the *1-of-** selection algorithms. The design of the *1-of-** selection algorithms is motivated by the fact that, in cooperative applications based on wireless ad-hoc networks, it is unrealistic to assume that the exact set of participants is initially known to all participants. This is due to several factors such as the high mobility of the nodes in such networks or the existence of lossy links among the nodes which can result in partitioning the system into several networks where some nodes are not aware of the existence of some others for a period of time [35].

We propose the *1-of-** selection algorithms to be run as the core logic of a VTL leader election protocol where the participating nodes should reach consensus on a leader among themselves. We know that it is impossible to gaurantee agreement in systems with lossy links [52]. Each node (or process) that participates in a *1-of-** selection algorithm executes the algorithm for R rounds of message exchange using an unreliable network. At the end of round R, depending on the specified decision criterion[1] for the algorithm, each process either decides to *select a leader* 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 a *1-of-** selection algorithm, it decides to abort. Likewise, we define a process *live* or *non-aborting* if it decides to select a leader.

Considering the outcomes of all participating nodes, the *1-of-** selection algorithms have three main outcomes: (i) *agreement on a leader*, (ii)

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

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 a leader while others decide to abort.

For the *1-of-n* selection algorithms introduced in Chapter 4 the safety is ensured i.e., all processes that select a value will select the same value. However, the above safety argument relies on the assumption that the processes have the same view of the set of processes participating in the protocol, i.e., all processes know n. On the other hand, in the design of the *1-of-** selection algorithms, under the assumption of not knowing n and the existence of lossy links, we cannot guarantee safety. Nevertheless, it is useful to further classify disagreement cases as *safe* and *unsafe* as follows[2]

Unsafe Disagreement We have unsafe disagreement if there are at least two processes deciding on 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 unsafe disagreement, 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 process used for detecting other nodes in the system. The oracle can be a combination of a set of on-board cameras and sensors attached to a process (or car). The oracle is assumed to be unreliable and may report incorrect information,

[2]In Chapter 5, we also categorized the disagreement cases to safe and unsafe disagreements yet with different definitions.

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

i.e. it might underestimate or overestimate the number of processes (cars) in the system. In order to account for the inaccuracy of the local oracles we use a *correction parameter*. We propose three decision criteria to solve the problem of *1-of-** selection called the *optimistic**, the *pessimistic** and the *moderately pessimistic** algorithm.

One important problem in the design of a leader election protocol for a VTL system is when an aborting process (or vehicle) is elected as the VTL leader by some other vehicles. In such a situation, the vehicles will be waiting to receive traffic light orders from a vehicle which has decided to abort. Therefore, we need to propose a VTL leader election protocol that can prevent the problem of vehicles waiting to receive traffic signals from an aborting process for an infinite amount of time, i.e. the problem of *starvation*.

The design of the *1-of-** selection algorithm as the core logic of a VTL leader election protocol raises new research questions some of which are explained briefly in the following:

- What are the probabilities of each outcome of the proposed *1-of-** selection algorithms as a function of the probability of message loss among the processes?
- How does the setting of different system parameters, e.g. the number of communication rounds, the values reported by the local oracles and the correction parameters affect the probability of each outcome of the algorithm?
- How does the behaviour of an algorithm differs with different number of participants?
- How can the design of a VTL leader election protocol minimize the effect of having unsafe disagreement, when there are more than one non-aborting process acting as the VTL leader?

The rest of this chapter is organized as follows: Section 6.1 presents the related works on the design of leader election algorithms for systems

with unknown participants and lossy links. In Section 6.2, we explain our proposed VTL leader election protocol in more details. We formally specify our system model and failure assumptions in Section 6.3. We explain our proposed *1-of-** selection algorithms in Section 6.3.1. In Section 6.4, we present a probabilistic analysis of the *1-of-** selection algorithms for different settings of the system parameters and under different probabilities of message loss. In Section 6.5, we present some discussions and finally our conclusions in Section 6.6.

6.1 Related Work

There are several papers in literature on the problem of leader election in dynamic networks with lossy links such as in [19, 26, 39, 41, 47, 58]. In such networks, the nodes can dynamically join and leave the network, e.g. mobile ad-hoc networks. Most of the proposed solutions to the leader election problem for such networks are based on defining restrictions on the communication failure model among the nodes of the system. For example the authors of [26] 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 Santoro and Widmayer's impossibility results for systems with unrestricted communication failures [52], the election of a unique leader can not be guaranteed. For this, the authors of [26] 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.

The authors of [5] solve the problem of eventual leader election in dynamic and unknown mobile network under a communication model called Time-Varying Communication Graph. In this model, it is assumed that the network remains connected over time.

In [26] a protocol is proposed 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 communication failure assumptions, this can not be guaranteed. For this, the authors introduce the notion of a *stable* partition in which all processes are connected to each other and are required to elect a leader within a bounded time.

In [19], a self-stabilizing leader election algorithm for an ad-hoc network is proposed that can tolerate multiple concurrent topological changes of the network. The suggested algorithm uses an approach based on directed acyclic graph[4] with the advantage of detecting partitions automatically using the TORA mechanism[5].

In [40], the TORA mechanism is used to elect a unique leader in each connected component of an ad-hoc network. Every component creates a leader oriented directed acyclic graph (DAG).

In [12], 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 [12] 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 leader election problem on every network whereas knowing only an upper bound on the size is not enough.

Most of the previous work on the problem of leader election in mobile ad-hoc networks are based on defining restrictive assumption on the connectivity or the topology of the network. In this thesis, however, we assume no limitations on the number or the pattern of the link failures

[4]A directed acyclic graph is a finite directed graph with no directed cycles.

[5]The Temporally Ordered Routing Algorithm is an algorithm for routing data across Wireless Mesh Networks or Mobile ad hoc networks developed by Vincent Park and Scott Corson.

among the nodes of the system. Moreover, in the design of the leader election algorithm, we assume that the participating nodes have no initial knowledge regarding the set of participants in the system. As mentioned in 5.1, the consensus problem in which the participants do not know a priori who the other participants are is first noted in [6] and is called *consensus with uncertain participants*(CUP).

6.2 The VTL Leader Election Protocol

As mentioned before, in Chapter 3, Ferreira et al. in [25] propose a VTL scheme which relies on two main procedures: *leader election* and *leader handover*. The leader is a road vehicle that is elected by the vehicles approaching an intersection using a *leader election* protocol. The elected VTL leader assumes the role of traffic controller for a period 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.

In Chapter 3 we showed an example of how a VTL for a 4-leg intersection is established among the cars (See Fig. 3.1). Based on the VTL scheme explained in Chapter 3, each leg of the intersection consists of a cluster of cars for each of which, there exists a cluster leader. It is assumed that only the cluster leaders participate in a VTL leader election protocol.

Fig. 6.1 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 system can be created.

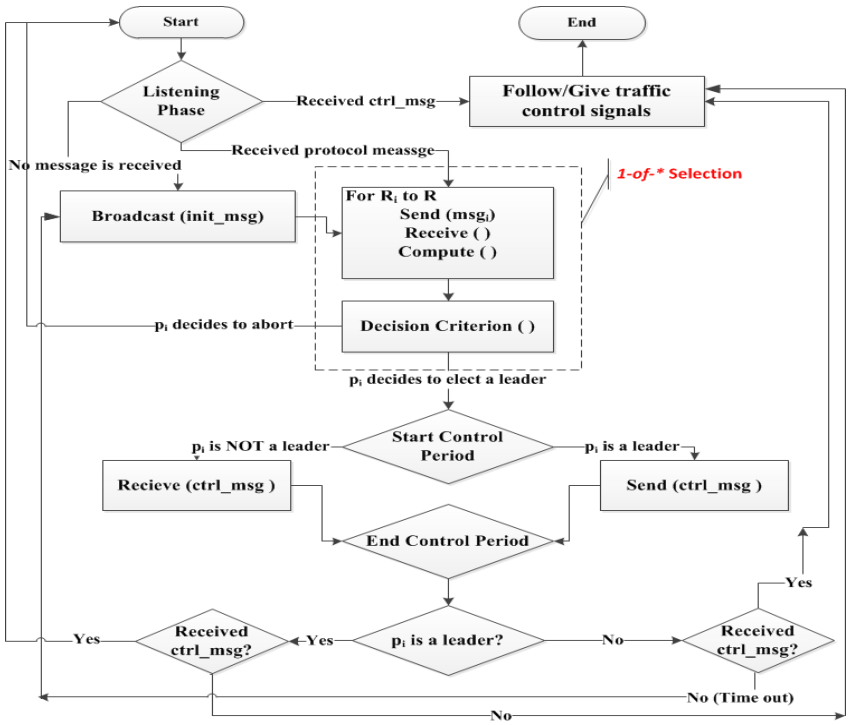


Figure 6.1: *Leader Election Protocol*

All vehicles are assumed to be equipped with the necessary technologies for data dissemination in a vehicular network in order to participate in a VTL system, e.g. the features for VANET geographical routing protocols such as beaconing and location table. (See Chapter 3)

As we see from Fig. 6.1, each vehicle (or process[6]) 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

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

solve the problem of *1-of-** selection or (ii) it follows the traffic light orders that it receives from an existing VTL leader. After the execution of the *1-of-** selection algorithm, each process 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 *1-of-** 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 *1-of-** 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 algorithm 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 another vehicle which is sent to call for starting the execution of a *1-of-** selection algorithm. A process which receives an initiating message during its listening phase will start executing the *1-of-** selection algorithm from round 1.

[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 *1-of-** selection algorithm.

This process is called an *initiator*. We assume that an initiator process will continue executing the protocol regardless of receiving acknowledgement messages indicating that they also participate in the algorithm 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]:** The processes which participate in executing a *1-of-** selection algorithm must reach an agreement on a leader among themselves. Each process in each round executes the three steps of *send*, *receive* and *compute*. We explain the details of each step in Section 6.3. 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 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.

≪**Control Period**≫ 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 the other processes. 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[7]. A non-aborting process which has not selected

[7]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. However, if we consider that a process can receive traffic signals at some point but never receives green light after a considerable

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 (which indicates that there are multiple leaders in the system), for safety reasons, 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 *1-of-** selection algorithm as the core logic of the leader election protocol.

6.3 Protocol Description

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 either through a global positioning system (GPS) [22] or relying on the existing clock synchronization methods

amount of time, it must abort and start the protocol from the beginning, i.e. listening phase.

for autonomous distributed applications such as in [29].

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.

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.

In order to account for the unreliability of the oracles, we consider a *correction parameter* (denoted by c). In this work, 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.

In the following section, we describe three different *1-of-** selection algorithms called the *optimistic**, the *pessimistic** and the *moderately pessimistic** selection algorithm.

6.3.1 The *1-of-** Selection Algorithms

Alg. 11 shows the pseudocode of a generic algorithm for the proposed *1-of-** selection algorithms, i.e. the *optimistic**, the *pessimistic** and the *moderately pessimistic** selection algorithms. Each participating process executes the *1-of-** 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*[8]) 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 11 Generic algorithm for *1-of-** Selection, p_i

```
1: msg_i ← {proposed_i, Π_i};
2: for r = 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. Fi-

[8]The ranking value depends on a number of parameters, such as the cluster leader's 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 thesis. In this work, for simplicity, we assign the identity of a process its ranking value.

nally 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 [24], 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 [24], 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 *1-of-** 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 $incomplete_r$.

Algorithm 12 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 ← proposed_j;
4:     end if
5:     Π_i ← Π_i ⋃ Π_j;
6:     msg_i ← {proposed_i, Π_i};
7: end for
```

Alg. 12 shows the psuedocode of the *compute* phase for the *optimistic** selection algorithm. Based on Alg. 12, 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. 13 shows the decision algorithm for the *optimistic** algorithm

Algorithm 13 Decision_Algorithm() for p_i: *Optimistic**

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

which is run by each process at the end of round R. According to Alg. 13, each process 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. 13 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 14 Compute (msg_i) for p_i: *Pessimistic**

```
1: for all p_j such that p_i has received msg_j do
2:     if r ≠ R then
3:         if proposed_i < proposed_j then
4:             proposed_i ← proposed_j;
5:         end if
6:         Π_i ← Π_i ⋃ Π_j;
7:     end if
8:     if m_j ≥ c * o_i then  //  Π_j is complete_r
9:         C_i[j] ← 1;
10:    end if
11: end for
12: msg_i ← {proposed_i, Π_i};
```

Alg. 14 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 local bit-vector called the *confirmation vector* which is denoted by C_i for process p_i. The confirmation vector,

Algorithm 15 Decision_Algorithm() for p_i: *Pessimistic**

```
1: if Π_i is complete_r then
2:     if C_i is complete_r then
3:         p_i selects proposed_i;
4:     else
5:         abort;
6:     end if
7: else
8:     abort;
9: end if
```

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. 15 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 16 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 ≠ R then
3:         Π_i ← Π_i ⋃ Π_j;
4:         if proposed_i < proposed_j then
5:             proposed_i ← proposed_j;
6:         end if
7:     end if
8: end for
9: msg_i ← {proposed_i, Π_i};
```

Alg. 16 shows the description of the compute phase for the *moderately*

Algorithm 17 Decision_Algorithm() for p_i: *Moderately Pessimistic**

```
1: if Π_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
```

*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. 17 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. 17). If p_i at round R, receives a message from a process p_j indicating that v_j is *relatively incomplete*, 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 *1-of-** 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.

6.4 Probabilistic Analysis

In this section, we present several graphs showing the outcome probabilities of the *1-of-** selection algorithms described in the previous section. We consider four types of outcomes: agreement on the identity of a process (`AG`), agreement on abort (`AB`), safe disagreement (`SD`) and unsafe disagreement (`UD`).

We show how the outcome probabilities varies for three system parameters: i) the values reported by the local oracles, ii) the value of the correction parameter (c), iii) the number of participating processes (n), and vi) the number rounds of message exchange.

We consider the same system model and failure model as in the previous chapters of this thesis. That is, we assume a synchronous system where the number of rounds, R, is fixed at design time, and unreliable links where messages are lost at the receiver side (receive omissions) with a probability of Q. We assume that Q is constant under the execution of one instance of the algorithm.

As in the previous chapters, we calculate the probability of each outcome of the algorithms using PRISM [36]. Due to the problem of state space explosion for large systems modelled in PRISM [16], we are only able to perform exact verification 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. In order to keep the uniformity of the results for different systems and system settings, in the following, we compute all results using simulations. In the next subsection, we briefly describe the PRISM model used for the analysis of the *1-of-** selection algorithms. Subsection 6.4.2 describes the formal specifications of the properties that PRISM uses to calculate the results. Subsection 6.4.3 presents results that illustrate the impact of variations in the correctness of the oracle values, while Subsection 6.4.3 presents results that show the impact of the correctness of the correction parameter. Subsection 6.4.3 and 6.4.3

present results illustrating the impact of the number of rounds, R, and the system size, n, respectively.

6.4.1 PRISM model

In this section, we present a PRISM model of a system with three processes executing a *1-of-** selection algorithm. The model is generic since it can be used for algorithms with different decision criteria. A PRISM model consists of a set of modules where each module represents a process. The communication failures are assumed to result in asymmetric message losses among the processes. So, a message sent from a process may be received or lost by a receiving process independently from other processes. This means that the number of states and transitions of a process's module depends on the number of participating processes in the system.

Fig. 6.2 presents a conceptual model of a process executing the algorithm in a system of three processes. Each node in the model represents a state of the process that is explained in Table. 6.1. The arrows in Fig. 6.2 show te transitions between the states. Finally in Table. 6.2 the states and transitions from Fig. 6.2 are described in more details. We assume there are three processes in the system called: p_i, p_j and p_k. Fig. 6.2 shows the states and transitions for process p_i; the same diagram can be used for other processes.

Starting from the initial state ($\bar{s}$), process p_i moves to the first state (s_1) to start executing the *1-of-** selection algorithm. Then, p_i moves from state s_1 to s_2 by transition T_1 and broadcasts its message to the network. This transition represents the *send* phase of the algorithm (See Alg. 11).

In the sequence, T_2 and T_3 are the probabilistic transitions which model the choice between receiving (with the probability of $1 - Q$) or losing (with the probability of Q) the messages sent from other two processes, i.e. p_j and p_k. T_2 and T_3 correspond to the *receive* phase of the *1-of-** selection algorithm.

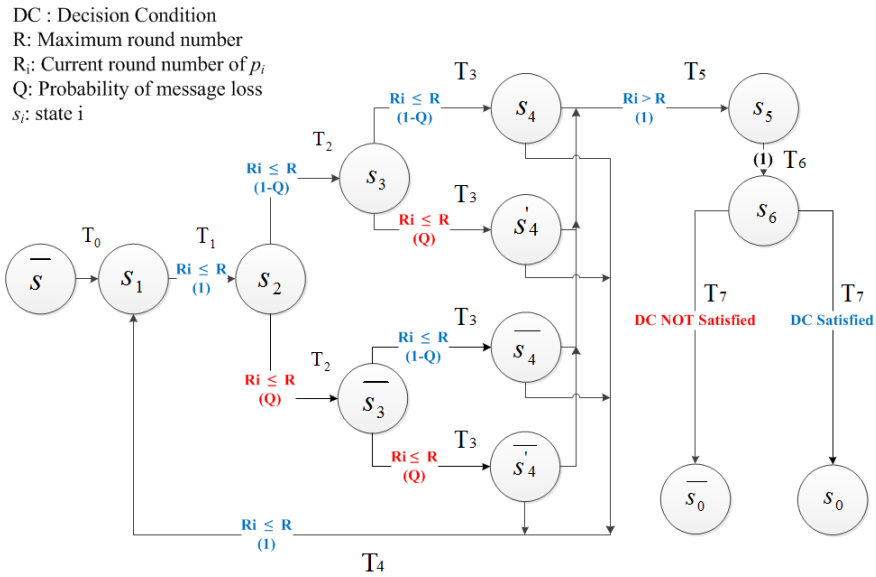


Figure 6.2: *The conceptual model of a process module p_i executing the 1-of-* selection algorithms for a system of three processes.*

Table 6.1: *The states of the PRISM module of process p_i in a system of three processes.*

State	Description
s_1	p_i sends message broadcast
s_2	p_i receives or loses the message sent by p_j
s_3	p_i receives the message sent by p_j
$\bar{s}_3$	p_i fails to receive the message sent by p_j
s_4	p_i receives the message sent by p_j and p_k
$s\prime_4$	p_i fails to receive the message sent by p_k but receives from p_j
$\bar{s}_4$	p_i receives the message sent by p_k but not from p_j
$\bar{s\prime}_4$	p_i does not receive messages neither from p_j nor from p_k
s_5	p_i computes the messages it received in round R
s_6	p_i computes the size of its view and confirmation set
s_0	p_i decides to select p_i
$\bar{s}_0$	p_i decides to abort

Table 6.2: *Transition descriptions of the PRISM module of a process p_i in a system of three processes executing the 1-of-* selection algorithm.*

Transition	From	To	Prob.	Description
T_0	$\bar{s}$	s_1	1	p_i moves from its initial state to state s_1
T_1: $R_i < R$	s_1	s_2	1	p_i broadcasts its message
T_2: $R_i \leq R$	s_2	s_3	$1-Q$	p_i receives the message sent by p_j
T_2: $R_i \leq R$	s_2	$\bar{s_3}$	Q	p_i does NOT receive the message sent by p_j
T_3: $R_i \leq R$	s_3	s_4	$1-Q$	p_i receives the messages sent by p_k and p_j
T_3: $R_i \leq R$	s_3	s'_4	Q	p_i does NOT receive the message sent by p_k but received from p_j
T_3: $R_i \leq R$	$\bar{s_3}$	$\bar{s_4}$	$1-Q$	p_i receives message sent by p_k but does not receive from p_j
T_3: $R_i \leq R$	$\bar{s_3}$	$\bar{s'_4}$	Q	p_i does NOT receive messages neither from p_k nor from p_j
T_4: $R_i \leq R$	s_4, s'_4, $\bar{s_4}$, $\bar{s'_4}$	s_1	1	Not the last round: p_i computes the round and increase R_i by 1. It moves to s_1 to continues to send receive and compute messages.
T_5: $R_i > R$	s_4, s'_4, $\bar{s_4}$, $\bar{s'_4}$	s_5	1	After the last round p_i computes the messages it received in round R
T_6	s_5	s_6	1	After the last round: p_i computes the size of its view and confirmation set.
T_7: decision condition satisfied	s_6	s_0	1	p_i decides to elect p_i
T_7: decision condition Not satisfied	s_6	$\bar{s_0}$	1	p_i decides to abort

At the end of all rounds except for the last one, the *compute* phase is performed by transition T_4. Moreover, by transition T_4, the round number is increased by one, i.e. $R_i = R_i + 1$.

At the end of the last round, T_5 fires instead of T_4 and performs the compute phase. With transition T_6, process p_i computes the size of its view vector and performs other necessary computations depending on the specified decision criterion. By transition T_7, process p_i makes a decision and moves to the final state, s_0 or $\bar{s_0}$ depending on the outcome of the decision process. State s_0 represents the state in which p_i decides to *select a leader* while state $\bar{s_0}$ corresponds to the state of p_i at which it decides to *abort*.

6.4.2 Specification of properties

In order to analyse the probabilistic PRISM model of the *1-of-** selection algorithm, it is necessary to identify the properties of the system model which can be evaluated by the tool. We specify each outcome of the *1-of-** selection algorithm as properties. We calculate the probability of each property by simulation using PRISM. We use PCTL temporal logics 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.

$$\forall p_i \quad (p_i \in S : p_i \; selects \; p_l \; , \; p_l \in S) \tag{6.1}$$

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

$$\forall p_i \;\; (p_i \in S : p_i \; decides \; to \; abort) \tag{6.2}$$

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.

$$\begin{aligned} &\exists p_i \;\; (p_i \in S^* \;\; : \;\; p_i \; selects \; p_x \;\; , \;\; p_x \in S) \;\; \wedge \\ &\quad \exists p_j \;\; (p_j \in S - S^* \;\; : \;\; p_j \; decides \; to \; abort) \end{aligned} \tag{6.3}$$

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.

$$\begin{aligned} &\exists p_i \;\; (p_i \in S^* \;\; : \;\; p_i \; selects \; p_l \;\; , \;\; p_l \in S \;\; , \;\; p_l \;\; is \; live) \;\; \wedge \\ &\qquad \forall p_j \;\; (\;\; (p_j \in S - S^* \;\; : \;\; p_j \; decides \; to \; abort) \;\; \vee \\ &\qquad (p_j \; selects \; p_x \;\; , \;\; p_x \in S \;\; , \;\; p_x \; decides \; to \; abort) \;\;) \end{aligned} \tag{6.4}$$

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.

$$\begin{aligned} &\exists p_i, \exists p_j, \exists p_x, \exists p_y \;\; (p_i, p_j, p_x, p_y \in S \;\wedge\; p_i \neq p_j \;\wedge\; p_x \neq p_y \;\; : \\ &(p_i \; selects \; p_x \;\; , \;\; p_x \;\; is \; live) \;\; \wedge \;\; (p_j \; selects \; p_y \;\; , \;\; p_y \; is \; live) \end{aligned} \tag{6.5}$$

In the following section, we present our results from the analysis of the *1-of-** selection algorithms based on the specified properties. We

calculate the probability of the occurrence of each property using the PRISM model checker.

6.4.3 Results

In the following, we present some of our interesting results from the probabilistic analysis of the *1-of-** selection algorithm.

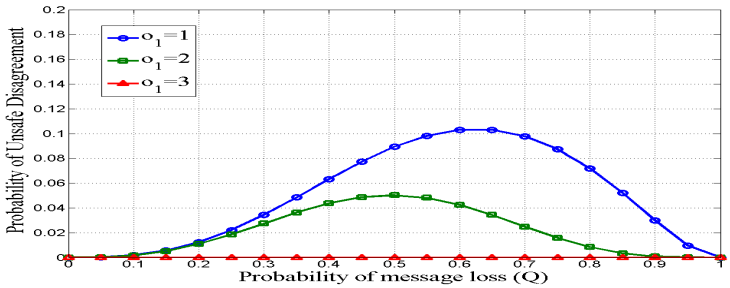
Varying the oracle values We present the results for systems where the local oracles of the participating processes are incorrect. We show how the probabilities of each outcome varies for different combinations of oracle values, for fixed values of R and c.

In the following, we only consider the cases where the incorrect oracles underestimate the the number of processes in the system and not overestimate it. If we consider the oracle values larger than the actual number of participants (i.e. $\forall i, o_i \geq n$), assuming $c = 1$ and knowing that $m_i \leq n$, the decision condition $m_i \geq c * o_i$ never holds for any process. Therefore, the processes always decide to abort. Moreover, in a VTL system, it is unrealistic to assume that the oracles (which are cameras and sensors in practice) are reporting the existence of some cars which are not actually present in the intersection.

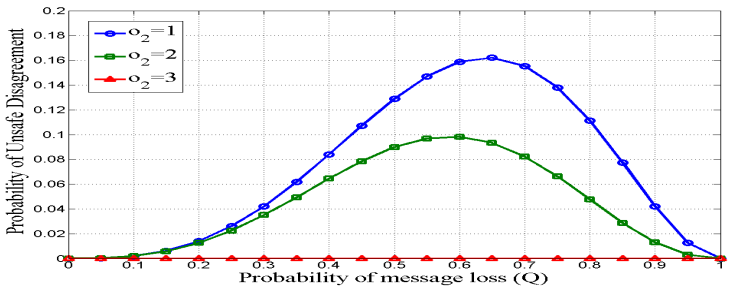
Optimistic* Fig. 6.3 shows the probability of unsafe disagreement for a system of three processes running the *optimistic** algorithm. The given results are for the cases where the oracle values of two of the processes are correct. We vary the oracle value of the third process.

As mentioned before, the decision function of the algorithm is set to the maximum function, i.e., the process with the highest ranking value is to be elected as the leader. The ranking values of each vehicle is set to its identity. Therefore, in a system of three processes, i.e. $\{p_1, p_2, p_3\}$, process p_3 has the highest ranking value and is to be elected as the leader by all processes.

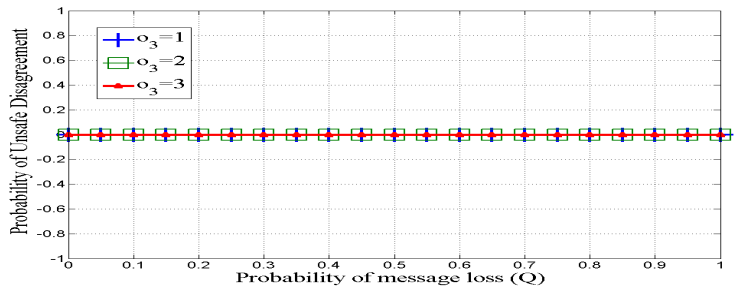
Fig. 6.3(a) shows the results for three combinations of the oracle values



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



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



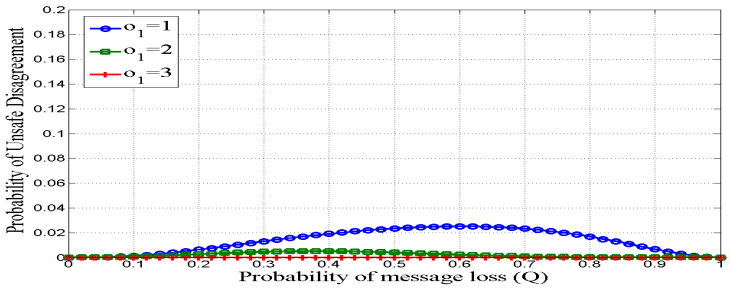
(c) $o_1 = 3$, $o_2 = 3$

Figure 6.3: *A comparison of the probability of unsafe disagreement for the 1-of-* selection algorithm using* ***optimistic**** *decision criterion for a system of three processes as a function of Q varying the oracle values. $R = 2$ and $c = 1$).*

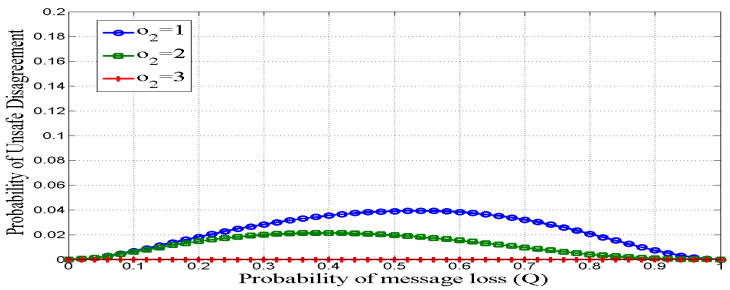
where process p_2 and p_3 always show correct local oracles and the oracle value of process p_1 assumes the values of 1, 2 and 3. As we see, the closer the value of o_1 becomes to the correct value (i.e. 3), we have lower probabilities of unsafe disagreement. For $o_1 = 3$ since all oracles for all processes show correct values, we have zero probabilities of `UD` . We see similar trend in Fig. 6.3(b), for the cases where we vary the oracle value of process p_2. Although, we have higher probabilities of `UD` for the cases where the oracle value of process p_2 is incorrect compared to the cases where the oracle value of process p_1 is incorrect. This is because of the maximum decision function and the ranking values of process p_1 and p_2. Process p_2 has a higher ranking value than p_1. Therefore, in cases where the view of p_2 is $\{p_1, p_2\}$ or $\{p_2\}$, p_2 selects itself as the leader while p_1 and p_3 with correct oracles select p_3 and as a result we have unsafe disagreement. On the other hand, for the same case in Fig. 6.3(a) where o_1 is incorrect and o_2 is correct, p_2 with the views of $\{p_1, p_2\}$ or $\{p_2\}$ decides to abort.

As we see in Fig. 6.3(c), for all values of o_3 and any probability of message loss (Q), the probability of unsafe disagreement is always zero. This is because p_1 and p_2 with correct oracles and using the *optimistic** decision criterion will decide on selecting a leader only if they have the complete view of the system. This means that p_1 and p_2 will only decide on a value 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 since process p_3 has the highest ranking value, p_1 and p_2 will select process p_3 as their leader if they satisfy the decision condition.

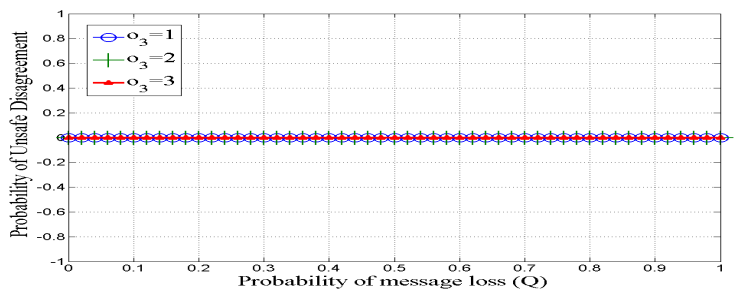
On the other hand, since process p_3 always includes its own identity in its view set which is the highest ranking value in the system, regardless of what information p_3 obtains from the system or which value its oracle provides, if it satisfies the decision condition (i.e. $m_3 \geq c * o_3$), it will always select itself (p_3) as the leader. Therefore, there is no such a case that two or more processes decide of different leaders, i.e. there is no case of unsafe disagreement.



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



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



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

Figure 6.4: *A comparison of the probability of unsafe disagreement for the 1-of-* selection algorithm using* ***moderately pessimistic**** *decision criterion as a function of Q for different settings of the oracle values and* $n = 3$, $R = 2$ *and* $c = 1$.

Moderately Pessimistic* Fig. 6.4 shows the results for the same setting as given in Fig. 6.3 but using the *moderately pessimistic** decision criterion. We see similar trend, but lower UD probabilities in general, as the one for the *optimistic** approach. However, the *moderately pessimistic** approach shows lower probabilities of unsafe disagreement compared to the *optimistic** approach for the same setting of the system.

Pessimistic* Fig. 6.5 shows the results for the same setting as given in Fig. 6.3 and 6.4 using the *pessimistic** decision criterion where there is only one process with an incorrect oracle value. As we see from the results for the *pessimistic** approach, the probability of UD is zero for all values of Q which means that all disagreement cases are safe. According to our analysis, for such a setting, the closer the value of an oracle becomes to the correct value, we have lower probabilities of safe disagreement.

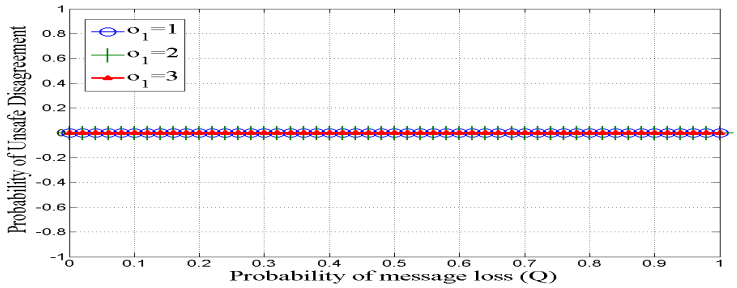
Our analysis show that in a system with the *pessimistic** decision criterion with any number of rounds, if there is only one process with an incorrect oracle value, the probability of unsafe disagreement is zero for all values of Q, i.e. all disagreement cases are safe. This is because of the decision condition specified for the *pessimistic** approach, 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. $m_j \geq c * o_i$)."

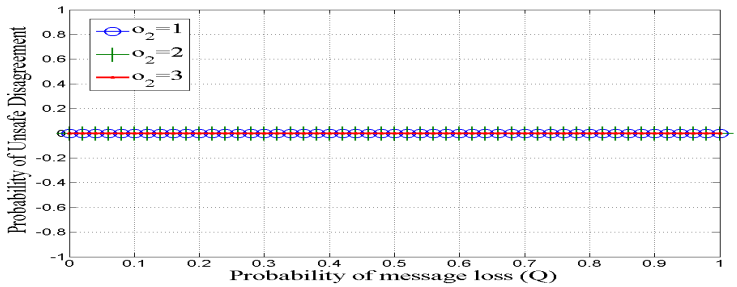
For example, consider the following setting of the system: In a system of three participating processes where the oracle values of process p_1, p_2 and p_3 are respectively $o_1 = 1$, $o_2 = 3$ and $o_3 = 3$.

Fig. 6.6 shows the message exchange among the processes at round 1 and round 2. As we see, at the first round, all messages are successfully delivered to their corresponding receivers except for the message sent by process p_3 to process p_1, illustrated in red dotted color. At the second round, two messages sent from p_2 and p_3 to process p_1 are lost.

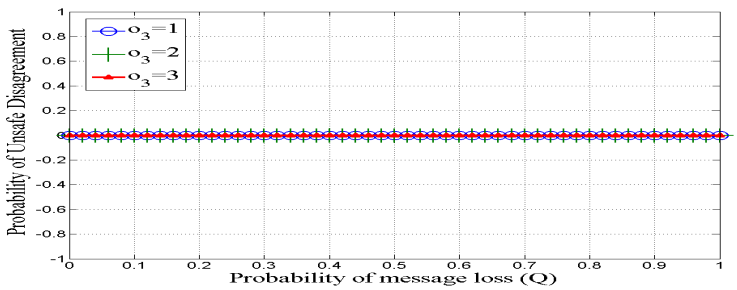
Table. 6.3 shows the three steps of the *1-of-** selection algorithm in



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



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



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

Figure 6.5: *A comparison of the probability of unsafe disagreement for the 1-of-* selection algorithm using* ***pessimistic**** *decision criterion as a function of Q for different settings of the oracle values and $n = 3$, $R = 2$ and $c = 1$.*

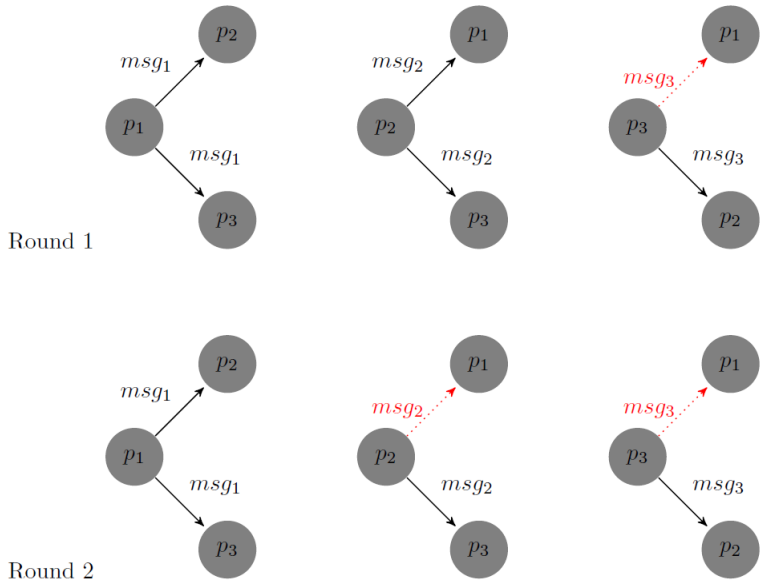


Figure 6.6: *The message exchange among three processes executing the 1-of-* selection algorithm for two rounds. The red dotted arrows indicate the lost messages.*

the first round according to the message exchanges given in Fig. 6.6. The message of a process p_i includes its proposed value and its view set, $[proposed_i, \Pi_i]$ (See Alg. 11). Table. 6.4 shows similar information for the second round. The last column shows the result of the compute phase for each process at the end of the last round.

After the execution of the *1-of-** selection algorithm, the view set of each process are as follows: $\Pi_1 = \{p_1, p_2\}$, $\Pi_2 = \{p_1, p_2, p_3\}$ and $\Pi_3 = \{p_1, p_2, p_3\}$.

Process p_1 decides to select p_2 since it satisfies both decision conditions: $m_1 = 2$ and $2 \geq c * o_1 (= 1)$ and process p_2 has at least one process in its view as it is seen by process p_1, i.e. m_2 seen by p_1 is 1 and $m_2 \geq c * o_1 (= 1)$.

p_i	Send (msg_i)	Receive ()	Compute(msg_i, C_i)
p_1	$[p_1, \{p_1\}]$	$[p_2, \{p_2\}]$	$[p_2, \{p_1, p_2\}]$, $C_2 = \{p_2\}$
p_2	$[p_2, \{p_2\}]$	$[p_1, \{p_1\}]$, $[p_3, \{p_3\}]$	$[p_3, \{p_1, p_2, p_3\}]$, $C_1 = \{p_1\}$, $C_3 = \{p_3\}$
p_3	$[p_3, \{p_3\}]$	$[p_1, \{p_1\}]$, $[p_2, \{p_2\}]$	$[p_3, \{p_1, p_2, p_3\}]$, $C_1 = \{p_1\}$, $C_2 = \{p_2\}$

Table 6.3: *Three steps of the 1-of-* selection algorithm using* ***pessimistic**** *decision criterion ($R = 2, c = 1, o_1 = 1, o_2 = 3, o_3 = 3$),* ***Round 1***

p_i	Send(msg_i)	Receive ( )	Compute(msg_i,C_i)
p_1	$[p_2, \{p_1, p_2\}]$	NULL	$[p_2, \{p_1, p_2\}]$, $C_2 = \{p_2\}$
p_2	$[p_3, \{p_1, p_2, p_3\}]$	$[p_2, \{p_1, p_2\}]$, $[p_3, \{p_1, p_2, p_3\}]$	$[p_3, \{p_1, p_2, p_3\}]$, $C_1 = \{p_1, p_2\}$, $C_3 = \{p_1, p_2, p_3\}$
p_3	$[p_3, \{p_1, p_2, p_3\}]$	$[p_2, \{p_1, p_2\}]$, $[p_3, \{p_1, p_2, p_3\}]$	$[p_3, \{p_1, p_2, p_3\}]$, $C_1 = \{p_1, p_2\}$, $C_2 = \{p_1, p_2, p_3\}$

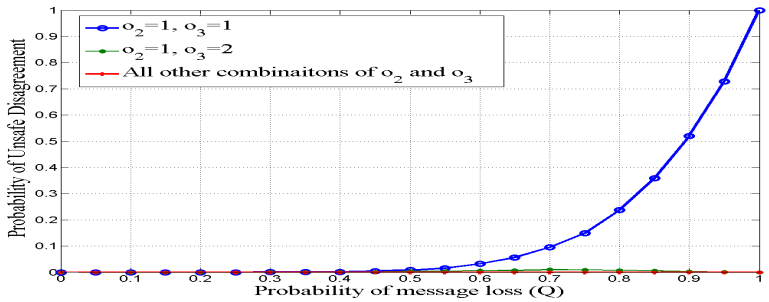
Table 6.4: *Three steps of the 1-of-* selection algorithm using* ***pessimistic**** *decision criterion ($R = 2, c = 1, o_1 = 1, o_2 = 3, o_3 = 3$),* ***Round 2***

.

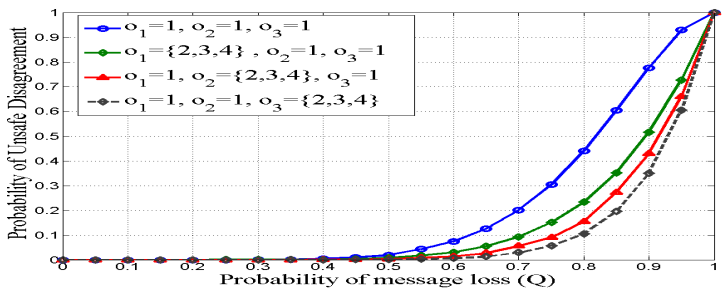
On the other hand, process p_2 and p_3 decide to abort because process p_1 in their view does not satisfy the second decision condition for the *pessimistic** approach: $m_1(1) \not\geq c * o_2(= 3)$ and $m_1(1) \not\geq c * o_3(= 3)$. In other words, process p_2 and process p_3 pessimistically assume that process p_1 will not satisfy the first condition (i.e. it has a correct oracle) and will abort. So, only process p_1 will select a leader (i.e. p_2) and others will abort and therefore, we have safe disagreement among the processes.

Fig. 6.7 shows how the probability of unsafe disagreement varies for a system of four processes running the *pessimistic** algorithm where two and three processes have incorrect oracle values respectively shown in Fig. 6.7(a) and Fig. 6.7(b).

As we see from both graphs given in Fig. 6.7, we have the highest probabilities of UD when all oracles show the value of 1. We can show that for all decision criteria we have up to 100% of probabilities of UD when



(a) $o_1 = 4$, $o_4 = 4$, Varying o_2 and o_3

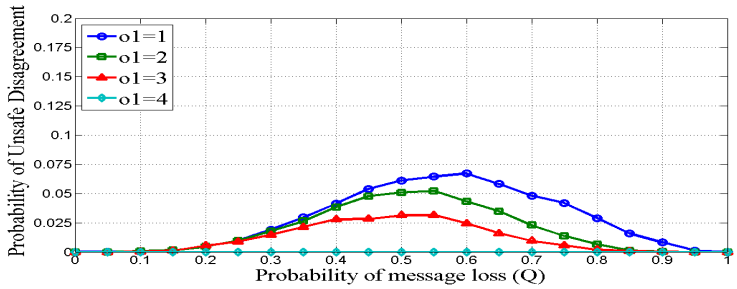


(b) $o_4 = 4$, Varying o_1, o_2 and o_3

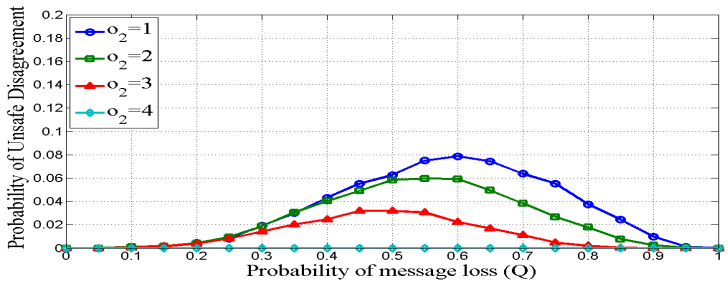
Figure 6.7: *A comparison of the probability of unsafe disagreement for the 1-of-* selection algorithm using* ***pessimistic**** *decision criterion as a function of Q for different settings of incorrect oracle values and* $n = 4$, $R = 2$ *and* $c = 1$.

the oracle values of all processes show the value of 1, i.e. $o_i = 1$. This is because of the fact that a process with the oracle value of 1 always satisfies the decision conditions given in Alg. 15 and therefore decides to select a leader. Based on the assumption, the number of processes in the view of all processes in the system is at least one, i.e. a process always sees itself in its own view set, i.e. $m_i \geq 1$. So, considering $c = 1$ and $o_i = 1$, process p_i always satisfies the decision conditions $m_i \geq c * o_i$ and $m_j \geq c * o_i$.

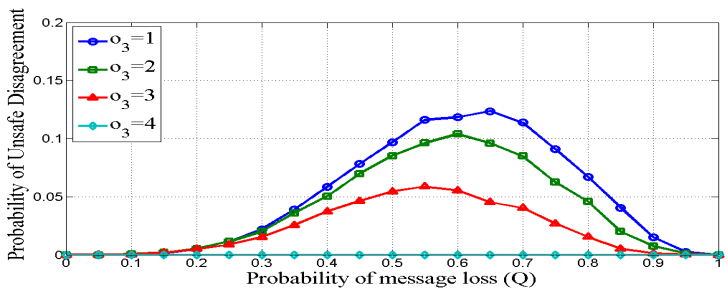
As we see in Fig. 6.7(b), we have the highest probabilities of `UD` when the oracle values of o_1, o_2 and o_3 are the minimum possible value, i.e. 1 (assuming that $o_4 = 4$). The next highest `UD` probabilities correspond to the settings of the oracles in which two of the oracles show the value of 1. We can show that for all other combinations of the oracle values where $o_4 = 4$, the probability of `UD` is always zero or very close to zero.



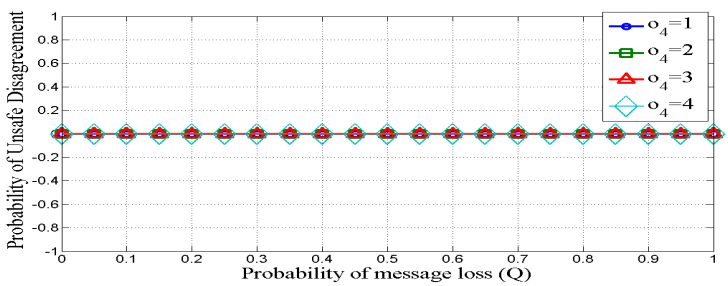
(a) $o_2 = 4$, $o_3 = 4$, $o_4 = 4$



(b) $o_1 = 4$, $o_3 = 4$, $o_4 = 4$



(c) $o_1 = 4$, $o_2 = 4$, $o_4 = 4$



(d) $o_1 = 4$, $o_2 = 4$, $o_3 = 4$

Figure 6.8: *The probability of unsafe disagreement for the 1-of-* selection algorithm using* ***optimistic**** *decision criterion as a function of Q for different combinations of the oracle values for a system of four participants ($R = 2$ and $c = 1$).*

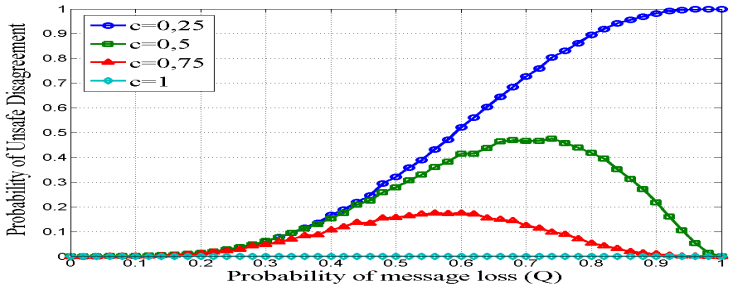
Fig. 6.8 shows the probability of unsafe disagreement for a system of four processes running the *1-of-** selection algorithm using *optimistic** decision criterion in two rounds of execution. In each of the graphs given in Fig. 6.8, three of the processes have correct oracle values.

Fig. 6.8(a) shows the results for four different combinations of the oracles values where o_1 can be $1, 2, 3$ or 4 and the remaining processes have correct values (i.e. $o_2 = o_3 = o_4 = 4$). As we see from the results, in all graphs, the closer the value of the o_1 becomes to the correct value we have lower probabilities of `UD` . Fig. 6.8(b) and 6.8(c) show similar results for different combinations of the oracle values of process p_2 and p_3 respectively. However, the results for different values of o_3 (Fig. 6.8(c)) shows higher probabilities of `UD` compared to the ones for o_2 and even higher compared to the results for various o_1. Comparing the results given in Fig. 6.8 and Fig. 6.3, we see similar trend. In all cases when the oracles values of all processes are correct, we have zero probabilities of `UD` . Similar to the results for a system of three processes, when we vary the oracle values of the process with the highest ranking value (which is p_4), while the remaining oracle values are correct, we have zero probabilities of `UD` for any Q (See Fig. 6.8(d)).

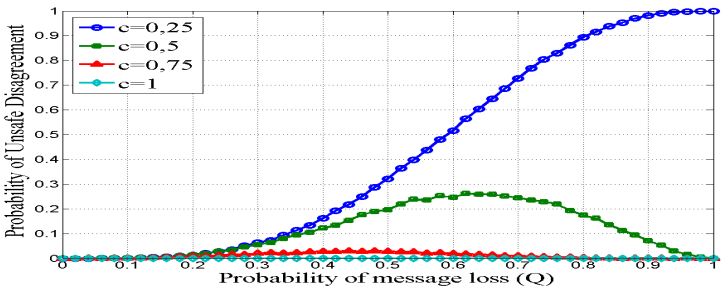
Varying the correction parameter (c) In this section, we show how different values of the *correction* parameter (i.e. c) affects the behaviour of the *1-of-** selection algorithms. For this, we assume that the local oracles of all processes show the correct number of the participants while we vary the value of c. We assume the same value of c for all participating processes.

As mentioned before, using the correction parameter a process intends to compensate for the unreliability of its local oracle. A process assumes the value of one for its correction parameter if it believes that its oracle is correct, i.e. it counts the value provided by its oracle as the actual number of processes in the system. Assuming $c < 1$, a process assumes that its oracle overestimates the number of processes in the system. Finally,

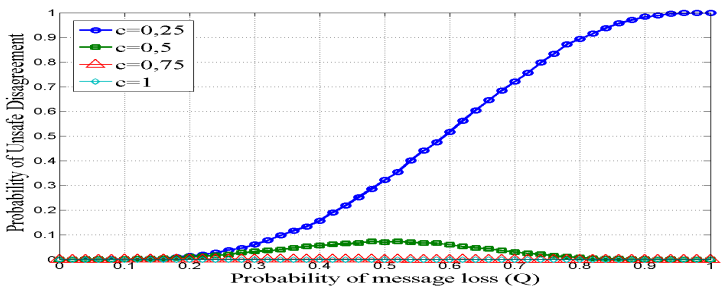
a process assumes $c > 1$ if it believes that its oracle underestimates the number of processes. As the decision conditions for the processes executing the *1-of-** selection algorithm relies on the factor of $c * o_i$, in order to see the effect of varying the value of c, we set the values of the oracles to the correct value, i.e. $o_i = n$.



(a) Optimistic*



(b) Moderately pessimistic*



(c) Pessimistic*

Figure 6.9: *The probability of unsafe disagreement for the 1-of-* selection algorithm using three decision criteria as a function of Q for different values of c for a system of four participants ($R = 2$ and $o_i = 4$)*

Fig. 6.9 shows the probabilities of unsafe disagreement for a system of four processes running the *1-of-** selection algorithm for two rounds using different decision criteria. As we see from the results given in Fig. 6.9, for all decision criteria, for larger values of c the probability of UD decreases in general. In all graphs, for $c = 1$ the probability of UD is zero for all values of Q, which is expected as all oracle values are assumed to be correct.

The *optimistic** approach shows the highest probability of UD compared to other decision criteria for $c = 0.25, 0.5$ and $= 0.75$. For $c = 0.5$ the probability of UD can go up to around 0.47 for $Q = 0.72$ using the *optimistic** approach. However, the *moderately pessimistic** and *pessimistic** approach show the highest probabilities of UD for smaller values of Q, respectively 0.26 and 0.072 for $Q = 0.68$ and $Q = 0.52$. For $c = 0.75$, the probability of UD for the *optimistic** approach can peak to around 0.20 while it is zero or close to zero for other decision criteria.

According to the results in Fig. 6.9, for all decision criteria, considering $c = 0.25$, the probability of UD can increase up to 1. This is because of the decision conditions specified for each decision algorithm. In all cases, if all messages among the processes are lost due to communication failures (i.e. $Q = 1$), the view of all processes consists of only itself, i.e. $\Pi_i = \{p_i\}$ and therefore $m_i = 1$. Using the *optimistic** decision criterion, assuming $o_i = 4$ and $c = 0.25$ the decision condition $m_i \geq c * o_i$ always satisfies for all processes. So, all processes always decide to select a leader. In the extreme case, when all messages are lost, each process will decide to select itself as the leader and as a result we have 100% probability of UD .

In case of using the *moderately pessimistic** approach, according to the decision condition specified in Alg. 17, a process p_i will decide to select a leader if it satisfies two conditions: (1) at the end of round R we have $m_i \geq c * o_i$ and (2) all messages p_i receives during the last round from other processes as p_j indicate that their sender's view is relatively complete, i.e. $m_j \geq c * o_i$. This means that all lost messages during the

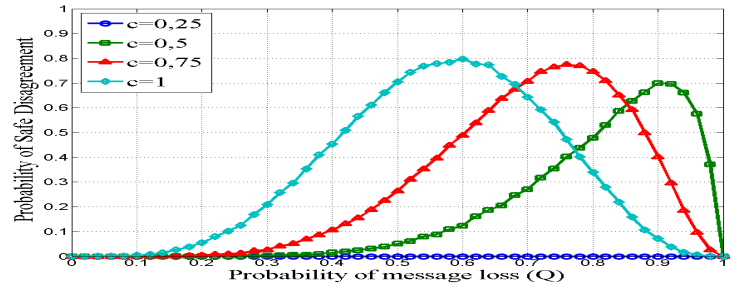
last round are optimistically assumed to be relatively complete. In the extreme case of $Q = 1$ (i.e. when all messages are lost), condition (1) holds for all processes for the same reason as for the *optimistic** approach. For condition (2), obviously as all messages are lost, process p_i will not receive a message from any process indicating the incompleteness of the view of its sender.

The results in Fig. 6.9(c) for the *pessimistic** approach show similar trend as the results for other decision criteria. According to Alg. 15, a process p_i will decide to select a leader if it satisfies two conditions at the end of round R: (1) we have $m_i \geq c * o_i$ and (2) all processes in Π_i have a relatively complete view of the system, i.e. their view is complete according to the oracle value of p_i.

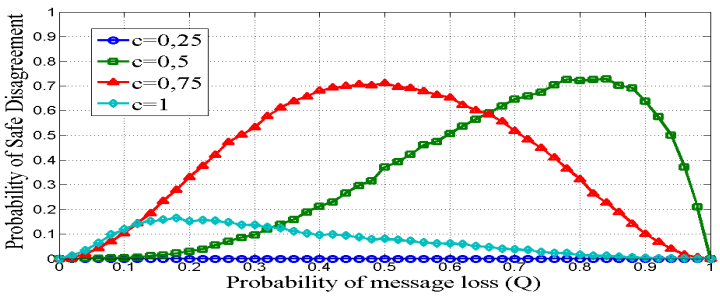
As we see from the results in Fig. 6.9, using the *pessimistic** approach we have the lowest probabilities of `UD` in general compared to other decision criteria except for the case where $c = 0.25$. This is because of the stricter decision criterion for a process specified for the *pessimistic** approach compared to other approaches which results in higher number of abort cases compared to the other decision criteria (See Fig. 6.11).

In the given results we have not considered the values of c which are larger than one. This is because with assuming correct oracles, if $c > 1$ all processes will always abort as they can never satisfy the condition $m_i \geq c * o_i$ even if their view is complete (i.e. even if $m_i = n$).

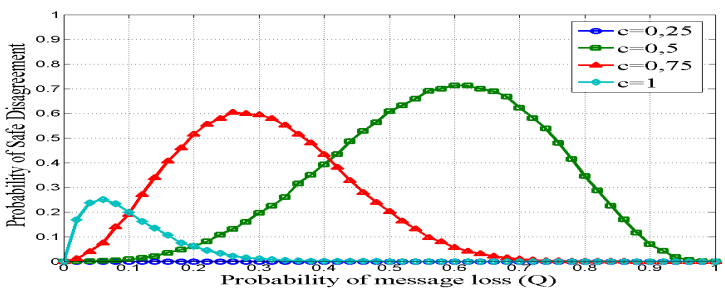
In the following figures, we show the results for the probability of other outcomes of the *1-of-** selection algorithm for the same settings of the system as given in Fig. 6.9. We present the probabilities of `SD` , `AB` and `AG` respectively in Fig. 6.10, 6.11 and 6.12.



(a) Optimistic*



(b) Moderately pessimistic*



(c) Pessimistic*

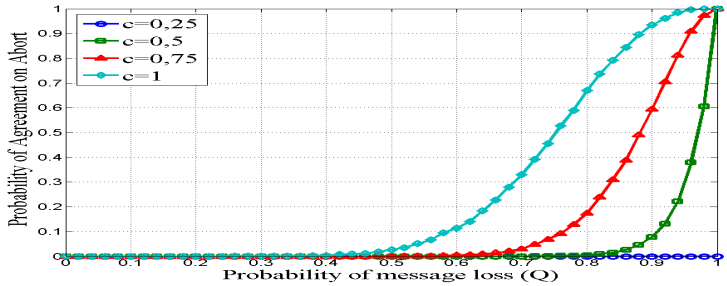
Figure 6.10: *The probability of safe disagreement for the 1-of-* selection algorithm using three decision criteria varying c as a function of Q for a system of four participants ($R = 2$ and $o_i = 4$)*

Fig. 6.10 shows the probability of safe disagreement (i.e. SD) for a system of four processes with correct oracles executing the *1-of-** selection algorithm for two rounds of execution varying the value of c for different decision criteria.

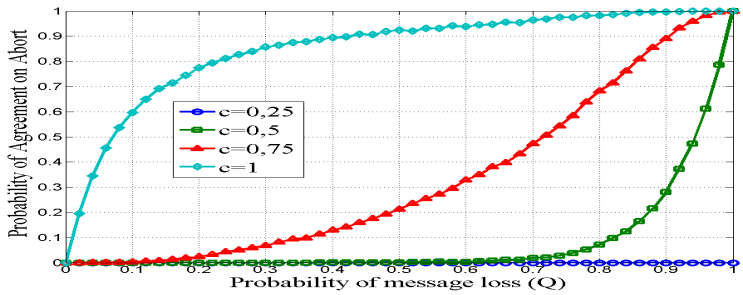
As we see from Fig. 6.10(a), for the *optimistic** approach, with increasing the value of c we have higher probabilities of safe disagreement. Moreover, with increasing c, the probability of SD curve moves to the left side of the x-axis, which means that the maximum value of the probability of SD occurs at lower probabilities message loss (Q) for larger c.

For $c = 1$, the *optimistic** approach shows the highest probabilities of SD compared to other decision criteria. The *moderately pessimistic** approach shows the lowest peak of the probability of SD for $c = 1$. For $c = 0.75$, the *optimistic** approach shows the highest peak of the probability of SD which occurs at the largest value of Q compared to other decision criteria. For $c = 0.5$, all decision criteria show almost the same maximum probability of SD which is $\approx 70\%$ which occurs at $Q = 0.9$ for the *optimistic** approach, and at $Q = 0.8$ and $Q = 0.6$ respectively for *moderately pessimistic** and *pessimistic** approach. For $c = 0.25$ we have zero probabilities of SD for all decision criteria. This is because with $c = 0.25$ and correct oracle values, no process will decide to abort and therefore all disagreement cases are unsafe.

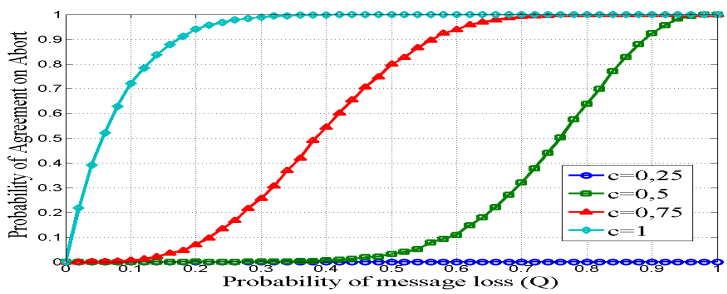
Assuming correct oracles, setting the value of c to a value smaller than one means that the processes consider their oracles as being overestimating the actual number of processes in the system. So, considering the *optimistic** decision condition for process p_i (i.e. $m_i \geq c * o_i$), with small values of c and correct values of oracles (i.e. $o_i = n$ in a system of n participants), it is more probable that p_i satisfies the decision condition and decide on its view. So, for the *optimistic** approach, the higher the value of c is it is more probable that a process makes correct (or safe) decisions since its view must include more processes in order to be able to decide.



(a) Optimistic*



(b) Moderately pessimistic*



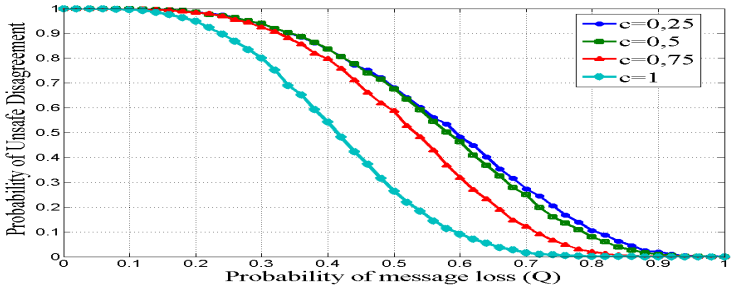
(c) Pessimistic*

Figure 6.11: *The probability of agreement on abort for the 1-of-* selection algorithm using three decision criteria varying c as a function of Q for a system of four participants ($R = 2$ and $o_i = 4$)*

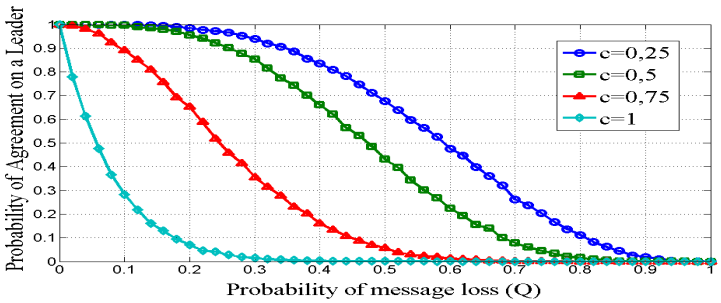
Fig. 6.11 shows how the probability of agreement on abort changes for the *1-of-** selection algorithm using three decision criteria varying c. The results are given as a function of Q for a system of four participating processes with correct oracles (i.e. $o_i = 4$), executing the algorithm in two rounds.

As we see from the results, for all decision criteria, for larger values of c we have higher probabilities of agreement on abort. This is because for a larger value of c, a process p_i must have higher number of processes in its view set to satisfy the first decision condition, i.e. $m_i \geq c * o_i$. Thus, with higher probabilities of message loss (i.e. Q) and larger values of c, it is more probable that a process decides to abort.

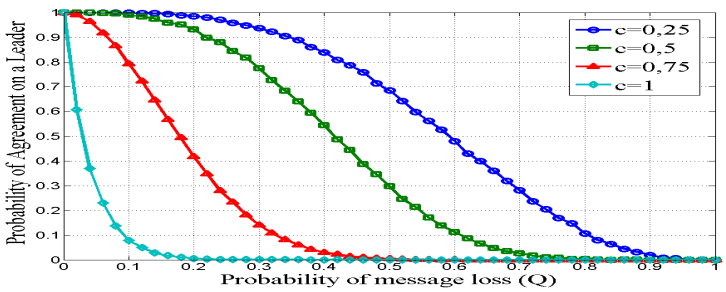
From the results given in Fig. 6.11, we see that the *optimistic** approach has the lowest probabilities of `AB` for smaller values of Q and the *pessimistic** one has the highest `AB` probabilities for the same system settings.



(a) Optimistic*



(b) Moderately pessimistic*



(c) Pessimistic*

Figure 6.12: *The probability of agreement on abort for the 1-of-* selection algorithm using three decision criteria varying c as a function of Q for a system of four participants ($R = 2$ and $o_i = 4$)*

Fig. 6.12 shows how the probability of agreement on a leader changes for the *1-of-** selection algorithm using three decision criteria varying c. The results are given as a function of Q for a system of four processes with correct oracles and $R = 2$. As we see from the results for all decision criteria for larger values of c, we have lower probabilities of agreement on a leader (`AG`). This is because with larger values of c it is less probable for a process to satisfy the decision condition (the processes are more likely decide to abort due to lack of information).

From the given results in Fig. 6.12, we can also see that the *pessimistic** approach shows the lowest probabilities of `AG` for the same settings of the system compared to other decision criteria, except for $c = 0.25$. All decision criteria show almost the same probabilities of `AG` for $c = 0.25$ which are also the highest probabilities of `AG` . Moreover the *optimistic** approach shows similar results for $c = 0.5$ as for $c = 0.25$. The *optimistic** and the *pessimistic** decision criteria show similar results when $c = 0.5$ for the *pessimistic** criterion and $c = 1$ for the optimistic decision criterion.

Fig. 6.13 and Fig. 6.14 show the probability of `UD` respectively, for a system of three and four processes executing the algorithm using three decision criteria 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.

As we see from Fig. 6.13, for all decision criteria, the probability of `UD` declines considerably first at $c = 0.33$ and then at $c = 0.66$. For a system of four processes, according to the given results in Fig. 6.14, for all decision criteria, the probability of `UD` shows sudden decreases at $c = 0.25$, $c = 0.50$ and $c = 0.75$. We can show that 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[9].

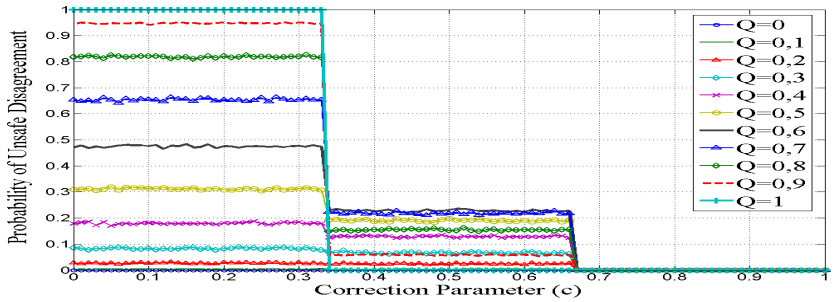
[9]Step function (or staircase function) is a function on the real numbers which can be written as a finite linear combination of indicator functions of intervals.

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 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.

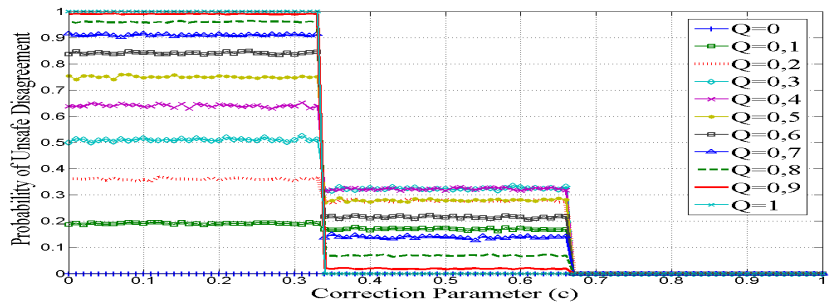
From the given results in 6.13, we also see that the probabilities of UD for different values of Q does not keep the same trend in different ranges of the value of c. For example, in Fig 6.13(a), for $0 \leq c \leq 0.25$ the probabilities of UD increase for larger values of Q while for the interval $0.26 \leq c \leq 0.51$ 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**

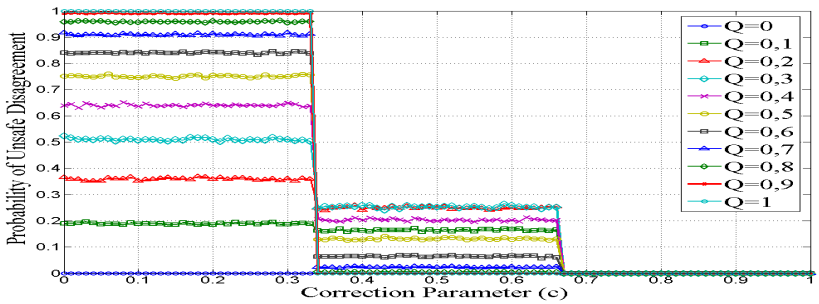
approach shows a lower probabilities of `UD` compared to the *moderately pessimistic** approach.



(a) Optimistic*

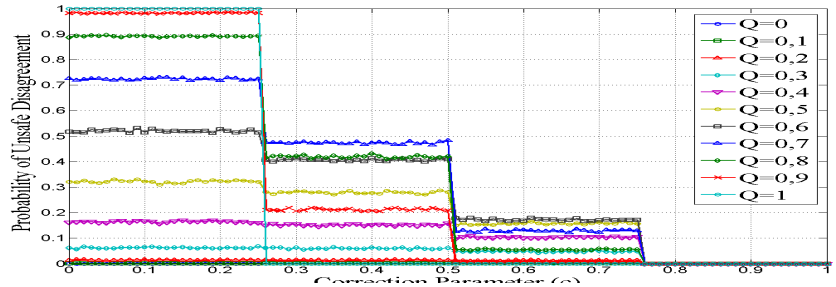


(b) Moderately pessimistic*

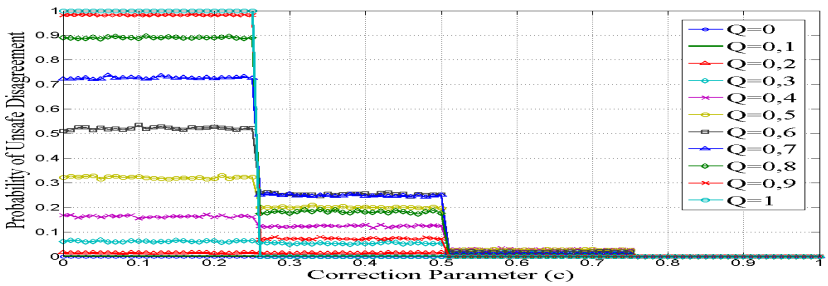


(c) Pessimistic*

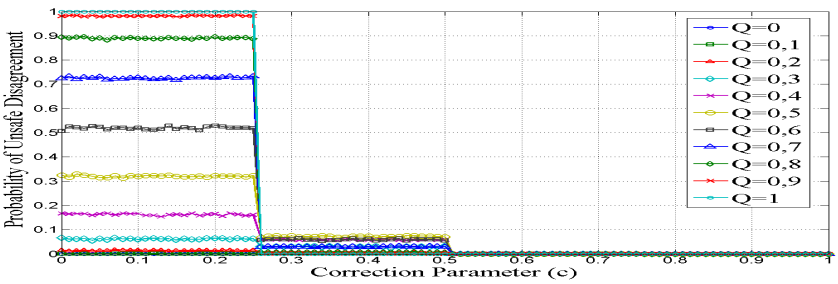
Figure 6.13: *The probability of unsafe disagreement for the 1-of-* selection algorithm using three decision criteria varying c as a function of Q for a system of three participants ($R = 2$ and $o_i = 3$)*



(a) Optimistic*

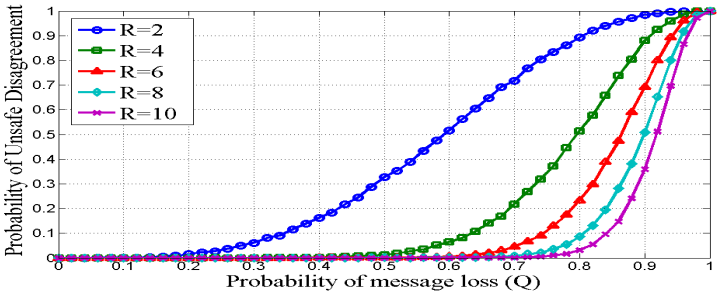


(b) Moderately pessimistic*

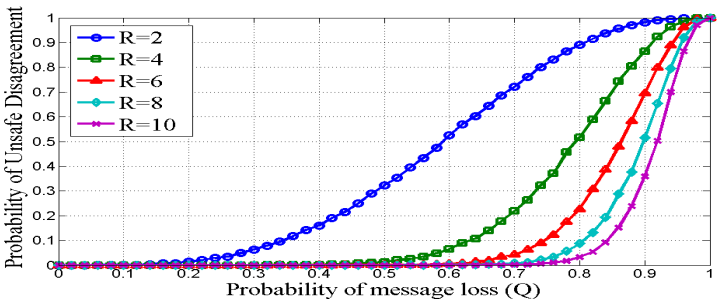


(c) Pessimistic*

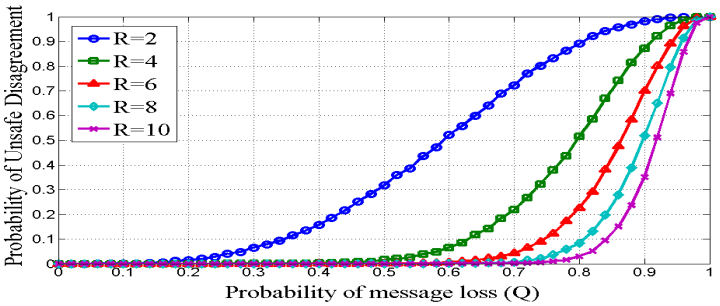
Figure 6.14: *The probability of unsafe disagreement for the 1-of-* selection algorithm using three decision criteria varying c as a function of Q for a system of four participants ($R = 2$ and $o_i = 4$).*



(a) Optimistic*

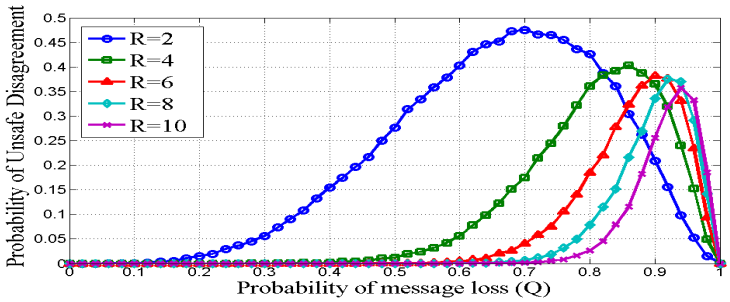


(b) Moderately pessimistic*

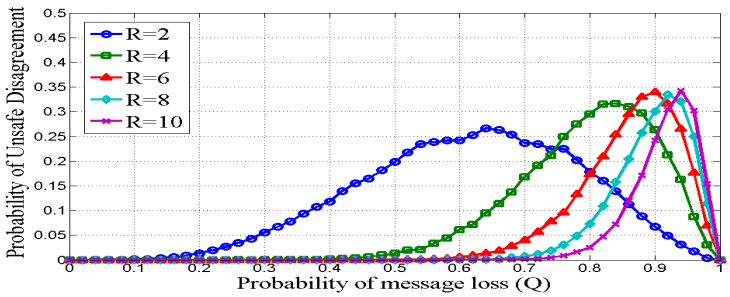


(c) Pessimistic*

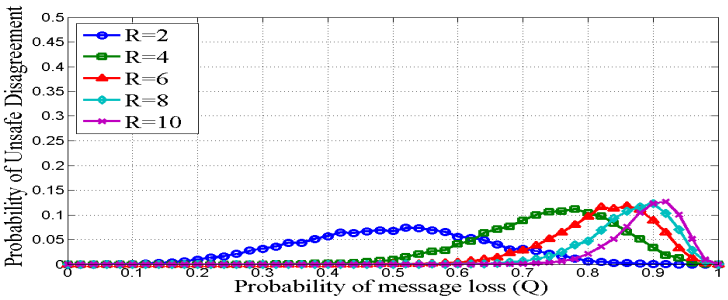
Figure 6.15: *A comparison of the probability of unsafe disagreement for the 1-of-* selection algorithm for three decision criteria as a function of Q for a system of four participants (Varying R, $c = 0.25$ and $o_i = 4$).*



(a) Optimistic*



(b) Moderately pessimistic*



(c) Pessimistic*

Figure 6.16: *A comparison of the probability of unsafe disagreement for the 1-of-* selection algorithm for three decision criteria as a function of Q for a system of four participants (Varying R, $c = 0.5$ and $o_i = 4$).*

Varying the number of rounds (R) Fig. 6.15 shows how the probability of UD varies for the *1-of-** selection algorithm using different decision criteria in a system of four processes with correct oracles and $c = 0.25$. We consider different number of rounds of message exchange for the algorithm as follows: $R = 2, 4, 6, 8, 10$.

As we see from the results in Fig. 6.15, for all decision criteria when we increase the value of R, the probability of UD decreases in the general. On the other hand, for all decision criteria if we assume correct oracle values but $c = 0.25$, we have the same probabilities of UD for a system setting of $n = 4, R = 2, 4, 6, 8, 10$. This is because with assuming $c = 0.25$, the initial decision condition which is $m_i \geq c * o_i$ always holds for all processes with correct oracles. Therefore, using the *optimistic** approach, all processes always decide to select a leader regardless of their view of the system.

For the same reason, in case of using the *pessimistic** and *moderately pessimistic** decision criteria, all decision conditions hold and as a result all processes always decide to select a leader regardless of the number of processes they see in their view.

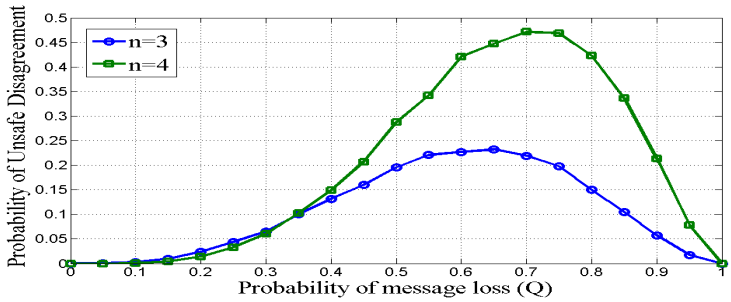
Fig. 6.16 shows a comparison of the probability of UD for a system of four participants with correct oracles and the correction parameter of $c = 0.5$ executing the *1-of-** selection algorithm for different number of rounds using different decision criteria. As we see from the results, increasing the number of rounds, does not show the same influence on the probability of UD for all decision criteria. For the *optimistic** decision criterion, increasing R results in lower UD probabilities in general.

Moreover, with increasing the value of R, the highest probabilities of UD occur at larger values of Q. For the *pessimistic** and *moderately pessimistic** decision criteria, increasing the number of rounds we have higher peaks of UD probabilities but occurring in larger values of Q.

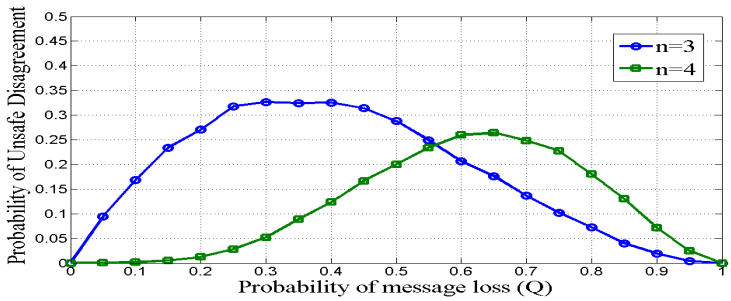
Varying the number of participants Fig. 6.17 shows a comparison of the probability of unsafe disagreement for two systems of three and

four participating processes executing the *1-of-** selection algorithm for two rounds. It is assumed that the oracle values are correct and the correction parameter is set to $c = 0.5$. As we see from the results, for the *optimistic** approach, the probability of `UD` is higher for $n = 4$ compared to $n = 3$, while this is not the same for the *pessimistic** and *moderately pessimistic** decision criteria.

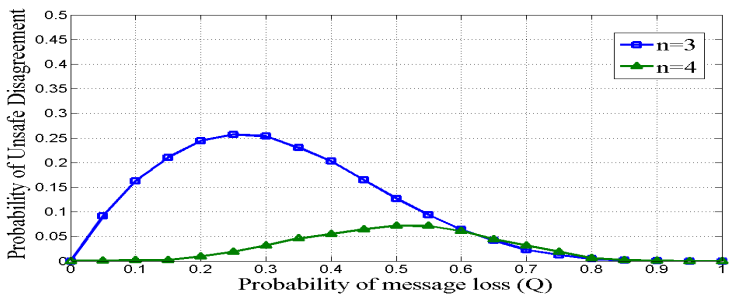
Fig. 6.18 shows the results for the probability of safe disagreement for the same settings as given in Fig. 6.17. As we see from the results, Similar to `UD` probabilities, the curves for the `SD` probabilities move to the right side of the x-axis for the system with four participants compared to the system with three participants. However, for the *pessimistic** and *moderately pessimistic** decision criterion, with increasing the number of participants, we have higher probabilities of `SD` for larger values of Q which is not the same for the *optimistic** approach.



(a) Optimistic*

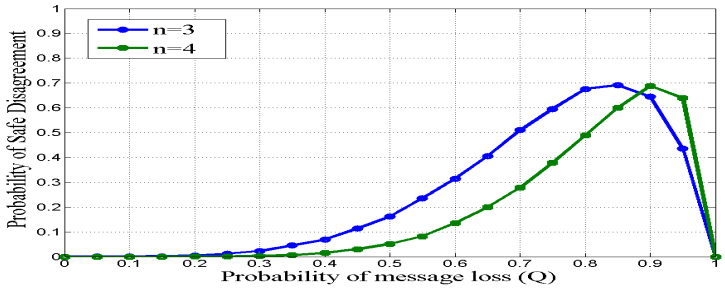


(b) Moderately pessimistic*

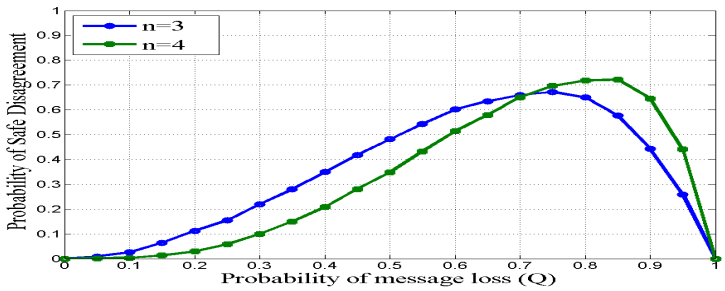


(c) Pessimistic*

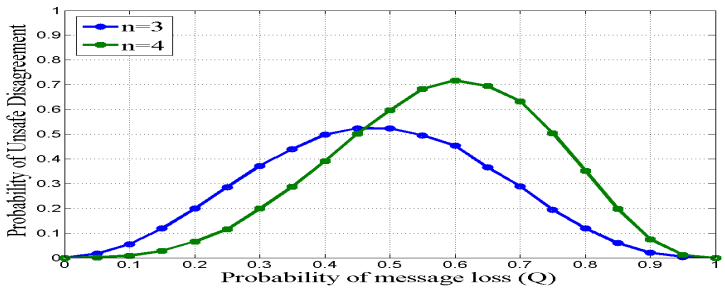
Figure 6.17: *A comparison of the probability of unsafe disagreement for the 1-of-* selection algorithm for three decision criteria as a function of* Q *(*$R = 2$*, Varying* n*,* $c = 0.5$ *and* $o_i = n$*).*



(a) optimistic*



(b) Moderately pessimistic*



(c) pessimistic*

Figure 6.18: *A comparison of the probability of safe disagreement for the 1-of-* selection algorithm for three decision criteria as a function of Q ($R = 2$, Varying n, $c = 0.5$ and $o_i = n$).*

6.5 Discussions

Our main observations from the probabilistic analysis of the *1-of-** selection algorithms are as follows:

- 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. 6.8). This observation is helpful in introducing the policies for defining ranking values for 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 define a relatively large ranking value for itself. Based on our observations, this vehicle will most probably select itself as the VTL leader and therefore will order the red light to its own lane first. Such a policy is in contrary to the one proposed in [25] where the closest vehicle to the intersection must be elected as the leader.

- We have the highest probabilities of `UD` when the local oracles of the majority of the processes show the minimum value of 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 conditions $m_i \geq c * o_i$ and $m_j \geq c * o_i$ regardless of the number of processes in its view (since $c * o_i \leq 1$ and $m \geq 1$). As a result, we have the highest possibility of having unsafe disagreement among the processes.

  Therefore, there is a need to define 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 on its identity as the leader as it is under the current design of the algorithm. In other words, a process p_i with an oracle value of $o_i = 1$, only decides to select a leader if its view contains only its own identity, i.e. p_i selects itself as the leader. Such an approach prevents the problem of starvation of a single vehicle in an intersection waiting for a green light.

- The *pessimistic** decision algorithm shows the lowest probabilities of `UD` compared to other decision criteria. This is interesting as in our previous works where we assumed that n is known, the *moderately pessimistic** approach showed the best results, i.e. the lowest probabilities of disagreement.

- For all decision criteria if we assume correct oracle values but $c = 0.25$, we have the same probabilities of `UD` for a system setting of $n = 4, R = 2, 4, 6, 8, 10$. For larger values of the correction parameter ($c \geq 0.25$), with increasing the number of rounds, we do not see the same effect on the probability of `UD` for all decision criteria. For the *optimistic** decision criterion, with increasing R, we have lower probabilities of `UD` in general, while for the *pessimistic** and the *moderately pessimistic** decision criteria we have higher maximum values of the `UD` probabilities but occurring in larger values of Q. This means that for systems with massive communication failure, increasing the number of rounds of execution of the *1-of-** algorithms can result in larger probabilities of unsafe disagreement.

6.6 Chapter Conclusions

In this chapter, we designed and analysed a family of distributed algorithms called *1-of-** selection algorithms to solve the problem of leader election among the processes in a self-organizing system with unrestricted

omission failures. In the proposed algorithms, processes are augmented with a local oracle that estimates, with a given confidence, the number of participants in the leader election protocol. We introduced the *1-of-** selection algorithms to be used as the core logic of a VTL leader election protocol.

Our analyses results confirmed that under our system model and failure assumptions, it is impossible to guarantee agreement [52]. Moreover, results show that when the number of nodes are unknown and the communication failures are unrestricted, *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 decision algorithm.

We introduced three different decision algorithms for the *1-of-** selection algorithms called the *optimistic**, *pessimistic** and *moderately pessimistic** decision algorithms. Our results show that the choice of a decision algorithm significantly influences the probability of each outcome of the algorithm. However, we cannot claim that one decision algorithm is better than another one since the outcome of the algorithm also depends on other system parameters. The probability of each outcome of the algorithm also depends on the quality of the network, i.e. the probability of a message being lost in the system.

We introduced three main outcomes for the *1-of-** selection algorithms: (i) *agreement on a leader*, (ii) *agreement on abort* and (iii) *disagreement*.

We defined two types of processes; *aborting* processes and *non-aborting* processes. We call a process an *aborting* process if after the execution of a *1-of-** selection algorithm, it decides to abort. Likewise, we define a process *live* or *non-aborting* if it decides to select a leader. Based on the given definitions, we introduced a new classification of *disagreement* to *safe* and *unsafe* compared to the classification we introduced in Chapter 5 as follows:

Unsafe Disagreement We have unsafe disagreement if there are at least two processes deciding on 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[10] 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.

Our analysis show that we can reduce the probability of *unsafe disagreement* with choosing the right decision algorithm and proper configurations of the system parameters such as the number of rounds of execution or the correction parameters.

We analysed the behaviour of the *1-of-** selection algorithms for systems of three and four participating processes. The participating processes are assumed to be the cluster leaders of each lane of an intersection in a VTL scenario. 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 *1-of-** selection algorithm for systems with larger number of participants is beneficial in understanding the behaviour of the algorithm.

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

7 Discussion

In this thesis, we addressed the problem of designing synchronous consensus algorithms that minimizes the probability of disagreement in the presence of an unbounded number of messages losses. Understanding this problem is vital for assessing the risk of failures in distributed systems where consensus algorithms are used to ensure that the nodes in the system make coordinated and consistent decisions. We described an example of such systems in Chapter 3. The design of reliable protocols for such applications comprises a lot of challenges of which we only investigated a few. In the following, we discuss some solutions we propose to the problem of having multiple leaders and the problem of integrating the new comers into a safety-critical system.

7.1 The problem of multiple leaders

We know from previous research [25] that it is impossible to avoid disagreement among the participating processes in a consensus protocol in the presence of unrestricted communication failures. Although, we can reduce the probability of the occurrence of unsafe situations due to disagreement with proposing decision algorithms and with proper setting of the system parameters, we cannot guarantee that the probability of disagreement is always zero. This means that, in a VTL leader election protocol based on wireless ad-hoc networks, the probability of election of multiple leaders by the nodes cannot be zero in general, what we call *the problem of multiple leaders*.

One solution to the problem of multiple leaders in a VTL system is to enforce the elected leaders to use the same traffic light policy for an intersection. For example, a policy can be that the VTL leader must give the green light in an order at which the lanes with the larger traffic congestions (i.e. larger number of waiting cars) receive the green light first. This, of course relies on the assumption that the elected leaders have the same information about the level of traffic congestion in each lane.

An alternative traffic light policy is to use an off-line database of the information collected statistically of a certain intersection. For example, if the leaders in an intersection have access to an information that implies that certain lanes are more prone to be congested with cars than others, the VTL leaders should automatically give the green light priority to that lane.

7.2 The problem of new comers

An important challenge in the design of distributed algorithms based on vehicular ad-hoc networks is that the participating vehicles in an algorithm may dynamically join and leave the network. As a result, the

number of participants in the algorithm may vary over the time.

In the design of the *1-of-n* selection algorithm presented in Chapter 4, for simplicity, we assumed a *fixed* number of participating processes which are initially *known* to all processes in the system (i.e. n was known by all nodes). In Chapter 6, in the design of the *1-of-** selection algorithm, we relaxed the simplifying assumption of knowing n. We assumed a system with a *fixed* but *unknown* number of processes. In the design of the *1-of-** selection algorithm, we assumed that all processes start the algorithm at the same time and no process leaves the system until the end of the algorithm. Moreover, we assumed that no process can join an already running algorithm by some other processes. We know that such a simplifying assumption is unrealistic for applications based on ad-hoc networks.

Thus, another challenge in the design of cooperative applications is to address the agreement problem under a new assumption where it is possible to include new participating processes to an already running algorithm. Under the new assumption, a process can join a running algorithm at a later round provided that it satisfies certain conditions. We define these conditions with the aim of minimizing the possibility of having unsafe situations. In a VTL scenario, *an incoming process* implies a vehicle that is just arrived to an intersection where the *1-of-** selection algorithm is running to construct a VTL.

We assume that an incoming process p_x, starts with a *listening phase* in which it listens to the network to see whether a *1-of-** selection algorithm is already running or not. If p_x during its *listening phase* receives messages indicating that an algorithm is running, it must satisfy certain conditions to be able to join the running algorithm. We assume that the messages sent by the participating processes in the algorithm (called the *protocol message*) contain the current round number of the algorithm and are timestamped.

To this end, we propose two main approaches for an incoming process to join a running algorithm called the *greedy* approach and the *exhaus-*

tive approach. We specify a decision algorithm for an incoming process whether to decide to *join* the algorithm or to decide to *abort* and wait for another listening phase.

Our ultimate goal is to investigate the performance of our previously proposed consensus algorithms with respect to the probability of disagreement under the new assumptions and using the proposed approaches. In the following we explain each approach in more details using simple examples.

Greedy Approach

Using the *greedy* approach, as soon as an incoming process receives a protocol message, it stops its listening phase and moves to an idle state until the starting of the next round to join the algorithm.

Fig. 7.1 shows an example of the execution of the *greedy* approach. Assume that process p_1 has initiated an algorithm. During the first round of the algorithm, process p_2 tries to join the system starting with the *listening* phase. Process p_2 successfully receives a message from p_1 indicating that it is running an algorithm and its current round is 1 ($R_c = 1$).

Figure 7.1: *Process inclusion using greedy approach*

We assume that each protocol message contains a *timestamp* value which shows the starting time of the round at which the message is sent. Moreover, we assume that all processes have access to a global clock. So, by receiving a message from p_1, process p_2 can calculate the starting point

of the next round (in this case round $R = 2$). Note that, in the greedy approach, we assume a maximum period of listening phase. Such an assumption prevents a single process in a system to stay in the listening phase forever.

As mentioned before, the design of the *greedy* approach implies that the participating processes have access to a global clock and are able synchronize their rounds according to that clock. There are a number of methods proposed in literature for clock synchronization in autonomous distributed applications such as in [42]. In a vehicular network, as each vehicle has access to the global positioning system and clocks, we can assume that the vehicles are synchronized using the globally known time information acquired from the Global Positioning System (GPS) [22].

Exhaustive Approach

The main difference between the *greedy* and the *exhaustive* approach is that using the *exhaustive* approach a process should remain in its *listening phase* until the end of the specified period. As a result, using the *exhaustive* approach, a process in its *listening phase* may receive several protocol messages containing different round numbers from different processes. This can specifically happen if due to network partitioning there are several instances of the algorithm running at the same time.

Fig. 7.2 shows an example of the execution of the *exhaustive* approach which is used by an incoming process p_2 in order to join the algorithm running by process p_1. Assume that process p_1 has initiated a VTL algorithm. During the first round of the algorithm, process p_2 tries to join the system and starts with the *listening* phase. Process p_2 receives two messages from process p_1 during its *listening phase*, one with $R_c = 1$ and one with $R_c = 2$. Process p_2 moves to the idle state when it is finished with its *listening phase* and remains waiting until the start of the closet round (Round 2).

Based on our failure assumptions any number of messages can be lost during the execution of the algorithm. So, in the example given in

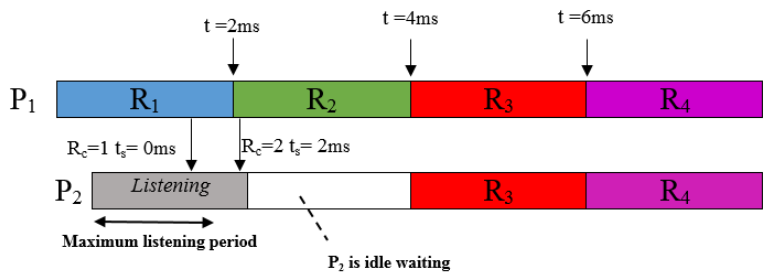


Figure 7.2: *Process inclusion using exhaustive approach, R_c: current round, t_s:timestamp*

Fig. 7.2, the second message sent from process p_1 might be lost. Nonetheless, since the processes are assumed to have the same clock, process p_2 can calculate the exact time at which it should start executing the algorithm (i.e. round 3) using the information it obtained from p_1 's message (i.e. $R_c = 1$ and $t_s = 0$). So, similar to the *greedy* approach, the processes using the *exhaustive* approach must have access to the same global clock.

As we see from the given example in Fig. 7.2, using the *exhaustive* approach, depending on the length of the *listening phase* and the time at which the incoming process receives protocol messages, its *idle* time[1] can vary. For example if process p_2 has started the *listening phase* a bit earlier (or if the *listening phase* was shorter), it could have started to join the algorithm at round 2 instead of round 3.

Fig. 7.3 shows an example of the inclusion of a process p_x where it receives two protocol messages from two processes who are running two different instances of the consensus algorithm. Process p_y and p_z are isolated from each other due to communication failures, i.e. they cannot communicate with each other. In such a case, p_x must decide to abort and possibly send acknowledgement messages to p_y and p_z to inform them that there are at least two instances of the algorithm running at

[1]The *idle* time refers to the time at which the process is actually waiting to join the algorithm.It can be shorter or longer than a *listening phase* period.

the same time. When p_y and p_z receive the information message from p_x they must decide to abort as well.

If process p_x uses the *greedy* approach it will not be able to detect the two instances of the algorithm since it stops its *listening phase* as soon as it receives the first protocol message.

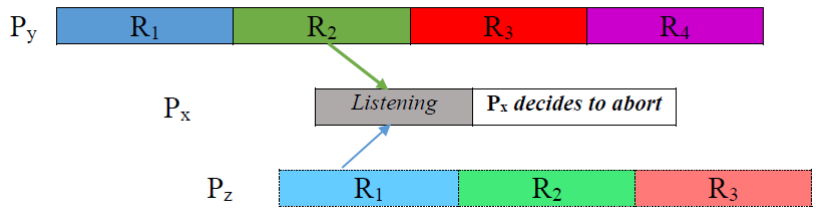


Figure 7.3: *Process inclusion using exhaustive approach. Process P_x receives two messages from two different executions of the algorithm.*

With the new system model assumptions the following research questions are raised:

- We assume that the number and the duration of the rounds of execution of the algorithm is fixed and set at the design time for all processes (i.e. R). Under this assumption when an *incoming process* receives a message containing the round number of a running algorithm it can calculate the number of remaining rounds and decide whether to join or not. Therefore, an open question is that what is the **latest** round at which an *incoming process* may join an algorithm in order to have the least probability of disagreement.

- Which of the given approaches, *greedy* or *exhaustive* approach, performs better in the process of including a new participant in a leader election algorithm?

- One of the main challenges in the design of leader election algorithms in an ad hoc network is to bootstrap the algorithm. In our

current design we assume that a process which has not detected a running algorithm in its *listening phase* can start up an algorithm by broadcasting initiating messages to the network. On the other hand, based on our communication failure assumptions, all messages sent to a process during its *listening phase* might be lost. Such a process initiates a new algorithm without being aware of the existence of one running algorithm in the system. Consequently we have several instances of the algorithm running asynchronously.

In the presence of several running algorithms in the system, an *incoming process* who wants to join the system may receive several protocol messages with different timestamps and round numbers. In the current suggested design of the leader election protocol, we assumed that such a process must decide to abort for safety reason. However, it is necessary to investigate the behaviour of the algorithm for two scenarios of the *incoming process* that is aborting or joining one of the running algorithms and inform the others to abort?

8 Conclusion

This thesis addresses the problem of reaching consensus in distributed systems that rely on wireless networks for data communication. Our work is mainly motivated by the increasing interest in developing cooperative automotive application to improve traffic safety and fuel efficiency, but it is also relevant for other types of cooperative applications.

We define and address three types of consensus problems that are relevant for cooperative systems. We call these problems *1-of-n selection*, *1-of-* selection* and *group formation*. The two first problems have the same goal - to reach consensus on a single value. In the *1-of-n selection* problem we assume that all nodes know the size of the systems at all times, while we assume that system size is initially unknown to all nodes in the *1-of-* selection* problem. The aim of the *group formation* problem

is to boot-strap a self-organizing system, i.e., to reach agreement on the identity of the nodes that form the system.

We propose and analyse several distributed agreement algorithms to provide probabilistic solutions to these problems. We address specifically the problem of reaching agreement in the presence of an arbitrary number of communication failures. Hence, since it is known that no algorithm can guarantee consensus in the presence of massive communication failures, our analyses focused on comparing and assessing the *probability of disagreement* for different algorithms.

For the *1-of-n* and the *1-of-** problems, we propose and investigate three variants of agreement algorithms. These variants are distinguished by the decision criterion the nodes use in order to determine whether they should decide on a value or decide to abort.

The fact that our algorithms allow processes to abort has two important implications. First, it introduces two major forms of agreement: i) *agreement on a value (or a group)* and ii) *agreement on abort*. Second, it also introduces two major forms of disagreement, which we call *safe* and *unsafe* disagreement. Safe disagreement corresponds to cases where all non-aborting processes decide to select the same value (or group), while at least one process decides to abort. Unsafe disagreement corresponds to cases where at least two non-aborting processes decide on different values (or groups).

To conclude, we choose to highlight the following important observations from our work:

- Our analyses show that the probability of disagreement depends strongly on the number of nodes in the system, the number of rounds of message exchange, the choice of decision criterion, as well as the probability of message loss.

- In general, it is not possible to rank different decision criteria, since their performance (probability of agreement) depends strongly on the probability of message loss. For example, in Chapter 4, we

show that the pessimistic decision criteria tends to have a high probability of disagreement at fairly low probabilities of message loss, while the optimistic one has a high probability of disagreement at fairly high probabilities of message loss. This suggests that the development of adaptive agreement algorithms, which alters their decision criterion based on an estimation of the current quality of the communication channel, would be an interesting topic for future research.

- We show that unsafe disagreement can be avoided if all nodes know the number of the participating nodes in a system. In the analysis of group formation algorithm in Chapter 5, we observe that unsafe disagreement only occurs when the local oracles underestimate the number of nodes in the system.

- An important parameter in the configuration of a system is the number of rounds of execution of a consensus protocol , i.e. R. Our observations implies that increasing the execution duration of our proposed consensus protocols does not necessarily show better results.

- We compared the behaviour of the proposed algorithms for different sizes of the systems. Based on our results we cannot claim that in general with larger number of participants we have higher probabilities of *disagreement*. We show that the effect of having larger values of n on the outcome of the algorithms also depends on the other system parameters.

Finally, we would like to comment on the limitation of our work. In analyzing complex systems, like the ones considered in this thesis, researchers and engineers are often forced to make simplifying assumptions to make the analysis tractable. This is also the case for the work presented in this thesis. One important limitation of our work is that we assume a synchronous model of execution. Assessing the validity of using

this model for reasoning about cooperative systems is a complex question, which we leave for future research.

Another obvious limitation is that we only consider send and receive omissions in our analyses. Since cooperative systems are safety-critical system, it is also necessary to investigate the behaviour of agreement algorithms in the presence of various types of node and communication failures, including malicious attacks. Yet another limitation is our assumption about the occurrence of message losses. In all of our analyses, we assume that the probability of message loss is constant during the execution of an algorithm. We also assume that the probability of message loss is the same for all messages. Clearly, both of these assumptions may not be valid for real systems.

In this thesis, we analysed the protocols for small systems of at most six nodes. Based on our main application example, i.e. the VTL system, only the cluster leaders of an intersection participate in the consensus protocol. The number of the cluster leaders is limited to the number of the lanes of the corresponding intersection which is usually less than six. However, as our proposed consensus protocols can be applied on any distributed application, it is important to analyse the behaviour of these algorithms for larger systems as part of our future work.

Bibliography

[1] Yehuda Afek, Hagit Attiya, Alan Fekete, Michael Fischer, Nancy Lynch, Yishay Mansour, Dai-Wei Wang, and Lenore Zuck. Reliable communication over unreliable channels. *J. ACM*, 41(6):1267–1297, November 1994.

[2] E. A. Akkoyunlu, K. Ekanadham, and R. V. Hubert. Some constraints and tradeoffs in the design of network communications. In *Proceedings of the fifth ACM Symposium on Operating Systems Principles*, pages 67–74. ACM, 1975.

[3] Eduardo A. Alchieri, Alysson Neves Bessani, Joni Silva Fraga, and Fabíola Greve. Byzantine consensus with unknown participants. In *Proceedings of the 12th International Conference on Principles of Distributed Systems*, OPODIS '08, pages 22–40, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] K. Alekeish and P. Ezhilchelvan. Consensus in sparse, mobile ad hoc networks. *Parallel and Distributed Systems, IEEE Transactions on*, 23(3):467–474, March 2012.

[5] Luciana Arantes, Fabíola Greve, Pierre Sens, and Véronique Simon. Eventual Leader Election in Evolving Mobile Networks. In Roberto Baldoni, Nicolas Nisse, and Maarten van Steen, editors, *OPODIS 2013 - 17th International Conference Principles of Distributed Systems*, volume 8304 of *Lecture Notes in Computer Science*, pages 23–37, Nice, France, December 2013. Springer.

[6] Ziv Bar-Joseph, Idit Keidar, and Nancy Lynch. Early-delivery dynamic atomic broadcast. In Dahlia Malkhi, editor, *Distributed Computing*, volume 2508 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2002.

[7] Michael Barborak, Anton Dahbura, and MiroslawProbabilistic verification Malek. The consensus problem in fault-tolerant computing. *ACM Comput. Surv.*, 25(2):171–220, June 1993.

[8] Carl Bergenhem, Steven Shladover, Erik Coelingh, Christoffer Englund, and Sadayuki Tsugawa. Overview of platooning systems. In *Proceedings of the 19th ITS World Congress, Oct 22-26, Vienna, Austria (2012)*, 2012.

[9] Martin Biely, Ulrich Schmid, and Bettina Weiss. Synchronous consensus under hybrid process and link failures. *Theoretical Computer Science*, 412(40):5602 – 5630, 2011.

[10] Christophe Bonnet and Hans Fritz. Fuel consumption reduction in a platoon: Experimental results with two electronically coupled trucks at close spacing. *Intelligent Vehicle Technology*, (SP-1558), 2000.

[11] David Cavin, Yoav Sasson, and AndrÃĺ Schiper. Consensus with unknown participants or fundamental self-organization. In *Ad-Hoc, Mobile, and Wireless Networks*, volume 3158 of *Lecture Notes in Computer Science*, pages 135–148. Springer Berlin Heidelberg, 2004.

[12] Jeremie Chalopin, Emmanuel Godard, and Antoine Naudin. What do we need to know to elect in networks with unknown participants? In MagnusM. Halldorsson, editor, *Structural Information and Communication Complexity*, volume 8576 of *Lecture Notes in Computer Science*, pages 279–294. Springer International Publishing, 2014.

[13] Bernadette Charron-Bost and AndrÃĺ Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22:49–71, 2009.

[14] Bernadette Charron-Bost and AndrÃĺ Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22:49–71, 2009.

[15] Gregory Chockler, Murat Demirbas, Seth Gilbert, Calvin Newport, and Tina Nolte. Consensus and collision detectors in wireless ad hoc networks. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Principles of Distributed Computing*, PODC '05, pages 197–206, New York, NY, USA, 2005. ACM.

[16] Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. *Model Checking and the State Explosion Problem*, pages 1–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[17] F. Cristian. Agreeing on who is present and who is absent in a synchronous distributed system. In *Fault-Tolerant Computing, 1988. FTCS-18, Digest of Papers., Eighteenth International Symposium on*, pages 206–211, June 1988.

[18] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Bastian Pochon. The perfectly synchronized round-based model of distributed computing. *Information and Computation*, 205(5):783 – 815, 2007.

[19] Abdelouahid Derhab and Nadjib Badache. A self-stabilizing leader election algorithm in highly dynamic ad hoc mobile networks. *IEEE Transactions on Parallel and Distributed Systems*, 19(7):926–939, 2008.

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

[21] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.

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

[23] N. Fathollahnejad, R. Pathan, and J. Karlsson. On the probability of unsafe disagreement in group formation algorithms for vehicular ad hoc networks. In *In proceeding of 11th European Dependable Computing Conference (EDCC) 2015*, 7-11 September, 2015,.

[24] Negin Fathollahnejad, Risat Pathan, Emilia Villani, Raul Barbosa, and Johan 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.

[25] Michel Ferreira, Ricardo Fernandes, Hugo Conceição, Wantanee Viriyasitavat, and Ozan K. Tonguz. Self-organized traffic control. In *Proceedings of the seventh ACM international workshop on VehiculAr InterNETworking*, VANET '10, pages 85–90, New York, NY, USA, 2010. ACM.

[26] Christof Fetzer and Flaviu Cristian. A highly available local leader election service. *IEEE Transactions on Software Engineering*, 25(5):603–618, 1999.

[27] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[28] Martin Gardner. *The Bells: versatile numbers that can count partitions of a set, primes and even rhymes.* Scientific American, 1987.

[29] A. Giridhar and P.R. Kumar. Distributed clock synchronization over wireless networks: Algorithms and analysis. In *Decision and Control, 2006 45th IEEE Conference on*, pages 4915–4920, Dec 2006.

[30] J. N. Gray. *Operating Systems: An Advanced Course*, chapter Notes on data base operating systems, pages 393–481. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978.

[31] J.N. Gray. Notes on data base operating systems. In *Operating Systems*, volume 60 of *Lecture Notes in Computer Science*, pages 393–481. 1978.

[32] F. Greve and S. Tixeuil. Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 82–91, June 2007.

[33] J. B. Kenney. Dedicated short-range communications (dsrc) standards in the united states. *Proceedings of the IEEE*, 99(7):1162–1182, July 2011.

[34] H. Kopetz, G. Grünsteidl, and J. Reisinger. *Dependable Computing for Critical Applications*, chapter Fault-Tolerant Membership Service in a Synchronous Distributed Real-Time System, pages 411–429. Springer Vienna, Vienna, 1991.

[35] Rakesh Kumar and Mayank Dave. Mobility models and their affect on data aggregation and dissemination in vehicular networks. *Wireless Personal Communications*, 79(3):2237–2269, 2014.

[36] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: probabilistic model checking for performance and reliability analysis. *SIGMETRICS Perform. Eval. Rev.*, 36(4):40–45, March 2009.

[37] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[38] Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[39] Navneet Malpani, Jennifer L. Welch, and Nitin 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*, DIALM '00, pages 96–103, New York, NY, USA, 2000. ACM.

[40] Navneet Malpani, Jennifer L. Welch, and Nitin 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*, DIALM '00, pages 96–103, New York, NY, USA, 2000. ACM.

[41] S.M. Masum, A.A. Ali, and M.T.-yI. Bhuiyan. Asynchronous leader election in mobile ad hoc networks. In *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, volume 2, pages 5 pp.–, April 2006.

[42] M. Mustafa, M. Papatriantafilou, E. M. Schiller, A. Tohidi, and P. Tsigas. Autonomous tdma alignment for vanets. In *2012 IEEE Vehicular Technology Conference (VTC Fall)*, pages 1–5, Sept 2012.

[43] Tamer Nadeem, Sasan Dashtinezhad, Chunyuan Liao, and Liviu Iftode. Trafficview: Traffic data dissemination using car-to-car communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 8(3):6–19, July 2004.

[44] Till Neudecker, Natalya An, Ozan K. Tonguz, Tristan Gaugel, and Jens Mittag. Feasibility of virtual traffic lights in non-line-of-sight environments. In *Proceedings of the ninth ACM international workshop on Vehicular inter-networking, systems, and applications*, VANET '12, pages 103–106, New York, NY, USA, 2012. ACM.

[45] U.S. Department of Transportation Institute of Transportation Engineers. Traffic signals issue brief 5.

[46] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, April 1980.

[47] Vaskar Raychoudhury, Jiannong Cao, Rajdeep Niyogi, Weigang Wu, and Yi Lai. Top -leader election in mobile ad hoc networks. *Pervasive and Mobile Computing*, 13:181 – 202, 2014.

[48] M. Raynal. Consensus in synchronous systems: a concise guided tour. In *Dependable Computing, 2002. Proceedings. 2002 Pacific Rim International Symposium on*, pages 221 – 228, dec. 2002.

[49] M Raynal and F Tronel. Group membership failure detection: a simple protocol and its probabilistic analysis. *Distributed Systems Engineering*, 6(3):95, 1999.

[50] B.C. Rennie and A.J. Dobson. On stirling numbers of the second kind. *Journal of Combinatorial Theory*, 7(2):116 – 121, 1969.

[51] Valério Rosset, Pedro F. Souto, Paulo Portugal, and Francisco Vasques. *Computer Safety, Reliability, and Security: 26th International Conference, SAFECOMP 2007, Nuremberg, Germany, September 18-21, 2007. Proceedings*, chapter A Reliability Evaluation of a Group Membership Protocol, pages 397–410. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[52] Nicola Santoro and Peter Widmayer. Time is not a healer. In B. Monien and R. Cori, editors, *STACS 89*, volume 349 of *Lecture Notes in Computer Science*, pages 304–313. Springer Berlin Heidelberg, 1989.

[53] Nicola Santoro and Peter Widmayer. Distributed function evaluation in the presence of transmission faults. In Tetsuo Asano, Toshihide Ibaraki, Hiroshi Imai, and Takao Nishizeki, editors, *Algorithms*, volume 450 of *Lecture Notes in Computer Science*, pages 358–367. Springer Berlin Heidelberg, 1990.

[54] Nicola Santoro and Peter Widmayer. Agreement in synchronous networks with ubiquitous faults. *Theor. Comput. Sci.*, 384(2-3):232–249, October 2007.

[55] U. Schmid. How to model link failures: A perception-based fault model. In *IEEE International Conference on Dependable Systems and Networks (DSN)*, pages 57 – 66, July 2001.

[56] Ozan K. Tonguz. Biologically inspired solutions to fundamental transportation problems. *IEEE Communications Magazine*, 49(11):106–115, 2011.

[57] T. Tsuchiya and A. Schiper. Model checking of consensus algorithm. In *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, pages 137 –148, 2007.

[58] 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*, pages 350–360, Oct 2004.

[59] K.H. Wang and Baochun Li. Group mobility and partition prediction in wireless ad-hoc networks. In *Communications, 2002. ICC 2002. IEEE International Conference on*, volume 2, pages 1017–1021 vol.2, 2002.

[60] Masafumi Yamashita and Tsunehiko Kameda. Computing on anonymous networks, part i: characterizing the solvable cases. *IEEE Transaction on Parallel and Distributed Computing*, 7:69–89, 1996.

Appendices

A

Appendix

A.1 Proofs

A.1.1 Proposition.4.1

Two or more processes fail to send their messages in all the $1 \ldots K$ rounds, if and only if, all n processes have incomplete views at the end of K^{th} round.

Proof. The given proposition is a theorem of the form "**A** IF AND ONLY IF **B**". The **A** part is true, when two or more processes fail to send their messages in all the $1 \ldots K$ rounds and the **B** part is true when all n processes have incomplete views at the end of K^{th} round. So, in order to prove the given proposition, we break the "IF AND ONLY IF" proposition into two *lemmas* and prove them separately. First, we prove IF **A** THEN **B**, then we prove IF **B** THEN **A**. This proof approach is called the 'forwards' and 'backwards' proof [ref].

Lemma A.1. *If two or more processes fail to broadcast their message in all the* $1 \ldots K$ *rounds then all processes will have incomplete views at the end of* K^{th} *round.*

We prove Lemma A.1 *by contradiction*, which means that we prove the validity of the given lemma by showing that assuming the lemma being false results in a contradiction. For this, we define Assumption. A.2:

Assumption A.2. *If there are less than two processes (i.e., zero or one process) that fails to broadcast its message for all* K *rounds, then all processes will have incomplete views at the end of* K^{th} *round.*

We show that the above assumption implies a contradiction. First, it is clear that if no process (i.e., zero process) fails to broadcast its message in all K rounds, then all processes receive messages from all other processes in the system and as a result, the views of all processes are complete at the end of K^{th} round. On the other hand, if exactly one process p_x fails to broadcast its message for K rounds, while each of the remaining $n-1$ processes successfully broadcast their message in at least one of the K

rounds, then only process p_x will have the complete view, and all other processes have incomplete views of the system. So we can conclude that there is no execution of the algorithm for which less than two processes fail to broadcast their message for K rounds and all processes have complete views at the end of K^{th} round (i.e., at least one process must have complete view). This contradicts our initial assumption and consequently shows the validity of Lemma A.1.

Lemma A.3. *If all processes have incomplete views at the end of K^{th} round, then two or more processes have failed to broadcast their messages in all the $1 \dots K$ rounds.*

Similar to Lemma A.1 we use proof by contradiction and make the following assumption:

Assumption A.4. *There is at least one process, say process p_x, that has complete view at the end of K^{th} round, and two or more processes have failed to broadcast their messages in all the $1 \dots K$ rounds.*

a process p_x with a complete view at the end of K^{th} round must have received messages from each of the $n-1$ processes at least once. This means that each of the processes in set $\{p_1, \dots, p_{x-1}, p_{x+1}, \dots p_n\}$ must have successfully broadcast their message at least once during K rounds of executions. This contradicts the initial assumption that two or more processes fail to send in all K rounds since $|\{p_1, \dots, p_{x-1}, p_{x+1}, \dots p_n\}| = n-1$. So, Lemma A.3 is valid.
We proved the validity of Lemma A.1 and Lemma A.3, which results in the validity of Proposition 4.1 being proved. □

A.1.2 Proposition.4.2

All processes have complete views at the end of K^{th} round, if and only if, each process successfully broadcasts its message in at least one of the K rounds.

Proof. To prove Proposition 4.2, we use the same given proving approach for Proposition 4.1. We divide the proposition into two lemmas.

Lemma A.5. *If all processes have complete views at the end of the K^{th} round, then each process must have successfully broadcast its message in at least one of the K rounds.*

We use proof by contradiction to prove Lemma A.5. First we define the Assumption. A.6:

Assumption A.6. *All processes have complete views at the end of the K^{th} round, however some processes failed to broadcast their message in all of the K rounds.*

We show that the above statement implies a contradiction. If all processes have complete views at the end of the k^{th} round, it means that each of the processes has received a message from the other processes *at least in one* of the K rounds of execution. This contradicts the assumption that some processes failed to broadcast their message in *all* of the K rounds.

Lemma A.7. *If all processes successfully broadcast their message in at least one of the K rounds, then all processes must have complete views at the end of K^{th} round.*

To prove the validity of Lemma A.7 we define the contradictory Assumption. A.8:

Assumption A.8. *All processes successfully broadcast their message in at least one of the K rounds, and there are some processes that have incomplete views at the end of K^{th} round.*

If all processes have successfully broadcast their message at least once in K rounds of execution, then all processes should have received the message from all other processes and therefore have a complete view of the system at the end of the K^th round. This contradicts the assumption that there are some processes that have incomplete views at the end of K^{th} round.

Given that Lemma A.5 and Lemma A.7 are proved using contradiction, we conclude that Proposition 4.2 is proved to be valid. □

A.1.3 Proposition.4.3

One process has complete view and the remaining $(n-1)$ nodes have incomplete views at the end of K^{th} round, if and only if, exactly one process fails to broadcast its message in all of the K rounds.

Proof. It is evident from the proof of Proposition 4.1 that at most one process can fail to send in all the K rounds if and only if at least one process has complete view at the end of K^{th} round. And, according to Proposition 4.2, no process fails to send in all the K rounds if and only if all processes have complete view. Combining these two observations, it is not difficult to see that exactly one process fails in all K rounds if and only if the view of this process is complete while the views of the remaining $(n-1)$ nodes are incomplete. □

A.1.4 Finding $P_{\mathtt{DG}}$ for Pessimistic Decision Criterion

Lemma. 4.4 In order to have disagreement among processes, it is necessary that **all** processes have complete views at the end of round $R-1$.

Proof. We prove the validity of Lemma 4.4 by contradiction. We know that disagreement occurs if some processes decide to abort while other processes decide to select a value. Based on Algorithm 4, a process p_x decides to abort if its view is not complete at the end of round $R-1$ (i.e., C_1 is false for p_x). On the other hand, if p_x does not have a complete view at the end of round $R-1$, none of the other processes receive a complete view from process p_x in any round including round R (i.e., C_2 is false for all other processes). So, all other processes also decide to abort and as a result, there is no disagreement but agreement to abort. □

Analysis of Case I We divide this case into two different subcases as follows:

Case I.I All processors have complete views at the end of round $r = 1$,

Case I.II All processors have incomplete views at the end of round $r-1$ and have complete views at round r, where $2 \leq r \leq R-1$.

Analysis of Case I.I The probability that all processes have complete views at the end of the first round is $(1-q)^n$. Disagreement occurs if during rounds $2 \ldots R$, there is exactly one process, say p_x, that does not broadcast a complete view in any round while the other $(n-1)$ processes broadcast complete views in at least one of the rounds $2 \ldots R$. The probability that a process p_x does not broadcast its complete view in any round $2 \ldots R$ is $q^{(R-1)}$ and the probability that all other processes in $\Pi - \{p_x\}$ broadcast their complete view in at least one of the rounds $2 \ldots R$ is $(1-q^{R-1})^{n-1}$. Since the process p_x can be selected in n possible ways, the probability of disagreement when all processes have complete views at the end of 1^{st} round is given in A.1.

$$P_{\text{DG } CaseI.I} = (1-q)^n \cdot q^{(R-1)} \cdot (1-q^{R-1})^{n-1} \cdot n \tag{A.1}$$

Analysis of Case I.II According to Proposition 4.1, if the views of all processes are incomplete at the end of round $r-1$, where $2 \leq r \leq R-1$, then at least two or more processes have failed to send in all $1 \ldots r-1$ rounds. For a given round r, the probability that all processes have incomplete views at the end of round $r-1$ is calculated from the given formula in A.2 as below:

$$\sum_{i=2}^{n} \binom{n}{i} (1-q^{r-1})^{n-i} \cdot q^{(r-1)\cdot i} \tag{A.2}$$

where the messages sent from i processes are lost in all $r-1$ rounds, but the other $n-i$ processes successfully broadcast their message in at least one of the $r-1$ rounds, where $2 \leq i \leq n$. According to Case I.II, the view of all processes are complete at the end of round r. So, during round r, all of the i processes in A.2 successfully broadcast their message which happens with the probability $(1-q)^i$. Therefore, for a given r, the probability that all processes have incomplete views at the end of round $r-1$ and have complete views at the end of round r is calculated fromA.3

$$\sum_{i=2}^{n} \binom{n}{i} (1-q^{r-1})^{n-i} \cdot q^{(r-1)\cdot i} \cdot (1-q)^i \tag{A.3}$$

Assuming that all processes have complete views at the end of round r, disagreement occurs if exactly one process, say p_x, does not successfully broadcast its complete view in any of the remaining $R-r$ rounds with the probability of $q^{(R-r)}$, while all remaining $n-1$ processes successfully broadcast their complete views in at least one of the remaining $R-r$ rounds, with the probability of $(1-q^{R-r})^{n-1}$. Process p_x can be selected in n possible ways. For a given r, the probability of disagreement for Case I.II is:

$$\sum_{i=2}^{n} \binom{n}{i} (1-q^{r-1})^{n-i} \cdot q^{(r-1)\cdot i} \cdot (1-q)^i n \cdot q^{(R-r)} \cdot (1-q^{R-r})^{n-1} \tag{A.4}$$

Since r ranges from 2 to $R-1$, the probability that disagreement occurs when all processes have complete views at the end of any round $2 \ldots R-1$ is given as follows:

$$P_{\text{DG } CaseI.II} = \tag{A.5}$$
$$\sum_{r=2}^{R-1} \sum_{i=2}^{n} \binom{n}{i} (1-q^{r-1})^{n-i} \cdot q^{(r-1)\cdot i}$$
$$\cdot (1-q)^i n \cdot q^{(R-r)} \cdot (1-q^{R-r})^{n-1}$$

Combining the probabilities for Case I.I Case I.II, we have the probability of disagreement for Case I given in A.6.

$$
\begin{aligned}
P_{\mathtt{DG}\ CaseI} \;=\; & (1-q)^n \cdot q^{(R-1)} \cdot (1-q^{R-1})^{n-1} \cdot n + \\
& \sum_{r=2}^{R-1} \sum_{i=2}^{n} \binom{n}{i} (1-q^{r-1})^{n-i} \cdot q^{(r-1)\cdot i} \cdot (1-q)^i \\
& \cdot n \cdot q^{(R-r)} \cdot (1-q^{R-r})^{n-1}
\end{aligned}
\tag{A.6}
$$

Analysis of Case II In this case, exactly $n-1$ processes have incomplete views during rounds $r-1$ and all the processes have complete views in round r, where $2 \leq r \leq R-1$. This can happen if exactly one process, say p_x, fails to send in all the $r-1$ rounds and other $n-1$ processes send at least in one of the $r-1$ rounds. Given a particular round r, $2 \leq r \leq R-1$, the probability that any of the n processes fails to send in all $r-1$ rounds is $n \cdot q^{(r-1)}$ and the probability that each of the other $n-1$ processes successfully broadcasts its message in at least one of the $r-1$ rounds is $(1-q^{r-1})^{n-1}$.

Process p_x at round r with the probability of $1-q$, successfully broadcasts its message and as a result, the views of all the processes are complete at the end of round r. We know that the probability that the views of all processes are complete at the end of round r is $n \cdot q^{(r-1)} \cdot (1-q^{r-1})^{n-1} \cdot (1-q)$. Since the view of process p_x is complete at the end of round $r-1$, the send operation by process p_x in round r also confirms the completeness of the view of p_x to all other processes.

Assuming that all processes have a complete view of the system at the round r, disagreement occurs if exactly one process, say p_y, where $p_x \neq p_y$, does not send its complete view in any of the remaining $R-r$ rounds with the probability of $q^{(R-r)}$, while each of the other $n-2$ processes in $\Pi - \{p_x, p_y\}$ broadcast their complete view in at least one of

the remaining $R-r$ rounds with the probability of $(1-q^{R-r})^{n-2}$. Process p_y can be selected in $n-1$ possible ways from set $\Pi - \{p_x\}$. So, for a given r, the probability of disagreement assuming Case II, given than all processes have complete views at the end of round r, can be calculated using the following expression:

$$n \cdot q^{r-1} \cdot (1-q^{r-1})^{n-1} \cdot (1-q) \cdot (n-1) \cdot q^{(R-r)} \cdot (1-q^{R-r})^{n-2} \quad \text{(A.7)}$$

Since r ranges from 2 to $R-1$, the probability of disagreement, given that all the processes have complete views in any of the $2 \ldots R-1$ rounds, is given in A.8:

$$\begin{aligned} P_{\texttt{DG } CaseII} &= \sum_{r=2}^{R-1} n \cdot q^{r-1} \cdot (1-q^{r-1})^{n-1} \\ &\quad \cdot (1-q) \cdot (n-1) \cdot q^{(R-r)} \cdot (1-q^{R-r})^{n-2} \end{aligned} \quad \text{(A.8)}$$

A.1.5 Finding $P_{\texttt{AG}}$ for Pessimistic Decision Criterion

Analysis of Case I' We divide this case into two subcases as follows:

Case I'.I All processes have complete views at the end of round $r = 1$.

Case I'.II All processes have incomplete views at the end of round $r-1$ and have complete views at the end of round r, where $2 \leq r \leq R-1$.

Analysis of Case I'.I The probability that all processes have complete views at the end of round 1 is $(1-q)^n$. Agreement can occur if during rounds $2 \ldots R$, each of the n processes successfully sends its complete view in at least one of the $2 \ldots R$ rounds. The probability of a successful message broadcast by a process in any round $2 \ldots R$ is $(1-q^{(R-1)})$, and the probability of successful complete message broadcast by all of

the n processes is $(1-q^{(R-1)})^n$. The probability of agreement when all processes have complete views at the end of 1^{st} round is given as follows:

$$P_{\texttt{AG}CaseI'.I} = (1-q)^n \cdot (1-q^{R-1})^n \tag{A.9}$$

Analysis of Case I'.II If the views of all processes are incomplete at the end of round $r-1$, where $2 \leq r \leq R-1$, then the messages sent from at least two or more processes have been lost in all $1 \ldots r-1$ rounds. For a given round r, the probability that all processes have incomplete views at the end of round $r-1$ is as below:

$$\sum_{i=2}^{n} \binom{n}{i} (1-q^{r-1})^{n-i} \cdot q^{(r-1)\cdot i} \tag{A.10}$$

where message broadcasts from i out of n processes are failed in all $r-1$ rounds and $n-i$ processes successfully broadcast their message in at least one of the $r-1$ rounds, where $2 \leq i \leq n$. Since the views of all processes are complete at the end of round r for this case, all i processes must successfully broadcast their message during round r which happens with the probability of $(1-q)^i$. Therefore, for some given r, the probability that all processes have incomplete views at the end of round $r-1$ and have complete views at the end of round r can be calculated from A.11.

$$\sum_{i=2}^{n} \binom{n}{i} (1-q^{r-1})^{n-i} \cdot q^{(r-1)\cdot i} \cdot (1-q)^i \tag{A.11}$$

If the views of all processes are complete at the end of round r, agreement on a value occurs, if each of the n processes successfully broadcast their complete message in at least one of the rounds $(r+1) \ldots R$. The probability of a successful message broadcast by one process in any round $(r+1) \ldots R$ is $1-q^{(R-r)}$ and the probability of successful message broadcast by all n processes in $(1-q^{(R-r)})^n$. The probability of agreement when all processes have complete views at the end of r^{th} round is given

in A.12:

$$\sum_{i=2}^{n} \binom{n}{i} (1-q^{r-1})^{n-i} \cdot q^{(r-1)\cdot i} \cdot (1-q)^{i} \cdot (1-q^{(R-r)})^{n} \tag{A.12}$$

Since r ranges from 2 to $R-1$, the probability that disagreement occurs when all the processes have complete view at the end of any round $2 \ldots R-1$ is given as follows:

$$P_{\text{AG}CaseI'.II} = \tag{A.13}$$

$$\sum_{r=2}^{R-1} \sum_{i=2}^{n} \binom{n}{i} (1-q^{r-1})^{n-i} \cdot q^{(r-1)\cdot i} \cdot (1-q)^{i} \cdot (1-q^{(R-r)})^{n} \tag{A.14}$$

Combining the given expressions in A.9 and A.13 to calculate the probabilities of disagreement for Case I'.I and Case I'.II, we have

$$P_{\text{AG}CaseI'} = (1-q)^{n} \cdot (1-q^{R-1})^{n} + \tag{A.15}$$

$$\sum_{r=2}^{R-1} \sum_{i=2}^{n} \binom{n}{i} (1-q^{r-1})^{n-i} \cdot q^{(r-1)\cdot i} \cdot (1-q)^{i} \cdot (1-q^{(R-r)})^{n}$$

Analysis of Case II' This case occurs if exactly one process, say p_x, fails to broadcast its message in all of the $r-1$ rounds while other $n-1$ processes successfully broadcast their message in at least one of the $r-1$ rounds. Given a r, $2 \leq r \leq R-1$, the probability that the message broadcast from any of the n processes fails in all the $r-1$ rounds is $n \cdot q^{r-1}$ and the probability that each of the other $n-1$ processes successfully broadcasts their message in at least one of the $r-1$ rounds is $(1-q^{r-1})^{n-1}$. The process p_x successfully broadcasts its message with the probability of $1-q$ at round r and as a result, the views of all other processes are complete at the end of round r. The probability that the view of each of

the processes is complete at the end of round r is

$$n \cdot q^{(r-1)} \cdot (1 - q^{r-1})^{n-1} \cdot (1 - q) \tag{A.16}$$

Since the view of process p_x is complete at the end of round $r-1$, the send operation by process p_x at round r also confirms that p_x has a complete view of the system.

If the views of all processes are complete at the end of round r, agreement on a value occurs if each of the $n-1$ processes in $\Pi - \{p_x\}$ successfully broadcast their message in at least one of the remaining $R-r$ rounds. This occurs with the probability of $(1-q^{R-r})^{n-1}$. The probability of agreement, given that all processes have complete views at round r, is equal to

$$n \cdot q^{r-1} \cdot (1 - q^{r-1})^{n-1} \cdot (1 - q) \cdot (1 - q^{R-r})^{n-1} \tag{A.17}$$

As r ranges from 2 to $R-1$, the probability of disagreement, given that all processes have complete views in any of the $2 \ldots R-1$ rounds, is given as follows:

$$P_{\texttt{AG}CaseII'} = \sum_{r=2}^{R-1} n \cdot q^{r-1} \cdot (1 - q^{r-1})^{n-1} \cdot (1 - q) \cdot (1 - q^{R-r})^{n-1} \tag{A.18}$$

A.1.6 Finding $P_{\texttt{DG}}$ for Moderately Pessimistic Decision Criterion

Lemma. 4.5 The set Π_x consists of exactly one process p_x which has a complete view at the end of round $R-1$ while all other processes have incomplete views at the end of round $R-1$.

Proof. We show that the set Π_x consists of exactly one process p_x. We prove this using contradiction. We consider the following assumptions:

Assumption A.9. *There are two processes, p_x and $p_{x'}$, in Π_x which decide to select a value.*

Assumption A.10. *All processes in $\Pi - \Pi_x$ decide to abort.*

Based on Assumption. A.9, process p_x must have complete view at the end of round $R-1$ (See Alg. 7). This means that all messages sent from the $n-2$ processes in $\Pi - \Pi_x$ and $p_{x'}$ are successfully delivered in at least one of the $R-1$ rounds. Also from Assumption. A.9, we know that process $p_{x'}$ must have complete view at the end of round $R-1$ and this shows that all messages sent from the $n-2$ processes in $\Pi - \Pi_x$ and p_x are successfully delivered in at least one of the $R-1$ rounds. So, we can conclude that according to Assumption. A.9, messages sent from all processes in $(\Pi - \Pi_x) \cup \{p_x\} \cup \{p_{x'}\}$ are successfully delivered in at least one of the $R-1$ rounds. Obviously $(\Pi - \Pi_x) \cup \{p_x\} \cup \{p_{x'}\}$ indicates the set of all processes (i.e., Π). In other words from Assumption. A.9 and Proposition 4.2 we can conclude that all processes have successfully delivered their messages in at least one of the $R-1$ rounds, which results in complete views for all n processes by the end of round $R-1$, which contradicts Assumption. A.10. So, we can conclude that there is only one process in the set of Π_x. □

A.1.7 Finding $P_{\mathtt{AB}}$ for Moderately Pessimistic Decision Criterion

Given the two cases of execution which result in agreement on abort among processes in 4.3.1, in the following, first we explain how we calculate the probability of Case (a) and then we analyse Case (b).

Case (a) refers to the executions in which all processes have incomplete views at the end of round $R-1$. According to Proposition 4.1, the views of all n processes are incomplete, if and only if, at least two processes fail to send during all $R-1$ rounds. If there are i processes that fail to send in all $R-1$ rounds, where $2 \leq i \leq n$, then all remaining $n-i$ processes successfully broadcast their message in at least one of the $R-1$ rounds.

For a given i, $2 \leq i \leq n$, there are $\sum_{j=1}^{R-1} \binom{R-1}{j})^{n-i} = (2^{R-1}-1)^{n-i}$ possibilities that each of the $(n-i)$ processes successfully broadcasts their message during at least one of the $R-1$ rounds for some given i. Moreover, the i processes out of n are selected in $\binom{n}{i}$ ways, where $2 \leq i \leq n$. Consequently, the number of possibilities that all processes have incomplete views by round $R-1$ is given in A.19.

$$Case(a)_{numberof} = \sum_{i=2}^{n} \binom{n}{i} \cdot (2^{R-1})^{n-i} \tag{A.19}$$

Given that the probability of a message loss is q, the probability that exactly i processes fail to send in all $R-1$ rounds is $(q^{R-1})^i$ while the probability that each of the $n-i$ processes successfully send their message during at least one of the $R-1$ rounds is:
$\left(\sum_{j=1}^{R-1} \binom{R-1}{j} \cdot (1-q)^j \cdot q^{R-1-j} \right)^{n-i} = (1-q^{R-1})^{n-i}$
where $2 \leq i \leq n$. Consequently, the probability that all the processes satisfying the condition in Case (a), agree to abort can be calculated from A.20.

$$P_{\text{AB } Case(a)} = \sum_{i=2}^{n} \binom{n}{i} \cdot (q^{R-1})^i \cdot (1-q^{R-1})^{n-i} \tag{A.20}$$

In the following, we explain how we calculate the probability of Case (b). According to Case (b), we assume a set of processes as Π_x which have complete views by the end of round $R-1$, but in round R they receive incomplete views from all or some of the processes in $\Pi - \Pi_x$. Following, we show that there is exactly one process in Π_x.

Proof. We prove by using contradiction that Π_x consists of exactly one process p_x. We make contradictory assumptions as follows:

Assumption A.11. *There are two processes as p_x and $p_{x'}$ in Π_x which both have complete views at the end of round $R-1$, but in round R they*

receive incomplete views from some or all processes in $\Pi - \Pi_x$.

Assumption A.12. *All processes in* $\Pi - \Pi_x$ *have incomplete views at the end of round* $R-1$.

Based on Assumption. A.11, process p_x has a complete view at the end of round $R-1$ which means that all messages sent from the $n-2$ processes in $\Pi - \Pi_x$ and $p_{x'}$ are successfully delivered in at least one of the $R-1$ rounds. From Assumption. A.11, we know that process $p_{x'}$ has also a complete view at the end of round $R-1$ which shows that all messages sent from the $n-2$ processes in $\Pi - \Pi_{x'}$ and p_x are successfully delivered in at least one of the $R-1$ rounds. Therefore we can conclude that according to Assumption. A.11, messages sent from all processes in $(\Pi - \Pi_x) \cup \{p_x\} \cup \{p_{x'}\}$ are successfully delivered in at least one of the $R-1$ rounds. Obviously $(\Pi - \Pi_x) \cup \{p_x\} \cup \{p_{x'}\}$ indicates the set of all processes (i.e., Π). In other words from Assumption. A.11 and Proposition 4.2 we can conclude that all processes have successfully delivered their messages in at least one of the $R-1$ rounds, which results in complete views for all n processes by the end of round $R-1$. This contradicts Assumption. A.12.

On the other hand, according to Assumption. A.12, considering a process in $\Pi - \Pi_x$ as p_i which has incomplete view by the end of round $R-1$, it means that p_i has not received some or all messages sent from other processes including p_x and $p_{x'}$ during $R-1$ rounds. As we only assume symmetric message losses, if p_i has not received from a process p_x, then $p_{x'}$ should have not received from p_x as well. The same applies to p_i receiving a message from $p_{x'}$, which obviously contradicts Assumption. A.11 that p_x and $p_{x'}$ have complete views by the end of round $R-1$.

So, we proved using contradiction that Π_x consists of exactly one process that fails to broadcast its message in all $R-1$ rounds. As the probability of losing a message broadcast is assumed to be q, we can say that the message broadcasts from the single process in Π_x as p_x, fail in all $R-1$ rounds with the probability of q^{R-1}. Consequently, the other

$n-1$ processes have incomplete views by the end of round $R-1$, while they all successfully broadcast their message in at least one of the $R-1$ rounds of execution, so that the view of p_x is complete by round $R-1$. In round R, p_x receives incomplete views from at least one of $n-1$ processes with the probability of $1-q^{n-1}$. As a result p_x decides to abort. □

Eq. A.21 shows the probability that processes executing the *1-of-n* selection algorithm, meet the given condition in Case (b) and decide to abort.

$$P_{\text{AB } Case(b)} = n \cdot (1-q^{R-1})^{n-1} \cdot q^{R-1} \cdot (1-q^{n-1}) \tag{A.21}$$

As it mentioned in Section 4.3.1, we can calculate the probability of agreement to abort combining the probabilities of two cases, Case (a) and Case (b).

A.2 PRISM Model for *1-of-3* Selection Algorithm

We present a PRISM model of a system of three processes executing the *1-of-3* selection algorithm under the asymmetric failure model. We also model each of the decision criteria for the *1-of-3* selection algorithm. We explain how we can use the same approach in modelling the *1-of-n* selection algorithm with different number of processes and a different communication failure model.

Global Declarations: Constants

The first line of a PRISM model declares its class which in our case is (`dtmc`). Then we insert the list of constant values. The given constant values in Listing A.1 define the system settings under which the *1-of-n* selection algorithm is run: the number of processes (`N`), the number of rounds of execution of the algorithm (`RN`), the probability of losing a message (`Q`) and the decision criterion (`DC`). This model supports three different decision criteria: optimistic `(DC=1)`, moderately pessimistic `(DC=2)` and pessimistic `(DC=3)`.

```
1 dtmc
2 const N=3;       // Number of processes in the network (N cannot be modified)
3 const RN=2;          // Number of rounds in the protocol (RN>=2)
4 const double Q;      // Probability of losing a message (0<=q<=1)
5 const DC=3;          // Decision criterion: OP=1; MP=2; PS=3;
```

Listing A.1: *Declaring the class of the PRISM model and the constant variables*

Constants `N1`, `N2` and `N3` are the processes' identifiers and are used to define which process has the `token` and shall fire the next transition. Constant `v_max` defines the highest value among the processes' initial values and is used to limit the range of the variables that stores processes' values.

Constants `v1_ini`, `v2_ini` and `v3_ini` store the initial value for each

```
6  const N1=1;          // Identity number of Process 1
7  const N2=2;          // Identity number of Process 2
8  const N3=3;          // Identity number of Process 3
9  const v_max=3;       // Maximum value of a process
```

Listing A.2: *Declaring the processes' identities*

process. These values can be chosen randomly between `1` and `v_max`. We know that the choice of the initial values does not affect the results of the algorithm. Regardless of the number of failures, if a process decides to select a value it selects the correct value. Therefore we leave the choice of defining the initial values to the user instead of implementing them as a probabilistic choice in the model, which unnecessarily increases the number of states and transitions.

```
10 const v1_ini=1;      // Initial value of Process 1
11 const v2_ini=2;      // Initial value of Process 2
12 const v3_ini=3;      // Initial value of Process 3
```

Listing A.3: *Declaring the processes' initial values*

Finally, constants `not_last` and `last` are used to define which process receives the `token` after the current process (see expression `next` in Section A.2).

```
13 const not_last=1;    // Auxiliary constant to define the next process
14 const last=0;        // Auxiliary constant to define the next process
```

Listing A.4: *Declaring auxiliary constants*

Global Declarations: Global Variables

In order to model the message exchange among processes we use global variables. At each round, each process writes its current value and view in the global variables and reads the values and views of the other processes from the global variables.

Variable `vi_ext` contains the value of `Process` i and must be defined within the range of `[0..v_max]`. As the range of acceptable values is between `1` and `v_max`, value `0` is used to indicate the loss of a message sent by a process `Process` i (See Listing A.5).

```
16 global v1_ext : [0..v_max] init 0;  // Message value of Process 1
17 global v2_ext : [0..v_max] init 0;  // Message value of Process 2
18 global v3_ext : [0..v_max] init 0;  // Message value of Process 3
```

Listing A.5: *Declaring auxiliary constants*

Variable `wi_vj_ext` contains the `Process` i 's view of `Process` j. This variable is *Boolean* and indicates whether or not `Process` i has received the value of `Process` j during the previous rounds (See Listing A.6).

```
20 global w1_v2_ext : bool init false;     // Process 1 view of Process 2
21 global w1_v3_ext : bool init false;     // Process 1 view of Process 3
22
23 global w2_v1_ext : bool init false;     // Process 2 view of Process 1
24 global w2_v3_ext : bool init false;     // Process 2 view of Process 3
25
26 global w3_v1_ext : bool init false;     // Process 3 view of Process 1
27 global w3_v2_ext : bool init false;     // Process 3 view of Process 2
28
29 global token : [1..N] init 1;           // Token used to coordinate the processes
30 global m_lost : [0..(RN*N*(N-1))] init 0;   // Number of lost messages
```

Listing A.6: *Declaring auxiliary constants*

Linsting A.6 also includes the global variable `token` which is used to coordinate the processes and is within the range of `[1..N]`. The `token`'s value indicates the identifier of a process that must perform the next transition. When `token=2`, only a transition of `Process 2` can be enabled. When `Process 2` performs the enabled transition it passes the token to the next process (`Process 3`) by assigning a new value to it `token=3`, then only a transition of `Process 3` can be enabled. The variable `m_lost` stores the number of lost messages during an execution of the algorithm. This variable is used for verification purposes (e.g. to determine the minimum number of lost messages which results in having disagreement among processes).

Table A.1: *Values of the variables for the* `next` *expression.*

Process	N1	not_last	next
1	1	1	2
2	N2=2	1	3
3	N2=3	last=0	1

Global Declarations: Expressions

The expressions in this section are defined from the perspective of `Process 1`, i.e., with the appropriate variables for `Process 1`. Then, when we define the modules for other processes, we must indicate how the global and local variables are replaced, so that the same expression can be used by other processes. The first expression, `next`, is defined to determine the

```
32 formula next = N1*not_last+1; // Define the next Process in the network
```

Listing A.7: *Definition of* `next` *expression*

next value of the `token`. Table A.1 shows how the variables defined for the `next` expression vary for each process. In the case of `Process 1`, the variables are not replaced, as the expression is defined from the perspective of this process. For `Process 2`, `N1` is replaced by `N2` and `not_last` is not replaced, resulting in `next=3`. Finally, for `Process 3`, `N1` is replaced by `N3` and `not_last` is replaced by the constant `last` (defined as `0`), which results in `next=1`.

```
34 formula v1_new = max(v1,(n1_nf2?v2_ext:0),(n1_nf3?v3_ext:0));  // Process 1 compute new value
35 formula w1_v2_new = w1_v2 | n1_nf2 | (n1_nf3 & w3_v2_ext); // Process 1 update its view of Process
       2
36 formula w1_v3_new = w1_v3 | n1_nf3 | (n1_nf2 & w2_v3_ext); // Process 1 update its view of Process
       3
37 // Process 1 knows that Process 2 view is complete
38 formula w1_c2_new = w1_c2 | (n1_nf2 & (w2_v1_ext & w2_v3_ext));
39 // Process 1 knows that Process 3 view is complete
40 formula w1_c3_new = w1_c3 | (n1_nf3 & (w3_v1_ext & w3_v2_ext));
```

Listing A.8: *`Process 1` computes new value*

We assume that in the case of having asymmetric failures, all messages

Table A.2: *Replacement of variables of* `v1_new` *expression.*

Process1	v1	v2_ext	v3_ext
Process2	v2	v3_ext	v1_ext
Process3	v3	v1_ext	v2_ext

are always sent, which means that their values and views are always copied to the global variables. To model a communication failure, each process `Process` i has a set of internal variables, such as `ni_nfj`, which indicates whether `Process` i has received the message sent from `Process` j during the previous round (`ni_nfj` = true) or not (`ni_nfj= false`). The expressions are redefined in order to consider the other processes' values and views only when their messages are received by `Process 1`. Each process computes its new value at the end of each round depending on the messages it received in that round. The expression `v1_new` is used to compute the new value of a process after a round by comparing the current value (`v1`, which is an internal variable of the module `Process 1`) with the values received from other processes (`v2_ext` and `v3_ext`).

Table A.2 presents how the variables of this expression are replaced when it is used by `Process 2` and `Process 3`.

The expression `v1_new` for computing the new value of `Process 1` uses the condition operator ? to replace the value of `v2_ext` and `v3_ext` with 0 when the corresponding message has not been received by `Process 1`.

The *Boolean* expressions `w1_v2_new` and `w1_v3_new`, given in Listing A.8, compute the view of `Process 1` of the values of `Process 2` and `Process 3`, respectively. For example for the case of `w1_v2_new` the result is *true* if at least one of the following conditions is satisfied:

1. It was already *true* in the previous round (`w1_v2` is *true*, `w1_v2` is an internal variable of the module `Process 1`); or

2. `Process 1` received the message of `Process 2` in the current round (`n1_nf2!=0`); or

3. `Process 1` obtained the information regarding the view of `Process 2` through `Process 3` (`w3_v2_ext`), if it received the message from `Process 3` (i.e., `n1_nf3` is *true*).

`Process 1` updates the Boolean expressions, `w1_c2_new` and `w1_c3_new`, to *true* if it received the information indicating that the view of `Process 2` and `Process 3` are complete, respectively. `Process 1` checks `n1_nf2` and `n1_nf3` to see whether the messages sent from the other processes have been received or not.

```
43 // Optimistic Decision Criterion
44 formula decision_OP = w1_v2 & w1_v3; // Process 1 has complete view at the last round (RN)
45 // Moderately Pessimistic Decision
46 formula received_message_complete = (!n1_nf2 | (n1_nf2 & (w2_v1_ext & w2_v3_ext))) & (!n1_nf3 |
       (n1_nf3 & (w3_v1_ext & w3_v2_ext))); // All messages received by the Process 1 are complete
47 formula decision_MP = (w1_v2 & w1_v3) & received_message_complete; // Process 1 has complete view
       at (RN-1) and all messages received at RN are complete
48 // Pessimistic Decision Criterion
49 formula decision_PS = w1_v2 & w1_v3 & w1_c2 & w1_c3; // Process 1 has complete view at (RN-1) and
       has received complete view from all processes at the last round (RN)
50 // General Decision Formula
51 formula decision = ((DC=1) & decision_OP) |((DC=2) & decision_MP)|((DC=3) & decision_PS); //
       Combine all decision criterea in a single formula
```

Listing A.9: *`Process 1` decision criteria*

Listing A.9 shows the expressions that are defined for each decision criterion to be executed by a process in order to provide a *Boolean* outcome: *true* means to decide on a value and *false* means to decide to abort. For these expressions, the replacement of variables when the expression is called by `Process 2` and `Process 3` is the same as indicated in the previous tables (see also Section A.2). The expression `decision_OP` verifies whether or not `Process 1` has a complete view, i.e., has the value of `Process 2` (i.e., `w1_v2` is *true*) and `Process 3` (i.e., `w1_v3` is *true*).

For the moderately pessimistic decision criterion, a new Boolean expression is defined named as `received_message_complete`. It determines whether or not all the messages received by a process `Process 1` from other processes are complete (i.e., the sending processes, `Process 2` and `Process 3`, have a complete view of the system). For example, if `Process 1` has not received any message from `Process 2` in the last round (i.e., `v2_ext=0`), it won't verify the view of `Process 2` to be com-

plete. On the other hand, if the message sent from `Process 2` has been received by `Process 1` (i.e., `v2_ext!=0`), then the condition for `Process 1` to select a value is that the view of `Process 2` is complete (i.e., both `w2_v1_ext` and `w2_v3_ext` must be *true*). The `decision_MP` expression returns *true* if the view of `Process 1` is complete and the senders of its received messages also have a complete view in the last round. It is important to observe that the requirement that the view of `Process 1` must be complete at `RN-1` is not explicitly embedded in the expression. This requirement is satisfied by not updating the view of `Process 1` in the last round (See Alg. 7).

The `decision_PS` expression verifies whether or not the view of `Process 1` is complete, i.e., `w1_v2` and `w1_v3` are *true*. Also, it checks whether `Process 1` has received a confirmation that all other processes, here `Process 2` and `Process 3` have the complete view of the system or not (i.e., `w1_c2` and `w1_c3` are *true*).

Finally, the expression `decision` combines the three decision criteria in a single expression using the value of the constant `DC`.

Global Description: Internal Variables

Listing A.10 shows the module defined for `Process 1`. In PRISM, the definition of a module starts with the word module, followed by its name, here `Process 1`. Then the internal variables of `Process 1` are given: `S1` refers to `Process 1`'s current state in the execution of the algorithm. The variable `S1` varies according to the current state of `Process 1` as described in Table 4.4 and Fig. 4.7. For example `S1=0` is equivalent to `S0`[1]. The current round of execution of `Process 1` is defined using variable `RN1`. The variable `d1` refers to `Process 1`'s decision and `v1` refers to its current value which is initiated as `v1_ini`. The current views of `Process 1` of the values of other processes are given as `w1_v2` and `w1_v3`. Finally

[1]It is important to observe that `S1` is defined in order to help the organization and understanding of the module. The actual state of the module results from the combination of the value of all its variables, not only `S1`.

```
53 module Process_1
54 s1 : [0..N+2] init 1;  // Process 1 current state
55 RN1 : [0..RN] init 0; // Current round
56 v1 : [0..v_max] init v1_ini; // Process 1 value
57 d1 : bool init false; // Process 1 decision
58
59 // Status of the message from the other processs
60 n1_nf2 : bool init true; // Process 1 has not received the message of Process 2
61 n1_nf3 : bool init true; // Process 1 has not received the message of Process 3
62 // Process 1 view of other processes
63 w1_v2 : bool init false; // Process 1 has the view of Process 2
64 w1_v3 : bool init false; // Process 1 has the view of Process 3
65 // Process 1 has confirmation that other processes have complete view
66 w1_c2 : bool init false; // Process 1 has confirmation from Process 2
67 w1_c3 : bool init false; // Process 1 has confirmation from Process 3
68 // Process 1 sends its message;
69 [] s1=1 & token=N1 & RN1<RN -> 1:(s1'=2) & (token'=next) & (v1_ext'=v1) & (w1_v2_ext'=w1_v2) &
       (w1_v3_ext'=w1_v3) & (RN1'=RN1+1);
70 // Process 1 receives or loses the message of Process 2
71 [] s1=2 & token=N1 & RN1<=RN & (m_lost<(RN*N*(N-1))) -> (1-Q): (s1'=3) & (n1_nf2'=true) + Q:
       (s1'=3) & (n1_nf2'=false) & (m_lost'=m_lost+1);
72 // Process 1 receives or loses the message of Process 3
73 [] s1=3 & token=N1 & RN1<=RN & (m_lost<(RN*N*(N-1))) -> (1-Q): (s1'=4) & (n1_nf3'=true) + Q:
       (s1'=4) & (n1_nf3'=false) & (m_lost'=m_lost+1);
74 // Not last round, Process 1 computes the messages of other processes: updates its value, views
       and confirmations;
75 [] s1=N+1 & token=N1 & RN1<RN -> 1: (s1'=1) & (v1'=v1_new) & (w1_v2'=w1_v2_new) &
       (w1_v3'=w1_v3_new) & (w1_c2'=w1_c2_new) & (w1_c3'=w1_c3_new)&(token'=next); //&
       (w1_c2'=w1_c2_new) & (w1_c3'=w1_c3_new)
76 // Last round, Process 1 computes the messages of other processes: updates its confirmations and,
       only for OP, updates value and views;
77 [] s1=N+1 & token=N1 & RN1=RN -> 1: (s1'=N+2) & (w1_c2'=w1_c2_new) & (w1_c3'=w1_c3_new) &
       (v1'=(DC=1)?v1_new:v1) & (w1_v2'=(DC=1)?w1_v2_new:w1_v2) & (w1_v3'=(DC=1)?w1_v3_new:w1_v3);
78 // Process 1 decides -> agree or abort
79 [] s1=N+2 & token=N1 -> 1: (s1'=0) & (token'=next) & (d1'= decision);
80 endmodule
```

Listing A.10: *Module of* `Process 1`

the Boolean expressions `w1_c2` and `w1_c3` used by `Process 1` are set to *true* when `Process 1` received a message from `Process 2` and `Process 3` respectively, when they each have a complete view of the system.

Global Description: Transitions

The transitions in a process module to different states are specified using a set of *guarded commands*. The guard is a *Boolean* expression that specifies the conditions under which the transition can be executed. A transition is defined with a guard and an *action* which are separated by an arrow: (`[ ] guard` $\rightarrow$ `action`). The action specifies how the internal and global variables are updated when the transition is performed. The given square brackets in the beginning of the definition of a transition are used for synchronizations among modules. However, as in our models we do not consider synchronizations, we leave the brackets empty as [ ]. The

update of each variable must be included in parentheses. The parentheses are combined with an & operator. Example: [ ] `guard` $\rightarrow$ ($update_1$) & ($update_2$) & ($update_3$).

In the case of probabilistic transitions, the action is composed of a set of possible actions with their corresponding probabilities which are separated by the plus signals. Example: [ ] `guard` $\rightarrow p1$: `action 1`+ $p2$: `action 2`+ $p3$: `action 3`. As for the case of the deterministic transitions, each action may be composed of one or more updates. The first transition of the module performs the message sending. The guard of this transition specifies that the module must be in the initial state (`S1=1`), has the token (`N1=1`), and must not have completed the last round (`RN1<RN`)[2]. We assume that `Process 1` always sends its message, i.e., copies its value and views to the global variables, without any failure. The global variables associated with `Process 1`'s value (i.e., `v1_ext`) and view (i.e., `w1_v2_ext` and `w1_v3_ext`) are updated with the current values of the internal variables. Additionally, the current round of `Process 1` is incremented `(RN1'=RN1+1)`, the token is passed to the next process `(token'=next)` and `Process 1` moves to state `S1=2`.

```
81  module Process_2=Process_1 [N1=N2, s1=s2, v1=v2, d1=d2, RN1=RN2, n1_nf2=n2_nf3, n1_nf3=n2_nf1,
        w1_v2=w2_v3, w1_v3=w2_v1, w1_c2=w2_c3, w1_c3=w2_c1, v1_ext=v2_ext, v2_ext=v3_ext,
        v3_ext=v1_ext, w1_v2_ext=w2_v3_ext, w1_v3_ext=w2_v1_ext, w2_v1_ext=w3_v2_ext,
        w2_v3_ext=w3_v1_ext, w3_v1_ext=w1_v2_ext, w3_v2_ext=w1_v3_ext, v1_ini=v2_ini] endmodule
82  module Process_3=Process_1 [N1=N3, s1=s3, v1=v3, d1=d3, RN1=RN3, n1_nf2=n3_nf1, n1_nf3=n3_nf2,
        w1_v2=w3_v1, w1_v3=w3_v2, w1_c2=w3_c1, w1_c3=w3_c2, v1_ext=v3_ext, v2_ext=v1_ext,
        v3_ext=v2_ext, w1_v2_ext=w3_v1_ext, w1_v3_ext=w3_v2_ext, w2_v1_ext=w1_v3_ext,
        w2_v3_ext=w1_v2_ext, w3_v1_ext=w2_v3_ext, w3_v2_ext=w2_v1_ext, v1_ini=v3_ini,
        not_last=last] endmodule
```

Listing A.11: *The module for `Process 2` and `Process 3`*

As we see in Listing A.10, a set of `N-1` probabilistic transitions are added in order to define, at each round, whether the messages sent from other processes have been received by `Process 1` or not. When the message is lost, `n1_nfi` is set to *false* and `m_lost` is incremented. The following transition describes the computation of the last round `(RN1=RN)`. The

[2]We observe that here we show the text of a transition broken in many lines due to the lack of space, however, in PRISM a transition must be specified in a single line.

main difference between this and the previous transition is that the value and views of `Process 1` are updated if and only if the decision criterion is the optimistic one `(DC=1)`. The expression $x' = (c)?a : b$ means that if c is true xâĂŹ $= a$, otherwise xâĂŹ $= b$. Also, differently from the previous transition, in this case `Process 1` goes to `S1=3` and does not pass the token.

After computing the last round, `Process 1` is at `S1=3` and makes a decision using the corresponding expression. It then goes to the final state `S1=0` and passes the token to the next process. The module is closed with `end_module`.

Modules of other processes

Listing A.11 shows the definition of `Process 2` and `Process 3` as a copy of `Process 1`. In this case, PRISM imposes that all the internal variables must be renamed. The external variables may be replaced or not by other external variables that have already been defined in the appropriate section.

The general rule adopted in the definition of a new `Process` i ($1 < i <= N$) as a copy of `Process 1` is as follows:

For each internal, global or constant variable used by `Process 1` and named as `X`$_j$ or `X`$_j$`_Y` or `X`$_j$ `_Yw`, where `X` and `Y` are the names of the variables and `j` and `w` are the references to other processes in the interval `[1..N]`:

- If $(j + i - 1 <= n)$ and/or $(w + i - 1 <= n)$ then replace it with $(j + i - 1)$ and/or $(w + i - 1)$.
- If $(j + i - 1 > n)$ and/or $(w + i - 1 > n)$ then replace it with $(j + i - 1 - n)$ and/or $(w + i - 1 - n)$.

For the definition of `Process` i, we need to add the replacement `not_last=last`. The application of the general rule results in the following definition of `Process 2` and `3`: Also, the variables `n1_nfi` must be renamed in the definition of `Process 2` and `Process 3`.

Specifying Verification Properties

We specify the verification properties using an extension of probabilistic temporal logic, which combines temporal relationships between events with probabilistic quantifiers. The specification language used by PRISM is named the Probabilistic Computation Tree Logic ($PCTL$), derived from the well-known Computation Tree Logic (CTL). We use $PCTL$ to calculate the probability that all processes reach the final state in a given condition (agreement, abort or disagreement). We specify the verification properties as they are shown in Listings A.12, A.13, A.14 and A.15.

```
1 P=? [ F (s1=0)&(s2=0)&(s3=0)&((d1!=d2)|(d2!=d3))]
```

Listing A.12: *Property (1), Probability of disagreement*

```
1 P= ? [ F(s1=0)&(s2=0)&(s3=0)&(d1=false)&(d2=false)&(d3=false)]
```

Listing A.13: *Property (2), Probability of agreement to abort*

```
1 P= ? [ F(s1=0)&(s2=0)&(s3=0)&(d1=true)&(d2=true)&(d3=true)]
```

Listing A.14: *Property (3), Probability of agreement on a value*

```
1 P= ? [ F(s1=0)&(s2=0)&(s3=0)&((d1!=d2)|(d2!=d3))&(m-lost=1)]
```

Listing A.15: *Property (4), Probability of disagreement with a specific number of lost messages*

Property (1) given in Listing A.12 refers to the probability `(P=?)` that eventually `(F)` the system reaches a state where all processes are in the final state`((s1=0)&(s2=0)&(s3=0)&(s4=0))` but have made different decisions as `((d1!=d2)|(d2!=d3))` that is the probability of disagreement among processes. Similarly, the next two given properties are to calculate

the probabilities of agreement to abort and agreement on a value. The last property given in Listing A.15 is to calculate the probability of disagreement when there are certain number of lost messages. We define this property in order to be able to compare different decision criteria with respect to the minimum number of lost messages to have disagreement among processes.

Model checking is one of the most effective formal techniques for verifying distributed systems and algorithms, which is based on exhaustively and automatically checking a system model against its specified properties to be hold for all reachable states. However, an important challenge with the application of model checking to real systems is the state-space explosion problem. This problem arises whenever it becomes computationally too expensive to examine the entire state-space. Due to the problem of state explosion, we are only able to calculate probabilities for a system with 3 processes. For larger number of processes, we use PRISM to estimate the value of a given property using simulation.

In Appendix A.3 we present our approach in expanding the PRISM model of the *1-of-3* selection algorithm to a *1-of-4* selection algorithm. We also show that to model the algorithm with more than four processes, we need to repeat the same procedure.

A.3 PRISM Model for *1-of-4* Selection Algorithm

We describe how to expand the PRISM model of a *1-of-3* selection algorithm, given in Section A.2, for a system of four processes. We also show that to model the algorithm with more than four processes, we need to repeat the same procedure ($N > 3$).

Global Declarations: Constants

Listing A.16 illustrates the constants which are modified or added to the PRISM model of the *1-of-3* selection algorithm. The number of processes is modified to 4. The identity number of `Process 4` with its initial value are added to the constant variables.

```
2  const N=4;       // Number of processes in the network (N cannot be modified)
3  const RN=2;          // Number of rounds in the protocol (RN>=2)
4  const double Q;      // Probability of losing a message (0<=q<=1)
5  const DC=3;          // Decision criterion: OP=1; MP=2; PS=3;
6  const N1=1;          // Identity number of Process 1
7  const N2=2;          // Identity number of Process 2
8  const N3=3;          // Identity number of Process 3
9  const N4=4;          // Identity number of Process 4
10 const v1_ini=1;      // Initial value of Process 1
11 const v2_ini=2;      // Initial value of Process 2
12 const v3_ini=3;      // Initial value of Process 3
13 const v4_ini=4;      // Initial value of Process 4
14 const not_last=1;    // Auxiliary constant to define the next process
15 const last=0;        // Auxiliary constant to define the next process
16 global v1_ext : [0..N] init 0;  // Message value of Process 1
```

Listing A.16: *Global declarations: Constants*

Global Declarations: Global Variables

Listing A.17 shows the constant variables that must be added to the list of global variables: the current value of `Process 4`, the view of each of the other processes of `Process 4`, and the `Process 4`'s view of other processes.

```
17 global v2_ext : [0..N] init 0;  // Message value of Process 2
18 global v3_ext : [0..N] init 0;  // Message value of Process 3
19 global v4_ext : [0..N] init 0;  // Message value of Process 4
20 global w1_v2_ext : bool init false;      // Process 1 view of Process 2
21 global w1_v3_ext : bool init false;      // Process 1 view of Process 3
22 global w1_v4_ext : bool init false;      // Process 1 view of Process 4
23 global w2_v1_ext : bool init false;      // Process 2 view of Process 1
24 global w2_v3_ext : bool init false;      // Process 2 view of Process 3
25 global w2_v4_ext : bool init false;      // Process 2 view of Process 4
26 global w3_v1_ext : bool init false;      // Process 3 view of Process 1
27 global w3_v2_ext : bool init false;      // Process 3 view of Process 2
28 global w3_v4_ext : bool init false;      // Process 3 view of Process 4
29 global w4_v1_ext : bool init false;      // Process 4 view of Process 1
30 global w4_v2_ext : bool init false;      // Process 4 view of Process 2
31 global w4_v3_ext : bool init false;      // Process 4 view of Process 3
32 global token : [1..N] init 1;       // Token used to coordinate the Processs
33 global m_lost: [0..RN*N*(N-1)] init 0;  // Number of lost messages
```

Listing A.17: *Global declarations: Global variables*

Global Declarations:Expressions

Listing A.18 shows the expression for computing the value, views and confirmations of `Process 1` are updated to consider the value and views of `Process 4`. Two new expressions are created for computing `Process 1` view and confirmation of `Process 4`. The decision criteria expressions

```
36 formula v1_new = max(v1,(n1_nf2?v2_ext:0),(n1_nf3?v3_ext:0),(n1_nf4?v4_ext:0)); // Process 1
       compute new value
37 formula w1_v2_new = w1_v2 | n1_nf2 | (n1_nf3 & w3_v2_ext) | (n1_nf4 & w4_v2_ext) ; // Process 1
       update its view of Process 2
38 formula w1_v3_new = w1_v3 | n1_nf3 | (n1_nf2 & w2_v3_ext) | (n1_nf4 & w4_v3_ext) ; // Process 1
       update its view of Process 3
39 formula w1_v4_new = w1_v4 | n1_nf4 | (n1_nf2 & w2_v4_ext) | (n1_nf3 & w3_v4_ext) ; // Process 1
       update its view of Process 4
40 formula w1_c2_new = w1_c2 | (n1_nf2 & (w2_v1_ext & w2_v3_ext & w2_v4_ext)); // Process 1 knows
       that Process 2 view is complete
41 formula w1_c3_new = w1_c3 | (n1_nf3 & (w3_v1_ext & w3_v2_ext & w3_v4_ext)); // Process 1 knows
       that Process 3 view is complete
42 formula w1_c4_new = w1_c4 | (n1_nf4 & (w4_v1_ext & w4_v2_ext & w4_v3_ext)); // Process 1 knows
       that Process 4 view is complete
43
44 // Optimistic Decision
45 formula decision_OP = w1_v2 & w1_v3 & w1_v4; // Process 1 has complete view at RN
46 // Pessimistic Decision
47 formula decision_PS = w1_v2 & w1_v3 & w1_v4 & w1_c2 & w1_c3 & w1_c4; // Process 1 has complete
       view at (RN-1) and received complete view from all processes at RN
48 // General Decision Formula
49 formula decision = ((DC=1) & decision_OP) | ((DC=3) & decision_PS); // Combine all decision
       criterea in a single formula
50 // Label definitions for propery specifications
```

Listing A.18: *Updating expressions*

are also updated to include the view and confirmation of `Process 4`.

Module Description: Internal Variables

Listing A.19 shows the view of `Process 1` and the variable are added to the list of internal variables. in order to keep the information regarding the completeness of `Process 4`.

```
label "p1_AB" = (d1=false); //Process p1 aborts
label "p2_AB" = (d2=false); //Process p2 aborts
label "p3_AB" = (d3=false); //Process p3 aborts
label "p4_AB" = (d4=false); //Process p3 aborts
//Decide
label "p1_DC" = (d1!=false); //Process p1 decides
label "p2_DC" = (d2!=false); //Process p2 decides
label "p3_DC" = (d3!=false); //Process p3 decides
label "p4_DC" = (d4!=false); //Process p3 decides

label "p1_NEQ_p2" = (d1!=d2); // p1's decision is not equal to p2's decision
label "p1_NEQ_p3" = (d1!=d3); // p1's decision is not equal to p3's decision
label "p1_NEQ_p4" = (d1!=d4); // p1's decision is not equal to p4's decision
label "p2_NEQ_p3" = (d2!=d3); // p2's decision is not equal to p3's decision
label "p2_NEQ_p4" = (d2!=d4); // p2's decision is not equal to p4's decision
label "p3_NEQ_p4" = (d3!=d4); // p3's decision is not equal to p4's decision
```

Listing A.19: *`Process 1` 's internal variables*

Module Description: Transitions

Compared to the *1-of-3* selection algorithm, the first transition is modified so that `Process 1` also sends or fails to send its view of `Process 4`. Furthermore, at the compute phase, `Process 1` view and confirmation of `Process 4` must be updated. Additionally, a transition is added to `Process 1` in order to include the reception of the message from `Process 4`.

```
label "p1_EQ_p2" = (d1=d2); // p1's decision is equal to p2's decision
label "p1_EQ_p3" = (d1=d3); // p1's decision is equal to p3's decision
label "p1_EQ_p4" = (d1=d4); // p1's decision is equal to p4's decision
label "p2_EQ_p3" = (d2=d3); // p2's decision is equal to p3's decision
label "p2_EQ_p4" = (d2=d4); // p2's decision is equal to p4's decision
label "p3_EQ_p4" = (d3=d4); // p3's decision is equal to p4's decision


module Process_1
s1 : [0..N+2] init 1;  // Process 1 current state
RN1: [0..RN] init 0; // Current round
v1 : [0..N] init v1_ini; // Process 1 value
d1: bool init false; // Process 1 decision
```

Listing A.20: *`Process 1` transitions*

Modules of other processes

In the definition of `Process 2` and `3`, the new internal and global variables related to `Process 4` are introduced in the list of variables that are renamed or replaced. Moreover, all the renaming and replacing previously defined for the case of `N=3` must be revised in other to follow the general rule defined in Section A.2. One example is the variable `w1_v3` in the definition of `Process 3`: in the case of `N=3` it is renamed as `w2_v1`, while for `N=4` it is renamed as `w2_v4`. Finally, the definition of `Process 4` as a copy of `Process 1` is added to the model (See Listing A.21).

```
82 // Status of the message of the other Processs
83 n1_nf2: bool init true; // Process 1 has not received the message of Process 2
84 n1_nf3: bool init true; // Process 1 has not received the message of Process 3
```

Listing A.21: *Processes' modules*

Specifying Verification Properties

We update the properties to include the final state and the decision of `Process 4` as they are given in Listings A.22, A.23, A.24 and A.25.

```
1 P = ? [F(s1=0)&(s2=0)&(s3=0)&(s4=0)&((d1!=d2)|(d1!=d3)|(d3!=d4)|(d2!=d3)|(d2!=d4)|(d3!=d4))]
```

Listing A.22: *Property (1), Probability of disagreement*

```
1 P = ?[F(s1=0)&(s2=0)&(s3=0)&(s4=0)&(d1=false)&(d2=false)&(d3=false)&(d4=false)]
```

Listing A.23: *Property (2), Probability of abort*

```
1 P = ? [F(s1=0)&(s2=0)&(s3=0)&(s4=0)&(d1=true)&(d2=true)&(d3=true)&(d4=true)]
```

Listing A.24: *Property (3), Probability of agreement*

```
1 $ P = ? [F(s1=0)&(s2=0)&(s3=0)&(s4=0)&((d1!=d2)|(d2!=d3)|(d3!=d4))&(m-lost=1)]$
```

Listing A.25: *Property (4), Probability of disagreement with a specific number of lost messages*

A.4 PRISM Models for the Group Formation Algorithms

In the following, we present the PRISM models for the group formation algorithm for systems of three, four and five participating processes, $n = 3$,$n = 4$ and $n = 5$.

A.4.1 n=3

```
dtmc
const N=3;      // Number of processes in the network (N cannot be modified)
const RN=2;         // Number of rounds in the protocol (RN>=2)
const double Q;     // Probability of losing a message (0<=q<=1)
const double c;     //Controlability of the oracles

const N1=1;         // Identity number of Process 1
const N2=2;         // Identity number of Process 2
const N3=3;         // Identity number of Process 3

const o1=3;     //Oracle value is given randomely
const o2=3;     //Oracle value is given randomely
const o3;       //Oracle value is given randomely

const p1_1_ini= 1;  // Initial value of Process 1 of Process 1
const p1_2_ini= 0;  // Initial value of Process 1 of Process 2
const p1_3_ini= 0;  // Initial value of Process 1 of Process 3
const p2_1_ini= 0;  // Initial value of Process 2 of Process 1
const p2_2_ini= 1;  // Initial value of Process 2 of Process 2
const p2_3_ini= 0;  // Initial value of Process 2 of Process 3
const p3_1_ini= 0;  // Initial value of Process 3 of Process 1
const p3_2_ini= 0;  // Initial value of Process 3 of Process 2
const p3_3_ini= 1;  // Initial value of Process 3 of Process 3
const not_last=1;   // Auxiliary constant to define the next process
const last=0;       // Auxiliary constant to define the next process

global p1_1_ext : [0..1] init 1;    // Message value of Process 1 of Process 1
global p1_2_ext : [0..1] init 0;    // Message value of Process 1 of Process 2
global p1_3_ext : [0..1] init 0;    // Message value of Process 1 of Process 3

global p2_1_ext : [0..1] init 0;    // Message value of Process 2 of Process 1
global p2_2_ext : [0..1] init 1;    // Message value of Process 2 of Process 2
global p2_3_ext : [0..1] init 0;    // Message value of Process 2 of Process 3

global p3_1_ext : [0..1] init 0;    // Message value of Process 3 of Process 1
global p3_2_ext : [0..1] init 0;    // Message value of Process 3 of Process 2
global p3_3_ext : [0..1] init 1;    // Message value of Process 3 of Process 3

global token : [1..N] init 1;       // Token used to coordinate the processes
formula next = N1*not_last+1;       // Define the next process to receive the token

formula p1_1_new = max (p1_1,(p2_1_ext * n1_nf2),(p3_1_ext * n1_nf3)); // Process 1 compute new
      value of Process 1
formula p1_2_new = max (p1_2,(p2_2_ext * n1_nf2),(p3_2_ext * n1_nf3)); // Process 1 compute new
      value of Process 2
formula p1_3_new = max (p1_3,(p2_3_ext * n1_nf2),(p3_3_ext * n1_nf3)); // Process 1 compute new
      value of Process 3

formula m1_new= p1_1 + p1_2 + p1_3; // length of view vector of process 1
formula m2_new= p2_1 + p2_2 + p2_3; // length of view vector of process 2
formula m3_new= p3_1 + p3_2 + p3_3; // length of view vector of process 2

// Label definitions for propery specifications
label "p1_AB" = (p1_d1=0) &(p1_d2=0) &(p1_d3=0); //Process p1 aborts
label "p2_AB" = (p2_d1=0) &(p2_d2=0) &(p2_d3=0); //Process p2 aborts
label "p3_AB" = (p3_d1=0) &(p3_d2=0) &(p3_d3=0); //Process p3 aborts
label "p1_DC" = (p1_d1!=0) |(p1_d2!=0) |(p1_d3!=0); //Process p1 decides
label "p2_DC" = (p2_d1!=0) |(p2_d2!=0) |(p2_d3!=0); //Process p2 decides
label "p3_DC" = (p3_d1!=0) |(p3_d2!=0) |(p3_d3!=0); //Process p3 decides

label "p1_NEQ_p2" =(p1_d1!=p2_d1) |(p1_d2!=p2_d2) |(p1_d3!=p2_d3) ; //Process p1 is not equal to p2
label "p1_NEQ_p3" =(p1_d1!=p3_d1) |(p1_d2!=p3_d2) |(p1_d3!=p3_d3) ; //Process p1 is not equal to p3
label "p2_NEQ_p3" =(p2_d1!=p3_d1) |(p2_d2!=p3_d2) |(p2_d3!=p3_d3) ; //Process p1 is not equal to p3

label "p1_EQ_p2" =(p1_d1=p2_d1) &(p1_d2=p2_d2) &(p1_d3=p2_d3) ; //Process p1 is equal to p2
label "p1_EQ_p3" =(p1_d1=p3_d1) &(p1_d2=p3_d2) &(p1_d3=p3_d3) ; //Process p1 is equal to p3
label "p2_EQ_p3" =(p2_d1=p3_d1) &(p2_d2=p3_d2) &(p2_d3=p3_d3) ; //Process p1 is equal to p3
```

Listing A.26: *Prism Model for a system of three processes executing the group formation algorithm, part (1)*

```
65
66  module Process_1
67  s1 : [0..N+3] init 1;   // Process 1 current state
68  RN1 : [0..RN] init 0;   // Current round
69  m1 : [0..N] init 1;
70
71  p1_d1 : [0..1] init 0;  // Process 1 decision of process 1
72  p1_d2 : [0..1] init 0;  // Process 1 decision of process 2
73  p1_d3 : [0..1] init 0;  // Process 1 decision of process 3
74
75  p1_1 : [0..1] init p1_1_ini;    // Process 1 value of process 1
76  p1_2 : [0..1] init p1_2_ini;    // Process 1 value of process 2
77  p1_3 : [0..1] init p1_3_ini;    // Process 1 value of process 3
78
79  // Status of the message from the other processs
80  n1_nf2 :[0..1] init 0;   // Process 1 has not received the message of Process 2
81  n1_nf3 :[0..1] init 0;   // Process 1 has not received the message of Process 3
82
83  // Process 1 sends or loses its message;
84  [] s1=1 & token=N1 & RN1<RN -> 1:(s1'=2) & (token'=next) & (p1_1_ext'=p1_1) & (p1_2_ext'=p1_2) &
        (p1_3_ext'=p1_3) & (RN1'=RN1+1);
85  // Process 1 receives or loses the message of Process 2
86  [] s1=2 & token=N1 & RN1<=RN -> (1-Q): (s1'=3) & (n1_nf2'=1) + Q: (s1'=3) & (n1_nf2'=0);
87  // Process 1 receives or loses the message of Process 3
88  [] s1=3 & token=N1 & RN1<=RN -> (1-Q): (s1'=4) & (n1_nf3'=1) + Q: (s1'=4) & (n1_nf3'=0);
89  // Not last round, Process 1 computes the messages of other processes: updates its values;
90  [] s1=N+1 & token=N1 & (RN1<RN) -> 1: (s1'=1) & (p1_1'=p1_1_new) & (p1_2'=p1_2_new) &
        (p1_3'=p1_3_new) & (token'=next);
91  // Last round, Process 1 computes the messages of other processes:  updates its values;
92  //[] s1=N+1 & token=N1 & RN1=RN -> 1: (s1'=N+2) & (p1_1'=p1_1_new) & (p1_2'=p1_2_new) &
        (p1_3'=p1_3_new) & (m1'=m1_new)& (orc1'=orc1_new) & (token'=next);
93  [] s1=N+1 & token=N1 & RN1=RN -> 1: (s1'=N+2) & (p1_1'=p1_1_new) & (p1_2'=p1_2_new) &
        (p1_3'=p1_3_new) & (token'=next);
94  [] s1=N+2 & token=N1 & RN1=RN -> 1: (s1'=N+3) &  (m1'=m1_new) & (token'=next);
95  // Process 1 decides -> agree or abort
96  [] s1=N+3 & token=N1 & (m1 >= (c* o1))->1: (s1'=0) & (token'=next) & (p1_d1'=p1_1) & (p1_d2'=
        p1_2)& (p1_d3'= p1_3);
97  [] s1=N+3 & token=N1 & (m1 < (c*o1))-> 1: (s1'=0) & (token'=next) & (p1_d1'=0) & (p1_d2'= 0) &
        (p1_d3'= 0);
98  endmodule
99  module Process_2 = Process_1 [N1=N2, s1=s2, m1=m2, o1=o2, p1_d1=p2_d2, p1_d2=p2_d3, p1_d3=p2_d1,
        n1_nf2=n2_nf3, n1_nf3=n2_nf1, p1_1=p2_2, p1_2=p2_3, p1_3=p2_1, p1_1_ini=p2_2_ini,
        p1_3_ini=p2_1_ini, p1_2_ini=p2_3_ini, p1_1_ext=p2_2_ext, p1_2_ext=p2_3_ext,
        p1_3_ext=p2_1_ext, p2_1_ext=p3_2_ext, p2_2_ext=p3_3_ext, p2_3_ext=p3_1_ext,
        p3_1_ext=p1_2_ext, p3_2_ext=p1_3_ext, p3_3_ext=p1_1_ext, RN1=RN2] endmodule
100 module Process_3 = Process_1 [N1=N3, s1=s3, m1=m3, o1=o3, p1_d1=p3_d3, p1_d2=p3_d1, p1_d3=p3_d2,
        n1_nf2=n3_nf1, n1_nf3=n3_nf2, p1_1=p3_3, p1_2=p3_1, p1_3=p3_2, p1_1_ini=p3_3_ini,
        p1_3_ini=p3_2_ini, p1_2_ini=p3_1_ini, p1_1_ext=p3_3_ext, p1_2_ext=p3_1_ext,
        p1_3_ext=p3_2_ext, p2_1_ext=p1_3_ext, p2_2_ext=p1_1_ext, p2_3_ext=p1_2_ext,
        p3_1_ext=p2_3_ext, p3_2_ext=p2_1_ext, p3_3_ext=p2_2_ext, RN1=RN3,not_last=last] endmodule
```

Listing A.27: *Prism Model for a system of three processes executing the group formation algorithm, part (2)*

A.4.2 n=4

```
dtmc
const N=4;      // Number of processes in the network (N cannot be modified)
const RN=2;         // Number of rounds in the protocol (RN>=2)
const double Q;     // Probability of losing a message (0<=q<=1)
const double c;     // controlability of the oracle value
const N1=1;         // Identity number of Process 1
const N2=2;         // Identity number of Process 2
const N3=3;         // Identity number of Process 3
const N4=4;         // Identity number of Process 3
const o1;       //Oracle value for process 1
const o2;       //Oracle value for process 2
const o3;       //Oracle value for process 3
const o4;       //Oracle value for process 4

const p1_1_ini= 1;  // Initial value of Process 1 of Process 1
const p1_2_ini= 0;  // Initial value of Process 1 of Process 2
const p1_3_ini= 0;  // Initial value of Process 1 of Process 3
const p1_4_ini= 0;  // Initial value of Process 1 of Process 4
const p2_1_ini= 0;  // Initial value of Process 2 of Process 1
const p2_2_ini= 1;  // Initial value of Process 2 of Process 2
const p2_3_ini= 0;  // Initial value of Process 2 of Process 3
const p2_4_ini= 0;  // Initial value of Process 2 of Process 4
const p3_1_ini= 0;  // Initial value of Process 3 of Process 1
const p3_2_ini= 0;  // Initial value of Process 3 of Process 2
const p3_3_ini= 1;  // Initial value of Process 3 of Process 3
const p3_4_ini= 0;  // Initial value of Process 3 of Process 4
const p4_1_ini= 0;  // Initial value of Process 4 of Process 1
const p4_2_ini= 0;  // Initial value of Process 4 of Process 2
const p4_3_ini= 0;  // Initial value of Process 4 of Process 3
const p4_4_ini= 1;  // Initial value of Process 4 of Process 4

const not_last=1;   // Auxiliary constant to define the next process
const last=0;       // Auxiliary constant to define the next process

global p1_1_ext : [0..1] init 1;    // Message value of Process 1 of Process 1
global p1_2_ext : [0..1] init 0;    // Message value of Process 1 of Process 2
global p1_3_ext : [0..1] init 0;    // Message value of Process 1 of Process 3
global p1_4_ext : [0..1] init 0;    // Message value of Process 1 of Process 3

global p2_1_ext : [0..1] init 0;    // Message value of Process 2 of Process 1
global p2_2_ext : [0..1] init 1;    // Message value of Process 2 of Process 2
global p2_3_ext : [0..1] init 0;    // Message value of Process 2 of Process 3
global p2_4_ext : [0..1] init 0;    // Message value of Process 2 of Process 4

global p3_1_ext : [0..1] init 0;    // Message value of Process 3 of Process 1
global p3_2_ext : [0..1] init 0;    // Message value of Process 3 of Process 2
global p3_3_ext : [0..1] init 1;    // Message value of Process 3 of Process 3
global p3_4_ext : [0..1] init 0;    // Message value of Process 3 of Process 4

global p4_1_ext : [0..1] init 0;    // Message value of Process 3 of Process 1
global p4_2_ext : [0..1] init 0;    // Message value of Process 3 of Process 2
global p4_3_ext : [0..1] init 0;    // Message value of Process 3 of Process 3
global p4_4_ext : [0..1] init 1;    // Message value of Process 3 of Process 4
global token : [1..N] init 1;       // Token used to coordinate the processes
```

Listing A.28: *Prism Model for a system of four processes executing the group formation algorithm, part (1)*

A.4.3 n=5

```
55
56 formula next = N1*not_last+1;       // Define the next process to receive the token
57 formula p1_1_new = max (p1_1,(p2_1_ext * n1_nf2),(p3_1_ext * n1_nf3),(p4_1_ext * n1_nf4)); //
       Process 1 compute new value of Process 1
58 formula p1_2_new = max (p1_2,(p2_2_ext * n1_nf2),(p3_2_ext * n1_nf3),(p4_2_ext * n1_nf4)); //
       Process 1 compute new value of Process 2
59 formula p1_3_new = max (p1_3,(p2_3_ext * n1_nf2),(p3_3_ext * n1_nf3),(p4_3_ext * n1_nf4)); //
       Process 1 compute new value of Process 3
60 formula p1_4_new = max (p1_4,(p2_4_ext * n1_nf2),(p3_4_ext * n1_nf3),(p4_4_ext * n1_nf4)); //
       Process 1 compute new value of Process 4
61
62 formula m1_new= p1_1 + p1_2 + p1_3 + p1_4; // length of view vector of process 1
63 formula m2_new= p2_1 + p2_2 + p2_3 + p2_4; // length of view vector of process 2
64 formula m3_new= p3_1 + p3_2 + p3_3 + p3_4; // length of view vector of process 2
65 formula m4_new= p4_1 + p4_2 + p4_3 + p4_4; // length of view vector of process 2
66
67 // Label definitions for propery specifications
68 label "p1_AB" = (p1_d1=0) &(p1_d2=0) &(p1_d3=0) &(p1_d4=0) ; //Process p1 aborts
69 label "p2_AB" = (p2_d1=0) &(p2_d2=0) &(p2_d3=0) &(p2_d4=0) ; //Process p2 aborts
70 label "p3_AB" = (p3_d1=0) &(p3_d2=0) &(p3_d3=0) &(p3_d4=0) ; //Process p3 aborts
71 label "p4_AB" = (p4_d1=0) &(p4_d2=0) &(p4_d3=0) &(p4_d4=0) ; //Process p4 aborts
72 label "p1_DC" = (p1_d1=1) |(p1_d2=1) |(p1_d3=1) |(p1_d4=1) ; //Process p1 decides
73 label "p2_DC" = (p2_d1=1) |(p2_d2=1) |(p2_d3=1) |(p2_d4=1) ; //Process p2 decides
74 label "p3_DC" = (p3_d1=1) |(p3_d2=1) |(p3_d3=1) |(p3_d4=1) ; //Process p3 decides
75 label "p4_DC" = (p4_d1=1) |(p4_d2=1) |(p4_d3=1) |(p4_d4=1) ; //Process p4 decides
76 label "p1_NEQ_p2" =(p1_d1!=p2_d1) |(p1_d2!=p2_d2) |(p1_d3!=p2_d3) |(p1_d4!=p2_d4) ; //Process p1
       is not equal to p2
77 label "p1_NEQ_p3" =(p1_d1!=p3_d1) |(p1_d2!=p3_d2) |(p1_d3!=p3_d3) |(p1_d4!=p3_d4) ; //Process p1
       is not equal to p3
78 label "p1_NEQ_p4" =(p1_d1!=p4_d1) |(p1_d2!=p4_d2) |(p1_d3!=p4_d3) |(p1_d4!=p4_d4) ; //Process p1
       is not equal to p4
79 label "p2_NEQ_p3" =(p2_d1!=p3_d1) |(p2_d2!=p3_d2) |(p2_d3!=p3_d3) |(p2_d4!=p3_d4) ; //Process p2
       is not equal to p3
80 label "p2_NEQ_p4" =(p2_d1!=p4_d1) |(p2_d2!=p4_d2) |(p2_d3!=p4_d3) |(p2_d4!=p4_d4) ; //Process p2
       is not equal to p4
81 label "p3_NEQ_p4" =(p3_d1!=p4_d1) |(p3_d2!=p4_d2) |(p3_d3!=p4_d3) |(p3_d4!=p4_d4) ; //Process p3
       is not equal to p4
82 label "p1_EQ_p2" =(p1_d1=p2_d1) & (p1_d2=p2_d2) & (p1_d3=p2_d3) & (p1_d4=p2_d4) ; //Process p1 is
       equal to p2
83 label "p1_EQ_p3" =(p1_d1=p3_d1) &(p1_d2=p3_d2) &(p1_d3=p3_d3) & (p1_d4=p3_d4) ; //Process p1 is
       equal to p3
84 label "p1_EQ_p4" =(p1_d1=p4_d1) &(p1_d2=p4_d2) &(p1_d3=p4_d3) & (p1_d4=p4_d4) ; //Process p1 is
       equal to p4
85 label "p2_EQ_p3" =(p2_d1=p3_d1) &(p2_d2=p3_d2) &(p2_d3=p3_d3) & (p2_d4=p3_d4) ; //Process p2 is
       equal to p3
86 label "p2_EQ_p4" =(p2_d1=p4_d1) &(p2_d2=p4_d2) &(p2_d3=p4_d3) & (p2_d4=p4_d4) ; //Process p2 is
       equal to p4
87 label "p3_EQ_p4" =(p3_d1=p4_d1) &(p3_d2=p4_d2) &(p3_d3=p4_d3) & (p3_d4=p4_d4) ; //Process p3 is
       equal to p4
```

Listing A.29: *Prism Model for a system of four processes executing the group formation algorithm, part (2)*

```
module Process_1
s1 : [0..N+3] init 1;   // Process 1 current state
RN1 : [0..RN] init 0;   // Current round
m1 : [0..N] init 1;

p1_d1 : [0..1] init 0;  // Process 1 decision of process 1
p1_d2 : [0..1] init 0;  // Process 1 decision of process 2
p1_d3 : [0..1] init 0;  // Process 1 decision of process 3
p1_d4 : [0..1] init 0;  // Process 1 decision of process 4

p1_1 : [0..1] init p1_1_ini;    // Process 1 value of process 1
p1_2 : [0..1] init p1_2_ini;    // Process 1 value of process 2
p1_3 : [0..1] init p1_3_ini;    // Process 1 value of process 3
p1_4 : [0..1] init p1_4_ini;    // Process 1 value of process 4

// Status of the message from the other processs
n1_nf2 :[0..1] init 0;   // Process 1 has not received the message of Process 2
n1_nf3 :[0..1] init 0;   // Process 1 has not received the message of Process 3
n1_nf4 :[0..1] init 0;   // Process 1 has not received the message of Process 4

// Process 1 sends or loses its message;
[] s1=1 & token=N1 & RN1<RN -> 1:(s1'=2) & (token'=next) & (p1_1_ext'=p1_1) & (p1_2_ext'=p1_2) &
      (p1_3_ext'=p1_3) & (p1_4_ext'=p1_4) & (RN1'=RN1+1);
// Process 1 receives or loses the message of Process 2,3,4
[] s1=2 & token=N1 & RN1<=RN -> (1-Q): (s1'=3) & (n1_nf2'=1) + Q: (s1'=3) & (n1_nf2'=0);
[] s1=3 & token=N1 & RN1<=RN -> (1-Q): (s1'=4) & (n1_nf3'=1) + Q: (s1'=4) & (n1_nf3'=0);
[] s1=4 & token=N1 & RN1<=RN -> (1-Q): (s1'=5) & (n1_nf4'=1) + Q: (s1'=5) & (n1_nf4'=0);
// Not last round, Process 1 computes thye messages of other processes: updates its values;
[] s1=N+1 & token=N1 & (RN1<RN) -> 1: (s1'=1) & (p1_1'=p1_1_new) & (p1_2'=p1_2_new) &
      (p1_3'=p1_3_new) & (p1_4'=p1_4_new) & (token'=next);
// Last round, Process 1 computes the messages of other processes:  updates its values;
//[] s1=N+1 & token=N1 & RN1=RN -> 1: (s1'=N+2) & (p1_1'=p1_1_new) & (p1_2'=p1_2_new) &
      (p1_3'=p1_3_new) & (m1'=m1_new)& (orc1'=orc1_new) & (token'=next);
[] s1=N+1 & token=N1 & RN1=RN -> 1: (s1'=N+2) & (p1_1'=p1_1_new) & (p1_2'=p1_2_new) &
      (p1_3'=p1_3_new)  & (p1_4'=p1_4_new) & (token'=next);
[] s1=N+2 & token=N1 & RN1=RN -> 1: (s1'=N+3) & (m1'=m1_new) & (token'=next);
// Process 1 decides -> agree or abort
[] s1=N+3 & token=N1 & (m1 >= (c*o1))-> (s1'=0) & (token'=next) & (p1_d1'=p1_1) & (p1_d2'= p1_2)&
      (p1_d3'= p1_3)& (p1_d4'= p1_4);
[] s1=N+3 & token=N1 & (m1 < (c*o1))-> (s1'=0) & (token'=next) & (p1_d1'=0) & (p1_d2'= 0) &
      (p1_d3'= 0)& (p1_d4'= 0);
endmodule
module Process_2 = Process_1 [N1=N2, s1=s2, o1=o2, m1=m2, p1_d1=p2_d2, p1_d2=p2_d3, p1_d3=p2_d4,
      p1_d4=p2_d1, n1_nf2=n2_nf3, n1_nf3=n2_nf4, n1_nf4=n2_nf1, p1_1=p2_2, p1_2=p2_3, p1_3=p2_4,
      p1_4=p2_1, p1_1_ini=p2_2_ini, p1_2_ini=p2_3_ini, p1_3_ini=p2_4_ini, p1_4_ini=p2_1_ini,
      p1_1_ext=p2_2_ext, p1_2_ext=p2_3_ext, p1_3_ext=p2_4_ext, p1_4_ext=p2_1_ext,
      p2_1_ext=p3_2_ext, p2_2_ext=p3_3_ext, p2_3_ext=p3_4_ext, p2_4_ext=p3_1_ext,
      p3_1_ext=p4_2_ext, p3_2_ext=p4_3_ext, p3_3_ext=p4_4_ext, p3_4_ext=p4_1_ext,
      p4_1_ext=p1_2_ext, p4_2_ext=p1_3_ext, p4_3_ext=p1_4_ext, p4_4_ext=p1_1_ext, RN1=RN2]
      endmodule
module Process_3 = Process_1 [N1=N3, s1=s3, o1=o3, m1=m3, p1_d1=p3_d3, p1_d2=p3_d4, p1_d3=p3_d1,
      p1_d4=p3_d2, n1_nf2=n3_nf4, n1_nf3=n3_nf1, n1_nf4=n3_nf2, p1_1=p3_3, p1_2=p3_4, p1_3=p3_1,
      p1_4=p3_2, p1_1_ini=p3_3_ini, p1_2_ini=p3_4_ini, p1_3_ini=p3_1_ini, p1_4_ini=p3_2_ini,
      p1_1_ext=p3_3_ext, p1_2_ext=p3_4_ext, p1_3_ext=p3_1_ext, p1_4_ext=p3_2_ext,
      p2_1_ext=p4_3_ext, p2_2_ext=p4_4_ext, p2_3_ext=p4_1_ext, p2_4_ext=p4_2_ext,
      p3_1_ext=p1_3_ext, p3_2_ext=p1_4_ext, p3_3_ext=p1_1_ext, p3_4_ext=p1_2_ext,
      p4_1_ext=p2_3_ext, p4_2_ext=p2_4_ext, p4_3_ext=p2_1_ext, p4_4_ext=p2_2_ext, RN1=RN3]
      endmodule
module Process_4 = Process_1 [N1=N4, s1=s4, o1=o4, m1=m4, p1_d1=p4_d4, p1_d2=p4_d1, p1_d3=p4_d2,
      p1_d4=p4_d3, n1_nf2=n4_nf1, n1_nf3=n4_nf2, n1_nf4=n4_nf3, p1_1=p4_4, p1_2=p4_1, p1_3=p4_2,
      p1_4=p4_3, p1_1_ini=p4_4_ini, p1_2_ini=p4_1_ini, p1_3_ini=p4_2_ini, p1_4_ini=p4_3_ini,
      p1_1_ext=p4_4_ext, p1_2_ext=p4_1_ext, p1_3_ext=p4_2_ext, p1_4_ext=p4_3_ext,
      p2_1_ext=p1_4_ext, p2_2_ext=p1_1_ext, p2_3_ext=p1_2_ext, p2_4_ext=p1_3_ext,
      p3_1_ext=p2_4_ext, p3_2_ext=p2_1_ext, p3_3_ext=p2_2_ext, p3_4_ext=p2_3_ext,
      p4_1_ext=p3_4_ext, p4_2_ext=p3_1_ext, p4_3_ext=p3_2_ext, p4_4_ext=p3_3_ext,
      RN1=RN4,not_last=last] endmodule
```

Listing A.30: *Prism Model for a system of four processes executing the group formation algorithm, part (3)*

```
dtmc
const N=5;      // Number of processes in the network (N cannot be modified)
const RN=2;         // Number of rounds in the protocol (RN>=2)
const double Q;     // Probability of losing a message (0<=q<=1)
const double c;
const N1=1;         // Identity number of Process 1
const N2=2;         // Identity number of Process 2
const N3=3;         // Identity number of Process 3
const N4=4;         // Identity number of Process 4
const N5=5;         // Identity number of Process 5
const o1;       //Oracle value of process 1
const o2;       //Oracle value of process 2
const o3;       //Oracle value of process 3
const o4;       //Oracle value of process 4
const o5;       //Oracle value of process 5

const p1_1_ini= 1;  // Initial value of Process 1 of Process 1
const p1_2_ini= 0;  // Initial value of Process 1 of Process 2
const p1_3_ini= 0;  // Initial value of Process 1 of Process 3
const p1_4_ini= 0;  // Initial value of Process 1 of Process 4
const p1_5_ini= 0;  // Initial value of Process 1 of Process 5
const p2_1_ini= 0;  // Initial value of Process 2 of Process 1
const p2_2_ini= 1;  // Initial value of Process 2 of Process 2
const p2_3_ini= 0;  // Initial value of Process 2 of Process 3
const p2_4_ini= 0;  // Initial value of Process 2 of Process 4
const p2_5_ini= 0;  // Initial value of Process 2 of Process 5
const p3_1_ini= 0;  // Initial value of Process 3 of Process 1
const p3_2_ini= 0;  // Initial value of Process 3 of Process 2
const p3_3_ini= 1;  // Initial value of Process 3 of Process 3
const p3_4_ini= 0;  // Initial value of Process 3 of Process 4
const p3_5_ini= 0;  // Initial value of Process 3 of Process 5
const p4_1_ini= 0;  // Initial value of Process 4 of Process 1
const p4_2_ini= 0;  // Initial value of Process 4 of Process 2
const p4_3_ini= 0;  // Initial value of Process 4 of Process 3
const p4_4_ini= 1;  // Initial value of Process 4 of Process 4
const p4_5_ini= 0;  // Initial value of Process 4 of Process 5
const p5_1_ini= 0;  // Initial value of Process 4 of Process 1
const p5_2_ini= 0;  // Initial value of Process 4 of Process 2
const p5_3_ini= 0;  // Initial value of Process 4 of Process 3
const p5_4_ini= 0;  // Initial value of Process 4 of Process 4
const p5_5_ini= 1;  // Initial value of Process 4 of Process 5
const not_last=1;   // Auxiliary constant to define the next process
const last=0;       // Auxiliary constant to define the next process

global p1_1_ext : [0..1] init 1;    // Message value of Process 1 of Process 1
global p1_2_ext : [0..1] init 0;    // Message value of Process 1 of Process 2
global p1_3_ext : [0..1] init 0;    // Message value of Process 1 of Process 3
global p1_4_ext : [0..1] init 0;    // Message value of Process 1 of Process 4
global p1_5_ext : [0..1] init 0;    // Message value of Process 1 of Process 5

global p2_1_ext : [0..1] init 0;    // Message value of Process 2 of Process 1
global p2_2_ext : [0..1] init 1;    // Message value of Process 2 of Process 2
global p2_3_ext : [0..1] init 0;    // Message value of Process 2 of Process 3
global p2_4_ext : [0..1] init 0;    // Message value of Process 2 of Process 4
global p2_5_ext : [0..1] init 0;    // Message value of Process 2 of Process 5

global p3_1_ext : [0..1] init 0;    // Message value of Process 3 of Process 1
global p3_2_ext : [0..1] init 0;    // Message value of Process 3 of Process 2
global p3_3_ext : [0..1] init 1;    // Message value of Process 3 of Process 3
global p3_4_ext : [0..1] init 0;    // Message value of Process 3 of Process 4
global p3_5_ext : [0..1] init 0;    // Message value of Process 3 of Process 5

global p4_1_ext : [0..1] init 0;    // Message value of Process 3 of Process 1
global p4_2_ext : [0..1] init 0;    // Message value of Process 3 of Process 2
global p4_3_ext : [0..1] init 0;    // Message value of Process 3 of Process 3
global p4_4_ext : [0..1] init 1;    // Message value of Process 3 of Process 4
global p4_5_ext : [0..1] init 0;    // Message value of Process 3 of Process 5

global p5_1_ext : [0..1] init 0;    // Message value of Process 3 of Process 1
global p5_2_ext : [0..1] init 0;    // Message value of Process 3 of Process 2
global p5_3_ext : [0..1] init 0;    // Message value of Process 3 of Process 3
global p5_4_ext : [0..1] init 0;    // Message value of Process 3 of Process 4
global p5_5_ext : [0..1] init 1;    // Message value of Process 3 of Process 5
global token : [1..N] init 1;       // Token used to coordinate the processes
```

Listing A.31: *Prism Model for a system of five processes executing the group formation algorithm, part (1)*

```
75
76  formula next = N1*not_last+1;       // Define the next process to receive the token
77  formula p1_1_new = max (p1_1,(p2_1_ext * n1_nf2),(p3_1_ext * n1_nf3),(p4_1_ext * n1_nf4),(p5_1_ext
        * n1_nf5)); // Process 1 compute new value of Process 1
78  formula p1_2_new = max (p1_2,(p2_2_ext * n1_nf2),(p3_2_ext * n1_nf3),(p4_2_ext * n1_nf4),(p5_2_ext
        * n1_nf5)); // Process 1 compute new value of Process 2
79  formula p1_3_new = max (p1_3,(p2_3_ext * n1_nf2),(p3_3_ext * n1_nf3),(p4_3_ext * n1_nf4),(p5_3_ext
        * n1_nf5)); // Process 1 compute new value of Process 3
80  formula p1_4_new = max (p1_4,(p2_4_ext * n1_nf2),(p3_4_ext * n1_nf3),(p4_4_ext * n1_nf4),(p5_4_ext
        * n1_nf5)); // Process 1 compute new value of Process 4
81  formula p1_5_new = max (p1_5,(p2_5_ext * n1_nf2),(p3_5_ext * n1_nf3),(p4_5_ext * n1_nf4),(p5_5_ext
        * n1_nf5)); // Process 1 compute new value of Process 5
82
83  formula m1_new= p1_1 + p1_2 + p1_3 + p1_4 + p1_5; // length of view vector of process 1
84  formula m2_new= p2_1 + p2_2 + p2_3 + p2_4 + p2_5; // length of view vector of process 2
85  formula m3_new= p3_1 + p3_2 + p3_3 + p3_4 + p3_5; // length of view vector of process 3
86  formula m4_new= p4_1 + p4_2 + p4_3 + p4_4 + p4_5; // length of view vector of process 4
87  formula m5_new= p5_1 + p5_2 + p5_3 + p5_4 + p5_5; // length of view vector of process 5
88
89  // Label definitions for propery specifications
90  label "p1_AB" = (p1_d1=0) &(p1_d2=0) &(p1_d3=0) &(p1_d4=0) &(p1_d5=0) ; //Process p1 aborts
91  label "p2_AB" = (p2_d1=0) &(p2_d2=0) &(p2_d3=0) &(p2_d4=0) &(p2_d5=0) ; //Process p2 aborts
92  label "p3_AB" = (p3_d1=0) &(p3_d2=0) &(p3_d3=0) &(p3_d4=0) &(p3_d5=0) ; //Process p3 aborts
93  label "p4_AB" = (p4_d1=0) &(p4_d2=0) &(p4_d3=0) &(p4_d4=0) &(p4_d5=0) ; //Process p4 aborts
94  label "p5_AB" = (p5_d1=0) &(p5_d2=0) &(p5_d3=0) &(p5_d4=0) &(p5_d5=0) ; //Process p5 aborts
95  label "p1_DC" = (p1_d1=1) |(p1_d2=1) |(p1_d3=1) |(p1_d4=1) |(p1_d5=1) ; //Process p1 decides
96  label "p2_DC" = (p2_d1=1) |(p2_d2=1) |(p2_d3=1) |(p2_d4=1) |(p2_d5=1) ; //Process p2 decides
97  label "p3_DC" = (p3_d1=1) |(p3_d2=1) |(p3_d3=1) |(p3_d4=1) |(p3_d5=1) ; //Process p3 decides
98  label "p4_DC" = (p4_d1=1) |(p4_d2=1) |(p4_d3=1) |(p4_d4=1) |(p4_d5=1) ; //Process p4 decides
99  label "p5_DC" = (p5_d1=1) |(p5_d2=1) |(p5_d3=1) |(p5_d4=1) |(p5_d5=1) ; //Process p5 decides
100
101 label "p1_NEQ_p2" =(p1_d1!=p2_d1) |(p1_d2!=p2_d2) |(p1_d3!=p2_d3) |(p1_d4!=p2_d4)|(p1_d5!=p2_d5) ;
        //Process p1 is not equal to p2
102 label "p1_NEQ_p3" =(p1_d1!=p3_d1) |(p1_d2!=p3_d2) |(p1_d3!=p3_d3) |(p1_d4!=p3_d4)|(p1_d5!=p3_d5) ;
        //Process p1 is not equal to p3
103 label "p1_NEQ_p4" =(p1_d1!=p4_d1) |(p1_d2!=p4_d2) |(p1_d3!=p4_d3) |(p1_d4!=p4_d4)|(p1_d5!=p4_d5) ;
        //Process p1 is not equal to p4
104 label "p1_NEQ_p5" =(p1_d1!=p5_d1) |(p1_d2!=p5_d2) |(p1_d3!=p5_d3) |(p1_d4!=p5_d4)|(p1_d5!=p5_d5) ;
        //Process p1 is not equal to p5
105 label "p2_NEQ_p3" =(p2_d1!=p3_d1) |(p2_d2!=p3_d2) |(p2_d3!=p3_d3) |(p2_d4!=p3_d4)|(p2_d5!=p3_d5) ;
        //Process p2 is not equal to p3
106 label "p2_NEQ_p4" =(p2_d1!=p4_d1) |(p2_d2!=p4_d2) |(p2_d3!=p4_d3) |(p2_d4!=p4_d4)|(p2_d5!=p4_d5);
        //Process p2 is not equal to p4
107 label "p2_NEQ_p5" =(p2_d1!=p5_d1) |(p2_d2!=p5_d2) |(p2_d3!=p5_d3) |(p2_d4!=p5_d4)|(p2_d5!=p5_d5);
        //Process p2 is not equal to p5
108 label "p3_NEQ_p4" =(p3_d1!=p4_d1) |(p3_d2!=p4_d2) |(p3_d3!=p4_d3) |(p3_d4!=p4_d4)|(p3_d5!=p4_d5) ;
        //Process p3 is not equal to p4
109 label "p3_NEQ_p5" =(p3_d1!=p5_d1) |(p3_d2!=p5_d2) |(p3_d3!=p5_d3) |(p3_d4!=p5_d4)|(p3_d5!=p5_d5) ;
        //Process p3 is not equal to p5
110 label "p4_NEQ_p5" =(p4_d1!=p5_d1) |(p4_d2!=p5_d2) |(p4_d3!=p5_d3) |(p4_d4!=p5_d4)|(p4_d5!=p5_d5) ;
        //Process p4 is not equal to p5
111
112 label "p1_EQ_p2" =(p1_d1=p2_d1) &(p1_d2=p2_d2) &(p1_d3=p2_d3) &(p1_d4=p2_d4)&(p1_d5=p2_d5) ;
        //Process p1 is equal to p2
113 label "p1_EQ_p3" =(p1_d1=p3_d1) &(p1_d2=p3_d2) &(p1_d3=p3_d3) &(p1_d4=p3_d4)&(p1_d5=p3_d5) ;
        //Process p1 is equal to p3
114 label "p1_EQ_p4" =(p1_d1=p4_d1) &(p1_d2=p4_d2) &(p1_d3=p4_d3) &(p1_d4=p4_d4)&(p1_d5=p4_d5) ;
        //Process p1 is equal to p4
115 label "p1_EQ_p5" =(p1_d1=p5_d1) &(p1_d2=p5_d2) &(p1_d3=p5_d3) &(p1_d4=p5_d4)&(p1_d5=p5_d5) ;
        //Process p1 is equal to p5
116 label "p2_EQ_p3" =(p2_d1=p3_d1) &(p2_d2=p3_d2) &(p2_d3=p3_d3) &(p2_d4=p3_d4)&(p2_d5=p3_d5) ;
        //Process p2 is equal to p3
117 label "p2_EQ_p4" =(p2_d1=p4_d1) &(p2_d2=p4_d2) &(p2_d3=p4_d3) &(p2_d4=p4_d4)&(p2_d5=p4_d5);
        //Process p2 is equal to p4
118 label "p2_EQ_p5" =(p2_d1=p5_d1) &(p2_d2=p5_d2) &(p2_d3=p5_d3) &(p2_d4=p5_d4)&(p2_d5=p5_d5);
        //Process p2 is equal to p5
119 label "p3_EQ_p4" =(p3_d1=p4_d1) &(p3_d2=p4_d2) &(p3_d3=p4_d3) &(p3_d4=p4_d4)&(p3_d5=p4_d5) ;
        //Process p3 is equal to p4
120 label "p3_EQ_p5" =(p3_d1=p5_d1) &(p3_d2=p5_d2) &(p3_d3=p5_d3) &(p3_d4=p5_d4)&(p3_d5=p5_d5) ;
        //Process p3 is equal to p5
121 label "p4_EQ_p5" =(p4_d1=p5_d1) &(p4_d2=p5_d2) &(p4_d3=p5_d3) &(p4_d4=p5_d4)&(p4_d5=p5_d5) ;
        //Process p4 is equal to p5
```

Listing A.32: *Prism Model for a system of five processes executing the group formation algorithm, part (2)*

```
123 module Process_1
124 s1 : [0..N+3] init 1;   // Process 1 current state
125 RN1: [0..RN] init 0;    // Current round
126 m1 : [0..N] init 1;
127
128 p1_d1 : [0..1] init 0;  // Process 1 decision of process 1
129 p1_d2 : [0..1] init 0;  // Process 1 decision of process 2
130 p1_d3 : [0..1] init 0;  // Process 1 decision of process 3
131 p1_d4 : [0..1] init 0;  // Process 1 decision of process 4
132 p1_d5 : [0..1] init 0;  // Process 1 decision of process 5
133
134 p1_1 : [0..1] init p1_1_ini;    // Process 1 value of process 1
135 p1_2 : [0..1] init p1_2_ini;    // Process 1 value of process 2
136 p1_3 : [0..1] init p1_3_ini;    // Process 1 value of process 3
137 p1_4 : [0..1] init p1_4_ini;    // Process 1 value of process 4
138 p1_5 : [0..1] init p1_5_ini;    // Process 1 value of process 5
139
140 // Status of the message from the other processs
141 n1_nf2 :[0..1] init 0;   // Process 1 has not received the message of Process 2
142 n1_nf3 :[0..1] init 0;   // Process 1 has not received the message of Process 3
143 n1_nf4 :[0..1] init 0;   // Process 1 has not received the message of Process 4
144 n1_nf5 :[0..1] init 0;   // Process 1 has not received the message of Process 5
145
146 // Process 1 sends or loses its message;
147 [] s1=1 & token=N1 & RN1<RN -> 1:(s1'=2) & (token'=next) & (p1_1_ext'=p1_1) & (p1_2_ext'=p1_2) &
        (p1_3_ext'=p1_3) & (p1_4_ext'=p1_4) & (p1_5_ext'=p1_5)& (RN1'=RN1+1);
148 // Process 1 receives or loses the message of Process 2,3,4
149 [] s1=2 & token=N1 & RN1<=RN -> (1-Q): (s1'=3) & (n1_nf2'=1) + Q: (s1'=3) & (n1_nf2'=0);
150 [] s1=3 & token=N1 & RN1<=RN -> (1-Q): (s1'=4) & (n1_nf3'=1) + Q: (s1'=4) & (n1_nf3'=0);
151 [] s1=4 & token=N1 & RN1<=RN -> (1-Q): (s1'=5) & (n1_nf4'=1) + Q: (s1'=5) & (n1_nf4'=0);
152 [] s1=5 & token=N1 & RN1<=RN -> (1-Q): (s1'=6) & (n1_nf5'=1) + Q: (s1'=6) & (n1_nf5'=0);
153 // Not last round, Process 1 computes the messages of other processes: updates its values;
154 [] s1=N+1 & token=N1 & (RN1<RN) -> 1: (s1'=1) & (p1_1'=p1_1_new) & (p1_2'=p1_2_new) &
        (p1_3'=p1_3_new) & (p1_4'=p1_4_new) & (p1_5'=p1_5_new) & (token'=next);
155 // Last round, Process 1 computes the messages of other processes:  updates its values;
156 //[] s1=N+1 & token=N1 & RN1=RN -> 1: (s1'=N+2) & (p1_1'=p1_1_new) & (p1_2'=p1_2_new) &
        (p1_3'=p1_3_new) & (m1'=m1_new)& (orc1'=orc1_new) & (token'=next);
157 [] s1=N+1 & token=N1 & RN1=RN -> 1: (s1'=N+2) & (p1_1'=p1_1_new) & (p1_2'=p1_2_new) &
        (p1_3'=p1_3_new) & (p1_4'=p1_4_new) & (p1_5'=p1_5_new) & (m1'=m1_new) & (token'=next);
158 //[] s1=N+2 & token=N1 & RN1=RN -> 1: (s1'=N+3) & (m1'=m1_new) & (o1'=floor(m1_new/2)+2) &
        (token'=next); ,
159 [] s1=N+2 & token=N1 & RN1=RN -> 1: (s1'=N+3) & (m1'=m1_new) & (token'=next);
160 // Process 1 decides -> agree or abort
161 //Local Oracle
162 //[] s1=N+3 & token=N1 & (m1 =(c*o1)) & m1=1-> (s1'=0) & (token'=next) & (p1_d1'=0) & (p1_d2'= 0)
        & (p1_d3'= 0)& (p1_d4'= 0)& (p1_d5'= 0);
163 //[] s1=N+3 & token=N1 & (m1 >= (c*o1)) & m1>1-> (s1'=0) & (token'=next) & (p1_d1'=p1_1) &
        (p1_d2'= p1_2)& (p1_d3'= p1_3)& (p1_d4'= p1_4)& (p1_d5'= p1_5);
164 [] s1=N+3 & token=N1 & (m1 >= (c*o1))-> (s1'=0) & (token'=next) & (p1_d1'=p1_1) & (p1_d2'= p1_2)&
        (p1_d3'= p1_3)& (p1_d4'= p1_4)& (p1_d5'= p1_5);
165 [] s1=N+3 & token=N1 & (m1 < (c*o1))-> (s1'=0) & (token'=next) & (p1_d1'=0) & (p1_d2'= 0) &
        (p1_d3'= 0)& (p1_d4'= 0)& (p1_d5'= 0);
166 endmodule
```

Listing A.33: *Prism Model for a system of five processes executing the group formation algorithm, part (3)*

```
167
168  module Process_2 = Process_1 [N1=N2, s1=s2, o1=o2, m1=m2, p1_d1=p2_d2, p1_d2=p2_d3, p1_d3=p2_d4,
         p1_d4=p2_d5, p1_d5=p2_d1, n1_nf2=n2_nf3, n1_nf3=n2_nf4, n1_nf4=n2_nf5, n1_nf5=n2_nf1,
         p1_1=p2_2, p1_2=p2_3, p1_3=p2_4, p1_4=p2_5, p1_5=p2_1, p1_1_ini=p2_2_ini,
         p1_2_ini=p2_3_ini, p1_3_ini=p2_4_ini, p1_4_ini=p2_5_ini, p1_5_ini=p2_1_ini,
         p1_1_ext=p2_2_ext, p1_2_ext=p2_3_ext, p1_3_ext=p2_4_ext, p1_4_ext=p2_5_ext,
         p1_5_ext=p2_1_ext, p2_1_ext=p3_2_ext, p2_2_ext=p3_3_ext, p2_3_ext=p3_4_ext,
         p2_4_ext=p3_5_ext, p2_5_ext=p3_1_ext, p3_1_ext=p4_2_ext, p3_2_ext=p4_3_ext,
         p3_3_ext=p4_4_ext, p3_4_ext=p4_5_ext, p3_5_ext=p4_1_ext, p4_1_ext=p5_2_ext,
         p4_2_ext=p5_3_ext, p4_3_ext=p5_4_ext, p4_4_ext=p5_5_ext, p4_5_ext=p5_1_ext,
         p5_1_ext=p1_2_ext, p5_2_ext=p1_3_ext, p5_3_ext=p1_4_ext, p5_4_ext=p1_5_ext,
         p5_5_ext=p1_1_ext, RN1=RN2] endmodule
169  module Process_3 = Process_1 [N1=N3, s1=s3, o1=o3, m1=m3, p1_d1=p3_d3, p1_d2=p3_d4, p1_d3=p3_d5,
         p1_d4=p3_d1, p1_d5=p3_d2, n1_nf2=n3_nf4, n1_nf3=n3_nf5, n1_nf4=n3_nf1, n1_nf5=n3_nf2,
         p1_1=p3_3, p1_2=p3_4, p1_3=p3_5, p1_4=p3_1, p1_5=p3_2, p1_1_ini=p3_3_ini,
         p1_2_ini=p3_4_ini, p1_3_ini=p3_5_ini, p1_4_ini=p3_1_ini, p1_5_ini=p3_2_ini,
         p1_1_ext=p3_3_ext, p1_2_ext=p3_4_ext, p1_3_ext=p3_5_ext, p1_4_ext=p3_1_ext,
         p1_5_ext=p3_2_ext, p2_1_ext=p4_3_ext, p2_2_ext=p4_4_ext, p2_3_ext=p4_5_ext,
         p2_4_ext=p4_1_ext, p2_5_ext=p4_2_ext, p3_1_ext=p5_3_ext, p3_2_ext=p5_4_ext,
         p3_3_ext=p5_5_ext, p3_4_ext=p5_1_ext, p3_5_ext=p5_2_ext, p4_1_ext=p1_3_ext,
         p4_2_ext=p1_4_ext, p4_3_ext=p1_5_ext, p4_4_ext=p1_1_ext, p4_5_ext=p1_2_ext,
         p5_1_ext=p2_3_ext, p5_2_ext=p2_4_ext, p5_3_ext=p2_5_ext, p5_4_ext=p2_1_ext,
         p5_5_ext=p2_2_ext, RN1=RN3] endmodule
170  module Process_4 = Process_1 [N1=N4, s1=s4, o1=o4, m1=m4, p1_d1=p4_d4, p1_d2=p4_d5, p1_d3=p4_d1,
         p1_d4=p4_d2, p1_d5=p4_d3, n1_nf2=n4_nf5, n1_nf3=n4_nf1, n1_nf4=n4_nf2, n1_nf5=n4_nf3,
         p1_1=p4_4, p1_2=p4_5, p1_3=p4_1, p1_4=p4_2, p1_5=p4_3, p1_1_ini=p4_4_ini,
         p1_2_ini=p4_5_ini, p1_3_ini=p4_1_ini, p1_4_ini=p4_2_ini, p1_5_ini=p4_3_ini,
         p1_1_ext=p4_4_ext, p1_2_ext=p4_5_ext, p1_3_ext=p4_1_ext, p1_4_ext=p4_2_ext,
         p1_5_ext=p4_3_ext, p2_1_ext=p5_4_ext, p2_2_ext=p5_5_ext, p2_3_ext=p5_1_ext,
         p2_4_ext=p5_2_ext, p2_5_ext=p5_3_ext, p3_1_ext=p1_4_ext, p3_2_ext=p1_5_ext,
         p3_3_ext=p1_1_ext, p3_4_ext=p1_2_ext, p3_5_ext=p1_3_ext, p4_1_ext=p2_4_ext,
         p4_2_ext=p2_5_ext, p4_3_ext=p2_1_ext, p4_4_ext=p2_2_ext, p4_5_ext=p2_3_ext,
         p5_1_ext=p3_4_ext, p5_2_ext=p3_5_ext, p5_3_ext=p3_1_ext, p5_4_ext=p3_2_ext,
         p5_5_ext=p3_3_ext, RN1=RN4] endmodule
171  module Process_5 = Process_1 [N1=N5, s1=s5, o1=o5, m1=m5, p1_d1=p5_d5, p1_d2=p5_d1, p1_d3=p5_d2,
         p1_d4=p5_d3, p1_d5=p5_d4, n1_nf2=n5_nf1, n1_nf3=n5_nf2, n1_nf4=n5_nf3, n1_nf5=n5_nf4,
         p1_1=p5_5, p1_2=p5_1, p1_3=p5_2, p1_4=p5_3, p1_5=p5_4, p1_1_ini=p5_5_ini,
         p1_2_ini=p5_1_ini, p1_3_ini=p5_2_ini, p1_4_ini=p5_3_ini, p1_5_ini=p5_4_ini,
         p1_1_ext=p5_5_ext, p1_2_ext=p5_1_ext, p1_3_ext=p5_2_ext, p1_4_ext=p5_3_ext,
         p1_5_ext=p5_4_ext, p2_1_ext=p1_5_ext, p2_2_ext=p1_1_ext, p2_3_ext=p1_2_ext,
         p2_4_ext=p1_3_ext, p2_5_ext=p1_4_ext, p3_1_ext=p2_5_ext, p3_2_ext=p2_1_ext,
         p3_3_ext=p2_2_ext, p3_4_ext=p2_3_ext, p3_5_ext=p2_4_ext, p4_1_ext=p3_5_ext,
         p4_2_ext=p3_1_ext, p4_3_ext=p3_2_ext, p4_4_ext=p3_3_ext, p4_5_ext=p3_4_ext,
         p5_1_ext=p4_5_ext, p5_2_ext=p4_1_ext, p5_3_ext=p4_2_ext, p5_4_ext=p4_3_ext,
         p5_5_ext=p4_4_ext, RN1=RN5, not_last=last] endmodule
```

Listing A.34: *Prism Model for a system of five processes executing the group formation algorithm, part (3)*

A.5 PRISM Models for the *1-of-** Selection Algorithms

In the following, we present the PRISM models for the *1-of-** selection algorithms for systems of three and four participating processes, $n = 3$ and $n = 4$.

A.5.1 n=3, Optimistic

```
dtmc
const N=3;      // Number of processes in the network (N cannot be modified)
const RN=2;         // Number of rounds in the protocol (RN>=2)
const double Q;     // Probability of losing a message (0<=q<=1)
const double c;     //Crrection parametr
const N1=1;         // Identity number of Process 1
const N2=2;         // Identity number of Process 2
const N3=3;         // Identity number of Process 3
const v1_ini=1;     // Initial value of Process 1
const v2_ini=2;     // Initial value of Process 2
const v3_ini=3;     // Initial value of Process 3
const not_last=1;   // Auxiliary constant to define the next process
const last=0;       // Auxiliary constant to define the next process

const o1=3;     //Oracle value
const o2=3;     //Oracle value is given randomely
const o3=3;     //Oracle value is given randomely

global v1_ext : [0..N] init 0;  // Message value of Process 1
global v2_ext : [0..N] init 0;  // Message value of Process 2
global v3_ext : [0..N] init 0;  // Message value of Process 3

global w1_v2_ext : [0..1] init 0;   // Process 1 view of Process 2
global w1_v3_ext : [0..1] init 0;   // Process 1 view of Process 3

global w2_v1_ext : [0..1] init 0;   // Process 2 view of Process 1
global w2_v3_ext : [0..1] init 0;   // Process 2 view of Process 3

global w3_v1_ext : [0..1] init 0;   // Process 3 view of Process 1
global w3_v2_ext : [0..1] init 0;   // Process 3 view of Process 2

global token : [1..N] init 1;           // Token used to coordinate the processes
formula next = N1*not_last+1; // Define the next Process in the network

formula m1_new= 1 + w1_v2 + w1_v3; // length of view vector of process 1
formula m2_new= 1 + w2_v1 + w2_v3; // length of view vector of process 2
formula m3_new= 1 + w3_v1 + w3_v2; // length of view vector of process 2

formula v1_new = max (v1 , (n1_nf2 * v2_ext) , (n1_nf3 * v3_ext));  // Process 1 compute new value

formula w1_v2_new = max (w1_v2 , n1_nf2, n1_nf3 * w3_v2_ext); // Process 1 update its view of
      Process 2
formula w1_v3_new = max (w1_v3 , n1_nf3, n1_nf2 * w2_v3_ext); // Process 1 update its view of
      Process 3

// Label definitions for property specifications
```

Listing A.35: *Prism Model for a system of three processes executing the 1-of-* selection algorithm with the optimistic* decision criterion, part (1)*

```
45  //Abort
46  label "p1_AB" = (d1=0); //Process p1 aborts
47  label "p2_AB" = (d2=0); //Process p2 aborts
48  label "p3_AB" = (d3=0); //Process p3 aborts
49  //Decide
50  label "p1_DC" = (d1!=0); //Process p1 decides
51  label "p2_DC" = (d2!=0); //Process p2 decides
52  label "p3_DC" = (d3!=0); //Process p3 decides
53
54  label "p1_NEQ_p2" = (d1!=d2); // p1's decision is not equal to p2's decision
55  label "p1_NEQ_p3" = (d1!=d3); // p1's decision is not equal to p3's decision
56  label "p2_NEQ_p3" = (d2!=d3); // p2's decision is not equal to p3's decision
57
58  module Process_1
59  s1 : [0..N+3] init 1;  // Process 1 current state
60  RN1: [0..RN] init 0; // Current round
61  v1 : [0..N] init v1_ini; // Process 1 value
62  d1 : [0..N] init 0; // Process 1 decision
63  m1 : [1..N] init 1;
64
65  // Status of the message from the other processs
66  n1_nf2 :[0..1] init 0; // Process 1 has not received the message of Process 2
67  n1_nf3 :[0..1] init 0; // Process 1 has not received the message of Process 3
68  // Process 1 view of other processes
69  w1_v2 :[0..1] init 0; // Process 1 has the view of Process 2
70  w1_v3 :[0..1] init 0; // Process 1 has the view of Process 3
71
72  // Process 1 sends its message;
73  [] s1=1 & token=N1 & RN1<RN -> 1:(s1'=2) & (token'=next) & (v1_ext'=v1) & (w1_v2_ext'=w1_v2) &
        (w1_v3_ext'=w1_v3) & (RN1'=RN1+1);
74  // Process 1 receives or loses the message of Process 2
75  [] s1=2 & token=N1 & RN1<=RN -> (1-Q): (s1'=3) & (n1_nf2'=1) + Q: (s1'=3) & (n1_nf2'=0);
76  // Process 1 receives or loses the message of Process 3
77  [] s1=3 & token=N1 & RN1<=RN -> (1-Q): (s1'=4) & (n1_nf3'=1) + Q: (s1'=4) & (n1_nf3'=0);
78  // Not last round, Process 1 computes the messages of other processes: updates its value, views
        and confirmations;
79  [] s1=N+1 & token=N1 & RN1<RN -> 1: (s1'=1) & (v1'=v1_new) & (w1_v2'=w1_v2_new) &
        (w1_v3'=w1_v3_new) &  (token'=next);
80  // Last round, Process 1 computes the messages of other processes: updates its confirmations and,
        only for OP, updates value and views;
81  [] s1=N+1 & token=N1 & RN1=RN -> 1: (s1'=N+2) & (w1_v2'=w1_v2_new) & (w1_v3'=w1_v3_new) &
        (v1'=v1_new) & (token'=next);
82  [] s1=N+2 & token=N1 & RN1=RN -> 1: (s1'=N+3) &  (m1'=m1_new) & (token'=next);
83  // Process 1 decides -> agree or abort
84  [] s1=N+3 & token=N1 & (m1 >= (c* o1))-> 1: (s1'=0) & (token'=next) & (d1'=v1);
85  [] s1=N+3 & token=N1 & (m1 < (c*o1))-> 1: (s1'=0) & (token'=next) & (d1'=0);
86  endmodule
87  module Process_2=Process_1 [N1=N2, s1=s2, m1=m2, o1=o2, v1=v2, d1=d2, RN1=RN2, n1_nf2=n2_nf3,
        n1_nf3=n2_nf1, w1_v2=w2_v3, w1_v3=w2_v1, v1_ext=v2_ext, v2_ext=v3_ext, v3_ext=v1_ext,
        w1_v2_ext=w2_v3_ext, w1_v3_ext=w2_v1_ext, w2_v1_ext=w3_v2_ext, w2_v3_ext=w3_v1_ext,
        w3_v1_ext=w1_v2_ext, w3_v2_ext=w1_v3_ext, v1_ini=v2_ini] endmodule
88  module Process_3=Process_1 [N1=N3, s1=s3, m1=m3, o1=o3, v1=v3, d1=d3, RN1=RN3, n1_nf2=n3_nf1,
        n1_nf3=n3_nf2, w1_v2=w3_v1, w1_v3=w3_v2, v1_ext=v3_ext, v2_ext=v1_ext, v3_ext=v2_ext,
        w1_v2_ext=w3_v1_ext, w1_v3_ext=w3_v2_ext, w2_v1_ext=w1_v3_ext, w2_v3_ext=w1_v2_ext,
        w3_v1_ext=w2_v3_ext, w3_v2_ext=w2_v1_ext, v1_ini=v3_ini, not_last=last] endmodule
```

Listing A.36: *Prism Model for a system of three processes executing the 1-of-* selection algorithm with the optimistic* decision criterion, part (2)*

A.5.2 n=3, Moderately pessimistic

A.5.3 n=3, Pessimistic

A.5.4 n=4, Optimistic

```
dtmc
const N=3;      // Number of processes in the network (N cannot be modified)
const RN=2;         // Number of rounds in the protocol (RN>=2)
const double Q;     // Probability of losing a message (0<=q<=1)
const double c;     //Crrection parametr
const N1=1;         // Identity number of Process 1
const N2=2;         // Identity number of Process 2
const N3=3;         // Identity number of Process 3
const v_max=3;      // Maximum value of a process
const v1_ini=1;     // Initial value of Process 1
const v2_ini=2;     // Initial value of Process 2
const v3_ini=3;     // Initial value of Process 3
const not_last=1;   // Auxiliary constant to define the next process
const last=0;       // Auxiliary constant to define the next process

const o1=3;     //Oracle value is given randomely
const o2=3;     //Oracle value is given randomely
const o3=3;     //Oracle value is given randomely

global v1_ext : [0..N] init 0;  // Message value of Process 1
global v2_ext : [0..N] init 0;  // Message value of Process 2
global v3_ext : [0..N] init 0;  // Message value of Process 3

global w1_v2_ext : [0..1] init 0;   // Process 1 view of Process 2
global w1_v3_ext : [0..1] init 0;   // Process 1 view of Process 3

global w2_v1_ext : [0..1] init 0;   // Process 2 view of Process 1
global w2_v3_ext : [0..1] init 0;   // Process 2 view of Process 3

global w3_v1_ext : [0..1] init 0;   // Process 3 view of Process 1
global w3_v2_ext : [0..1] init 0;   // Process 3 view of Process 2

global token : [1..N] init 1;           // Token used to coordinate the processes

formula next = N1*not_last+1; // Define the next Process in the network

formula m1_new= 1 + w1_v2 + w1_v3; // length of view vector of process 1
formula m2_new= 1 + w2_v1 + w2_v3; // length of view vector of process 2
formula m3_new= 1 + w3_v1 + w3_v2; // length of view vector of process 3

//formula c1_new= 1 + w1_c2 + w1_c3; // length of complete view vector of process 1
//formula c2_new= 1 + w2_c1 + w2_c3; // length of complete view vector of process 2
//formula c3_new= 1 + w3_c1 + w3_c2; // length of complete view vector of process 2

formula c1_new=  1 + w1_c2 + w1_c3; // length of complete view vector of process 1
formula c2_new=  1 + w2_c1 + w2_c3; // length of complete view vector of process 2
formula c3_new=  1 + w3_c1 + w3_c2; // length of complete view vector of process 2

formula v1_new = max (v1 , (n1_nf2 * v2_ext),(n1_nf3 * v3_ext));  // Process 1 compute new value
formula w1_v2_new = max (w1_v2 , n1_nf2, n1_nf3 * w3_v2_ext); // Process 1 update its view of
      Process 2
formula w1_v3_new = max (w1_v3 , n1_nf3, n1_nf2 * w2_v3_ext); // Process 1 update its view of
      Process 3

// Process 1 knows that Process 2 view is complete
//formula w1_c2_new =  ((n1_nf2 * (w2_v1_ext + w2_v3_ext) + 1)>=(c*o1)|(n1_nf2=0))?1:0;
// Process 1 knows that Process 3 view is complete
//formula w1_c3_new =  ((n1_nf3 * (w3_v1_ext + w3_v2_ext) + 1)>=(c*o1)|(n1_nf3=0))?1:0;

// Process 1 knows that Process 2 view is complete
formula w1_c2_new =  ( (n1_nf2 * (w2_v1_ext + w2_v3_ext + 1)) >= (c*o1)|(n1_nf2=0) )?1:0;
// Process 1 knows that Process 3 view is complete
```

Listing A.37: *Prism Model for a system of three processes executing the 1-of-* * *selection algorithm with the moderately pessimistic* decision criterion, part (1)*

A.5.5 n=4, Moderately pessimistic

A.5.6 n=4, Pessimistic

```
formula w1_c3_new =  ( (n1_nf3 * (w3_v1_ext + w3_v2_ext + 1)) >= (c*o1)|(n1_nf3=0) )?1:0;


// Moderately Pessimistic Decision
//formula rec_msg_comp1_new =( (n1_nf2 + (n1_nf2 * (w2_v1_ext * w2_v3_ext))) , (n1_nf3 + (n1_nf3 *
      (w3_v1_ext * w3_v2_ext)))); // All messages received by the Process 1 are complete
//formula rec_msg_comp2_new = min (n2_nf3 + (n1_nf2 * (w2_v1_ext * w2_v3_ext))) + (!n2_nf1 +
      (n1_nf3 * (w3_v1_ext * w3_v2_ext))); // All messages received by the Process 1 are complete
//formula rec_msg_comp3_new = min (n3_nf1 + (n1_nf2 * (w2_v1_ext * w2_v3_ext))) + (!n3_nf2 +
      (n1_nf3 * (w3_v1_ext * w3_v2_ext))); // All messages received by the Process 1 are complete


// Label definitions for property specifications
//Abort
label "p1_AB" = (d1=0); //Process p1 aborts
label "p2_AB" = (d2=0); //Process p2 aborts
label "p3_AB" = (d3=0); //Process p3 aborts
//Decide
label "p1_DC" = (d1!=0); //Process p1 decides
label "p2_DC" = (d2!=0); //Process p2 decides
label "p3_DC" = (d3!=0); //Process p3 decides

label "p1_NEQ_p2" = (d1!=d2); // p1's decision is not equal to p2's decision
label "p1_NEQ_p3" = (d1!=d3); // p1's decision is not equal to p3's decision
label "p2_NEQ_p3" = (d2!=d3); // p2's decision is not equal to p3's decision

module Process_1
s1 : [0..N+3] init 1;  // Process 1 current state
RN1: [0..RN] init 0; // Current round
v1 : [0..N] init v1_ini; // Process 1 value
d1 : [0..N] init 0; // Process 1 decision
m1 : [1..N] init 1;
c1 : [0..N] init 0;

// Status of the message from the other processs
n1_nf2 :[0..1] init 1; // Process 1 has not received the message of Process 2
n1_nf3 :[0..1] init 1; // Process 1 has not received the message of Process 3
// Process 1 view of other processes
w1_v2 :[0..1] init 0; // Process 1 has the view of Process 2
w1_v3 :[0..1] init 0; // Process 1 has the view of Process 3
// Process 1 has confirmation that other processes have complete view
w1_c1 :[0..1] init 0; // Process 1 has confirmation from Process 1
w1_c2 :[0..1] init 0; // Process 1 has confirmation from Process 2
w1_c3 :[0..1] init 0; // Process 1 has confirmation from Process 3

// Process 1 sends its message;
[] s1=1 & token=N1 & RN1<RN -> 1:(s1'=2) & (token'=next) & (v1_ext'=v1) & (w1_v2_ext'=w1_v2) &
      (w1_v3_ext'=w1_v3) & (RN1'=RN1+1);
// Process 1 receives or loses the message of Process 2
[] s1=2 & token=N1 & RN1<=RN -> (1-Q): (s1'=3) & (n1_nf2'=1) + Q: (s1'=3) & (n1_nf2'=0);
// Process 1 receives or loses the message of Process 3
[] s1=3 & token=N1 & RN1<=RN -> (1-Q): (s1'=4) & (n1_nf3'=1) + Q: (s1'=4) & (n1_nf3'=0);
// Not last round, Process 1 computes the messages of other processes: updates its value, views
      and confirmations;
[] s1=N+1 & token=N1 & RN1<RN -> 1: (s1'=1) & (v1'=v1_new) & (w1_v2'=w1_v2_new) &
      (w1_v3'=w1_v3_new)& (token'=next);
// Last round, Process 1 computes the messages of other processes: updates its confirmations and,
      only for OP, updates value and views;
[] s1=N+1 & token=N1 & RN1=RN -> 1: (s1'=N+2) & (w1_c2'=w1_c2_new) & (w1_c3'=w1_c3_new) &
      (token'=next);
[] s1=N+2 & token=N1 & RN1=RN -> 1: (s1'=N+3) & (c1'=c1_new) & (m1'=m1_new) & (token'=next);
// Process 1 decides -> agree or abort
//[] s1=N+2 & token=N1 -> 1: (s1'=0) & (token'=next) & (d1'= decision_Orc);
// Process 1 decides -> agree or abort &
[] s1=N+3 & token=N1 & (m1 >= (c*o1)) & (c1 >= (c*(o1)))->1: (s1'=0) & (token'=next) & (d1'=v1);
[] s1=N+3 & token=N1 & ( (m1 < (c*o1)) | (c1 < (c*(o1))) )-> 1: (s1'=0) & (token'=next) & (d1'=0);
endmodule
module Process_2=Process_1 [N1=N2, s1=s2, m1=m2, c1=c2, o1=o2, v1=v2, d1=d2, RN1=RN2,
      n1_nf2=n2_nf3, n1_nf3=n2_nf1, w1_v2=w2_v3, w1_v3=w2_v1, w1_c1=w2_c2, w1_c2=w2_c3,
      w1_c3=w2_c1, v1_ext=v2_ext, v2_ext=v3_ext, v3_ext=v1_ext, w1_v2_ext=w2_v3_ext,
      w1_v3_ext=w2_v1_ext, w2_v1_ext=w3_v2_ext, w2_v3_ext=w3_v1_ext, w3_v1_ext=w1_v2_ext,
      w3_v2_ext=w1_v3_ext, v1_ini=v2_ini] endmodule
module Process_3=Process_1 [N1=N3, s1=s3, m1=m3, c1=c3, o1=o3, v1=v3, d1=d3, RN1=RN3,
      n1_nf2=n3_nf1, n1_nf3=n3_nf2, w1_v2=w3_v1, w1_v3=w3_v2, w1_c1=w3_c3, w1_c2=w3_c1,
      w1_c3=w3_c2, v1_ext=v3_ext, v2_ext=v1_ext, v3_ext=v2_ext, w1_v2_ext=w3_v1_ext,
      w1_v3_ext=w3_v2_ext, w2_v1_ext=w1_v3_ext, w2_v3_ext=w1_v2_ext, w3_v1_ext=w2_v3_ext,
      w3_v2_ext=w2_v1_ext, v1_ini=v3_ini, not_last=last] endmodule
```

Listing A.38: *Prism Model for a system of three processes executing the 1-of-* selection algorithm with the moderately pessimistic* decision criterion, part (2)*

```
dtmc
const N=3;      // Number of processes in the network (N cannot be modified)
const RN=2;         // Number of rounds in the protocol (RN>=2)
const double Q;     // Probability of losing a message (0<=q<=1)
const double c;     //Crrection parametr
const N1=1;         // Identity number of Process 1
const N2=2;         // Identity number of Process 2
const N3=3;         // Identity number of Process 3
const v1_ini=1;     // Initial value of Process 1
const v2_ini=2;     // Initial value of Process 2
const v3_ini=3;     // Initial value of Process 3
const not_last=1;   // Auxiliary constant to define the next process
const last=0;       // Auxiliary constant to define the next process

const o1=3;     //Oracle value is given randomely
const o2=3;     //Oracle value is given randomely
const o3=3;     //Oracle value is given randomely

global v1_ext : [0..N] init 0;  // Message value of Process 1
global v2_ext : [0..N] init 0;  // Message value of Process 2
global v3_ext : [0..N] init 0;  // Message value of Process 3

global w1_v2_ext : [0..1] init 0;   // Process 1 view of Process 2
global w1_v3_ext : [0..1] init 0;   // Process 1 view of Process 3

global w2_v1_ext : [0..1] init 0;   // Process 2 view of Process 1
global w2_v3_ext : [0..1] init 0;   // Process 2 view of Process 3

global w3_v1_ext : [0..1] init 0;   // Process 3 view of Process 1
global w3_v2_ext : [0..1] init 0;   // Process 3 view of Process 2

global token : [1..N] init 1;           // Token used to coordinate the processes
formula next = N1*not_last+1; // Define the next Process in the network

formula m1_new= 1 + w1_v2 + w1_v3; // length of view vector of process 1
formula m2_new= 1 + w2_v1 + w2_v3; // length of view vector of process 2
formula m3_new= 1 + w3_v1 + w3_v2; // length of view vector of process 3

formula c1_new= 1 + w1_c2 + w1_c3; // length of view vector of process 1
formula c2_new= 1 + w2_c1 + w2_c3; // length of view vector of process 2
formula c3_new= 1 + w3_c1 + w3_c2; // length of view vector of process 3

formula v1_new = max (v1 , (n1_nf2 * v2_ext),(n1_nf3 * v3_ext));  // Process 1 compute new value
formula w1_v2_new = max (w1_v2 , n1_nf2, (n1_nf3 * w3_v2_ext)); // Process 1 update its view of
      Process 2
formula w1_v3_new = max (w1_v3 , n1_nf3, (n1_nf2 * w2_v3_ext)); // Process 1 update its view of
      Process 3

// Process 1 knows that Process 2 view is complete
formula w1_c2_new =  ((w1_c2=1)|((n1_nf2 * (w2_v1_ext + w2_v3_ext + 1))>=(c*o1)))?1:0;
// Process 1 knows that Process 3 view is complete
formula w1_c3_new =  ((w1_c3=1)|((n1_nf3 * (w3_v1_ext + w3_v2_ext + 1))>=(c*o1)))?1:0;
```

Listing A.39: *Prism Model for a system of three processes executing the 1-of-* selection algorithm with the pessimistic* decision criterion, part (1)*

```
51
52  // Label definitions for property specifications
53  //Abort
54  label "p1_AB" = (d1=0); //Process p1 aborts
55  label "p2_AB" = (d2=0); //Process p2 aborts
56  label "p3_AB" = (d3=0); //Process p3 aborts
57  //Decide
58  label "p1_DC" = (d1!=0); //Process p1 decides
59  label "p2_DC" = (d2!=0); //Process p2 decides
60  label "p3_DC" = (d3!=0); //Process p3 decides
61
62  label "p1_NEQ_p2" = (d1!=d2); // p1's decision is not equal to p2's decision
63  label "p1_NEQ_p3" = (d1!=d3); // p1's decision is not equal to p3's decision
64  label "p2_NEQ_p3" = (d2!=d3); // p2's decision is not equal to p3's decision
65
66  module Process_1
67  s1 : [0..N+3] init 1;  // Process 1 current state
68  RN1: [0..RN] init 0; // Current round
69  v1 : [0..N] init v1_ini; // Process 1 value
70  d1 : [0..N] init 0; // Process 1 decision
71  m1 : [1..N] init 1;
72  c1 : [0..N] init 0;
73
74  // Status of the message from the other processs
75  n1_nf2 :[0..1] init 0; // Process 1 has not received the message of Process 2
76  n1_nf3 :[0..1] init 0; // Process 1 has not received the message of Process 3
77  // Process 1 view of other processes
78  w1_v2 :[0..1] init 0; // Process 1 has the view of Process 2
79  w1_v3 :[0..1] init 0; // Process 1 has the view of Process 3
80  // Process 1 has confirmation that other processes have complete view
81  w1_c1 :[0..1] init 0; // Process 1 has confirmation from Process 1
82  w1_c2 :[0..1] init 0; // Process 1 has confirmation from Process 2
83  w1_c3 :[0..1] init 0; // Process 1 has confirmation from Process 3
84
85  // Process 1 sends its message;
86  [] s1=1 & token=N1 & RN1<RN -> 1:(s1'=2) & (token'=next) & (v1_ext'=v1) & (w1_v2_ext'=w1_v2) &
        (w1_v3_ext'=w1_v3) & (RN1'=RN1+1);
87  // Process 1 receives or loses the message of Process 2
88  [] s1=2 & token=N1 & RN1<=RN -> (1-Q): (s1'=3) & (n1_nf2'=1) + Q: (s1'=3) & (n1_nf2'=0);
89  // Process 1 receives or loses the message of Process 3
90  [] s1=3 & token=N1 & RN1<=RN -> (1-Q): (s1'=4) & (n1_nf3'=1) + Q: (s1'=4) & (n1_nf3'=0);
91  // Not last round, Process 1 computes the messages of other processes: updates its value, views
        and confirmations;
92  [] s1=N+1 & token=N1 & RN1<RN -> 1: (s1'=1) & (v1'=v1_new) & (w1_v2'=w1_v2_new) &
        (w1_v3'=w1_v3_new) & (w1_c2'=w1_c2_new) & (w1_c3'=w1_c3_new) & (token'=next);
93  // Last round, Process 1 computes the messages of other processes: updates its confirmations and,
        only for OP, updates value and views;
94  [] s1=N+1 & token=N1 & RN1=RN -> 1: (s1'=N+2) & (w1_c2'=w1_c2_new) & (w1_c3'=w1_c3_new) &
        (token'=next);
95  [] s1=N+2 & token=N1 & RN1=RN -> 1: (s1'=N+3) & (c1'=c1_new) & (m1'=m1_new) & (token'=next);
96  // Process 1 decides -> agree or abort &
97  [] s1=N+3 & token=N1 & (m1 >= (c*o1)) & (c1 >= (c* o1) )->1: (s1'=0) & (token'=next) & (d1'=v1);
98  [] s1=N+3 & token=N1 & ((m1 < (c*o1)) | (c1 < (c*o1)) )-> 1: (s1'=0) & (token'=next) & (d1'=0);
99  endmodule
100 module Process_2=Process_1 [N1=N2, s1=s2, m1=m2, c1=c2, o1=o2, v1=v2, d1=d2, RN1=RN2,
        n1_nf2=n2_nf3, n1_nf3=n2_nf1, w1_v2=w2_v3, w1_v3=w2_v1, w1_c1=w2_c2, w1_c2=w2_c3,
        w1_c3=w2_c1, v1_ext=v2_ext, v2_ext=v3_ext, v3_ext=v1_ext, w1_v2_ext=w2_v3_ext,
        w1_v3_ext=w2_v1_ext, w2_v1_ext=w3_v2_ext, w2_v3_ext=w3_v1_ext, w3_v1_ext=w1_v2_ext,
        w3_v2_ext=w1_v3_ext, v1_ini=v2_ini] endmodule
101 module Process_3=Process_1 [N1=N3, s1=s3, m1=m3, c1=c3, o1=o3, v1=v3, d1=d3, RN1=RN3,
        n1_nf2=n3_nf1, n1_nf3=n3_nf2, w1_v2=w3_v1, w1_v3=w3_v2, w1_c1=w3_c3, w1_c2=w3_c1,
        w1_c3=w3_c2, v1_ext=v3_ext, v2_ext=v1_ext, v3_ext=v2_ext, w1_v2_ext=w3_v1_ext,
        w1_v3_ext=w3_v2_ext, w2_v1_ext=w1_v3_ext, w2_v3_ext=w1_v2_ext, w3_v1_ext=w2_v3_ext,
        w3_v2_ext=w2_v1_ext, v1_ini=v3_ini, not_last=last] endmodule
```

Listing A.40: *Prism Model for a system of three processes executing the 1-of-* selection algorithm with the pessimistic* decision criterion, part (2)*

```
dtmc
const N=4;      // Number of processes in the network (N cannot be modified)
const RN=2;         // Number of rounds in the protocol (RN>=2)
const double Q;     // Probability of losing a message (0<=q<=1)
const double c=0.5;     //Crrection parametr
const N1=1;         // Identity number of Process 1
const N2=2;         // Identity number of Process 2
const N3=3;         // Identity number of Process 3
const N4=4;         // Identity number of Process 4
const v1_ini=1;     // Initial value of Process 1
const v2_ini=2;     // Initial value of Process 2
const v3_ini=3;     // Initial value of Process 3
const v4_ini=4;     // Initial value of Process 4
const not_last=1;   // Auxiliary constant to define the next process
const last=0;       // Auxiliary constant to define the next process
const o1=4;     //Oracle value is given randomely
const o2=4;     //Oracle value is given randomely
const o3=4;     //Oracle value is given randomely
const o4=4;     //Oracle value is given randomely
global v1_ext : [0..N] init 0;  // Message value of Process 1
global v2_ext : [0..N] init 0;  // Message value of Process 2
global v3_ext : [0..N] init 0;  // Message value of Process 3
global v4_ext : [0..N] init 0;  // Message value of Process 4

global w1_v2_ext : [0..1] init 0;   // Process 1 view of Process 2
global w1_v3_ext : [0..1] init 0;   // Process 1 view of Process 3
global w1_v4_ext : [0..1] init 0;   // Process 1 view of Process 4

global w2_v1_ext : [0..1] init 0;   // Process 2 view of Process 1
global w2_v3_ext : [0..1] init 0;   // Process 2 view of Process 3
global w2_v4_ext : [0..1] init 0;   // Process 2 view of Process 4

global w3_v1_ext : [0..1] init 0;   // Process 3 view of Process 1
global w3_v2_ext : [0..1] init 0;   // Process 3 view of Process 2
global w3_v4_ext : [0..1] init 0;   // Process 3 view of Process 4

global w4_v1_ext : [0..1] init 0;   // Process 4 view of Process 1
global w4_v2_ext : [0..1] init 0;   // Process 4 view of Process 2
global w4_v3_ext : [0..1] init 0;   // Process 4 view of Process 3

global token : [1..N] init 1;           // Token used to coordinate the processes
formula next = N1*not_last+1; // Define the next Process in the network

formula m1_new= 1 + w1_v2 + w1_v3 + w1_v4; // length of view vector of process 1
formula m2_new= 1 + w2_v1 + w2_v3 + w2_v4; // length of view vector of process 2
formula m3_new= 1 + w3_v1 + w3_v2 + w3_v4; // length of view vector of process 3
formula m4_new= 1 + w4_v1 + w4_v2 + w4_v3; // length of view vector of process 4

formula v1_new = max (v1 , (n1_nf2 * v2_ext) , (n1_nf3 * v3_ext), (n1_nf4 * v4_ext));  // Process
      1 compute new value
formula w1_v2_new = max (w1_v2 , n1_nf2, (n1_nf3 * w3_v2_ext), (n1_nf4 * w4_v2_ext)); // Process 1
      update its view of Process 2
formula w1_v3_new = max (w1_v3 , n1_nf3, (n1_nf2 * w2_v3_ext), (n1_nf4 * w4_v3_ext)); // Process 1
      update its view of Process 3
formula w1_v4_new = max (w1_v4 , n1_nf4, (n1_nf2 * w2_v4_ext), (n1_nf3 * w3_v4_ext)); // Process 1
      update its view of Process 4

// Label definitions for property specifications
label "p1_AB"= (d1=0); //Process p1 aborts
```

Listing A.41: *Prism Model for a system of four processes executing the 1-of-* selection algorithm with the optimistic* decision criterion, part (1)*

```
label "p2_AB"= (d2=0); //Process p2 aborts
label "p3_AB"= (d3=0); //Process p3 aborts
label "p4_AB"= (d4=0); //Process p3 aborts
label "p1_DC"= (d1!=0); //Process p1 decides
label "p2_DC"= (d2!=0); //Process p2 decides
label "p3_DC"= (d3!=0); //Process p3 decides
label "p4_DC"= (d4!=0); //Process p3 decides

label "p1_NEQ_p2" = (d1!=d2); // p1's decision is not equal to p2's decision
label "p1_NEQ_p3" = (d1!=d3); // p1's decision is not equal to p3's decision
label "p1_NEQ_p4" = (d1!=d4); // p1's decision is not equal to p4's decision
label "p2_NEQ_p3" = (d2!=d3); // p2's decision is not equal to p3's decision
label "p2_NEQ_p4" = (d2!=d4); // p2's decision is not equal to p4's decision
label "p3_NEQ_p4" = (d3!=d4); // p3's decision is not equal to p4's decision

label "p1_EQ_p2" = (d1=d2); // p1's decision is equal to p2's decision
label "p1_EQ_p3" = (d1=d3); // p1's decision is equal to p3's decision
label "p1_EQ_p4" = (d1=d4); // p1's decision is equal to p4's decision
label "p2_EQ_p3" = (d2=d3); // p2's decision is equal to p3's decision
label "p2_EQ_p4" = (d2=d4); // p2's decision is equal to p4's decision
label "p3_EQ_p4" = (d3=d4); // p3's decision is equal to p4's decision

module Process_1
s1 : [0..N+3] init 1;       // Process 1's current state
RN1: [0..RN] init 0;        // Process 1's current round
v1 : [0..N] init v1_ini;    // Process 1's value
d1 : [0..N] init 0;         // Process 1's decision
m1 : [1..N] init 1;     // Size of Process 1's view vector

// Status of the messages sent from the other processs
n1_nf2 :[0..1] init 0;      // Process 1 has not received the message of Process 2
n1_nf3 :[0..1] init 0;      // Process 1 has not received the message of Process 3
n1_nf4 :[0..1] init 0;      // Process 1 has not received the message of Process 4
// Process 1's view of other processes
w1_v2 :[0..1] init 0;       // Process 1 has the view of Process 2
w1_v3 :[0..1] init 0;       // Process 1 has the view of Process 3
w1_v4 :[0..1] init 0;       // Process 1 has the view of Process 4

// Process 1 sends its message;
[] s1=1 & token=N1 & RN1<RN -> 1:(s1'=2) & (token'=next) & (v1_ext'=v1) & (w1_v2_ext'=w1_v2) &
      (w1_v3_ext'=w1_v3) & (w1_v4_ext'=w1_v4) & (RN1'=RN1+1);
// Process 1 receives or loses the message of Process 2
[] s1=2 & token=N1 & RN1<=RN -> (1-Q): (s1'=3) & (n1_nf2'=1) + Q: (s1'=3) & (n1_nf2'=0);
// Process 1 receives or loses the message of Process 3
[] s1=3 & token=N1 & RN1<=RN -> (1-Q): (s1'=4) & (n1_nf3'=1) + Q: (s1'=4) & (n1_nf3'=0);
// Process 1 receives or loses the message of Process 4
[] s1=4 & token=N1 & RN1<=RN -> (1-Q): (s1'=5) & (n1_nf4'=1) + Q: (s1'=5) & (n1_nf4'=0);
// Not last round, Process 1 computes the messages of other processes: updates its value, views
      and confirmations;
[] s1=N+1 & token=N1 & RN1<RN -> 1: (s1'=1) & (v1'=v1_new) & (w1_v2'=w1_v2_new) &
      (w1_v3'=w1_v3_new) & (w1_v4'=w1_v4_new) & (token'=next);
// Last round, Process 1 computes the messages of other processes: updates its confirmations and,
      only for OP, updates value and views;
[] s1=N+1 & token=N1 & RN1=RN -> 1: (s1'=N+2) & (v1'=v1_new) & (w1_v2'=w1_v2_new) &
      (w1_v3'=w1_v3_new) & (w1_v4'=w1_v4_new) & (token'=next);
[] s1=N+2 & token=N1 & RN1=RN -> 1: (s1'=N+3) & (m1'=m1_new) & (token'=next);
// Process 1 decides -> agree or abort
[] s1=N+3 & token=N1 & (m1 >= (c* o1))-> 1: (s1'=0) & (token'=next) & (d1'=v1);
[] s1=N+3 & token=N1 & (m1 < (c*o1))-> 1: (s1'=0) & (token'=next) & (d1'=0);
endmodule
module Process_2=Process_1 [N1=N2, s1=s2, m1=m2, o1=o2, v1=v2, d1=d2, RN1=RN2, n1_nf2=n2_nf3,
      n1_nf3=n2_nf4, n1_nf4=n2_nf1, w1_v2=w2_v3, w1_v3=w2_v4, w1_v4=w2_v1, w1_c2=w2_c3,
      w1_c3=w2_c4, w1_c4=w2_c1, v1_ext=v2_ext, v2_ext=v3_ext, v3_ext=v4_ext, v4_ext=v1_ext,
      w1_v2_ext=w2_v3_ext, w1_v3_ext=w2_v4_ext, w1_v4_ext=w2_v1_ext, w2_v1_ext=w3_v2_ext,
      w2_v3_ext=w3_v4_ext, w2_v4_ext=w3_v1_ext, w3_v1_ext=w4_v2_ext, w3_v2_ext=w4_v3_ext,
      w3_v4_ext=w4_v1_ext, w4_v1_ext=w1_v2_ext, w4_v2_ext=w1_v3_ext, w4_v3_ext=w1_v4_ext,
      v1_ini=v2_ini] endmodule
module Process_3=Process_1 [N1=N3, s1=s3, m1=m3, o1=o3, v1=v3, d1=d3, RN1=RN3, n1_nf2=n3_nf4,
      n1_nf3=n3_nf1, n1_nf4=n3_nf2, w1_v2=w3_v4, w1_v3=w3_v1, w1_v4=w3_v2, w1_c2=w3_c4,
      w1_c3=w3_c1, w1_c4=w3_c2, v1_ext=v3_ext, v2_ext=v4_ext, v3_ext=v1_ext, v4_ext=v2_ext,
      w1_v2_ext=w3_v4_ext, w1_v3_ext=w3_v1_ext, w1_v4_ext=w3_v2_ext, w2_v1_ext=w4_v3_ext,
      w2_v3_ext=w4_v1_ext, w2_v4_ext=w4_v2_ext, w3_v1_ext=w1_v3_ext, w3_v2_ext=w1_v4_ext,
      w3_v4_ext=w1_v2_ext, w4_v1_ext=w2_v3_ext, w4_v2_ext=w2_v4_ext, w4_v3_ext=w2_v1_ext,
      v1_ini=v3_ini] endmodule
module Process_4=Process_1 [N1=N4, s1=s4, m1=m4, o1=o4, v1=v4, d1=d4, RN1=RN4, n1_nf2=n4_nf1,
      n1_nf3=n4_nf2, n1_nf4=n4_nf3, w1_v2=w4_v1, w1_v3=w4_v2, w1_v4=w4_v3, w1_c2=w4_c1,
      w1_c3=w4_c2, w1_c4=w4_c3, v1_ext=v4_ext, v2_ext=v1_ext, v3_ext=v2_ext, v4_ext=v3_ext,
      w1_v2_ext=w4_v1_ext, w1_v3_ext=w4_v2_ext, w1_v4_ext=w4_v3_ext, w2_v1_ext=w1_v4_ext,
      w2_v3_ext=w1_v2_ext, w2_v4_ext=w1_v3_ext, w3_v1_ext=w2_v4_ext, w3_v2_ext=w2_v1_ext,
      w3_v4_ext=w2_v3_ext, w4_v1_ext=w3_v4_ext, w4_v2_ext=w3_v1_ext, w4_v3_ext=w3_v2_ext,
      v1_ini=v4_ini, not_last=last] endmodule
```

Listing A.42: *Prism Model for a system of four processes executing the 1-of-* selection algorithm with the optimistic* decision criterion, part (2)*

```
dtmc
const N=4;      // Number of processes in the network (N cannot be modified)
const RN=2;          // Number of rounds in the protocol (RN>=2)
const double Q;      // Probability of losing a message (0<=q<=1)
const double c=0.5;     //Crrection parametr
const N1=1;          // Identity number of Process 1
const N2=2;          // Identity number of Process 2
const N3=3;          // Identity number of Process 3
const N4=4;          // Identity number of Process 4
const v1_ini=1;      // Initial value of Process 1
const v2_ini=2;      // Initial value of Process 2
const v3_ini=3;      // Initial value of Process 3
const v4_ini=4;      // Initial value of Process 4
const not_last=1;    // Auxiliary constant to define the next process
const last=0;        // Auxiliary constant to define the next process
const o1=4;      //Oracle value is given randomely
const o2=4;      //Oracle value is given randomely
const o3=4;      //Oracle value is given randomely
const o4=4;      //Oracle value is given randomely
global v1_ext : [0..N] init 0;  // Message value of Process 1
global v2_ext : [0..N] init 0;  // Message value of Process 2
global v3_ext : [0..N] init 0;  // Message value of Process 3
global v4_ext : [0..N] init 0;  // Message value of Process 4

global w1_v2_ext : [0..1] init 0;   // Process 1 view of Process 2
global w1_v3_ext : [0..1] init 0;   // Process 1 view of Process 3
global w1_v4_ext : [0..1] init 0;   // Process 1 view of Process 4

global w2_v1_ext : [0..1] init 0;   // Process 2 view of Process 1
global w2_v3_ext : [0..1] init 0;   // Process 2 view of Process 3
global w2_v4_ext : [0..1] init 0;   // Process 2 view of Process 4

global w3_v1_ext : [0..1] init 0;   // Process 3 view of Process 1
global w3_v2_ext : [0..1] init 0;   // Process 3 view of Process 2
global w3_v4_ext : [0..1] init 0;   // Process 3 view of Process 4

global w4_v1_ext : [0..1] init 0;   // Process 4 view of Process 1
global w4_v2_ext : [0..1] init 0;   // Process 4 view of Process 2
global w4_v3_ext : [0..1] init 0;   // Process 4 view of Process 3

global token : [1..N] init 1;           // Token used to coordinate the processes
formula next = N1*not_last+1; // Define the next Process in the network

formula m1_new= 1 + w1_v2 + w1_v3 + w1_v4; // length of view vector of process 1
formula m2_new= 1 + w2_v1 + w2_v3 + w2_v4; // length of view vector of process 2
formula m3_new= 1 + w3_v1 + w3_v2 + w3_v4; // length of view vector of process 3
formula m4_new= 1 + w4_v1 + w4_v2 + w4_v3; // length of view vector of process 4

//formula c1_new= w1_c2 + w1_c3 + w1_c4; // length of view vector of process 1
//formula c2_new= w2_c1 + w2_c3 + w2_c4; // length of view vector of process 2
//formula c3_new= w3_c1 + w3_c2 + w3_c4; // length of view vector of process 3
//formula c4_new= w4_c1 + w4_c2 + w3_c3; // length of view vector of process 4

formula c1_new= 1 + w1_c2 + w1_c3 + w1_c4; // length of view vector of process 1
formula c2_new= 1 + w2_c1 + w2_c3 + w2_c4; // length of view vector of process 2
formula c3_new= 1 + w3_c1 + w3_c2 + w3_c4; // length of view vector of process 3
formula c4_new= 1 + w4_c1 + w4_c2 + w4_c3; // length of view vector of process 4

formula v1_new = max (v1 , (n1_nf2 * v2_ext) , (n1_nf3 * v3_ext), (n1_nf4 * v4_ext));  // Process
      1 compute new value
formula w1_v2_new = max (w1_v2 , n1_nf2, (n1_nf3 * w3_v2_ext), (n1_nf4 * w4_v2_ext)); // Process 1
      update its view of Process 2
formula w1_v3_new = max (w1_v3 , n1_nf3, (n1_nf2 * w2_v3_ext), (n1_nf4 * w4_v3_ext)); // Process 1
      update its view of Process 3
formula w1_v4_new = max (w1_v4 , n1_nf4, (n1_nf2 * w2_v4_ext), (n1_nf3 * w3_v4_ext)); // Process 1
      update its view of Process 4

// Process 1 knows that Process 2 view is complete
formula w1_c2_new =  ((n1_nf2 * (w2_v1_ext + w2_v3_ext + w2_v4_ext + 1))>=(c*o1)|(n1_nf2=0))?1:0;
// Process 1 knows that Process 3 view is complete
formula w1_c3_new =  ((n1_nf3 * (w3_v1_ext + w3_v2_ext + w3_v4_ext + 1))>=(c*o1)|(n1_nf3=0))?1:0;
// Process 1 knows that Process 2 view is complete
formula w1_c4_new =  ((n1_nf4 * (w4_v1_ext + w4_v2_ext + w4_v3_ext + 1))>=(c*o1)|(n1_nf4=0))?1:0;
```

Listing A.43: *Prism Model for a system of four processes executing the 1-of-* selection algorithm with the moderately pessimistic* decision criterion, part (1)*

```
71  //Abort
72  label "p1_AB" = (d1=0); //Process p1 aborts
73  label "p2_AB" = (d2=0); //Process p2 aborts
74  label "p3_AB" = (d3=0); //Process p3 aborts
75  //Decide
76  label "p1_DC" = (d1!=0); //Process p1 decides
77  label "p2_DC" = (d2!=0); //Process p2 decides
78  label "p3_DC" = (d3!=0); //Process p3 decides
79
80  label "p1_NEQ_p2" = (d1!=d2); // p1's decision is not equal to p2's decision
81  label "p1_NEQ_p3" = (d1!=d3); // p1's decision is not equal to p3's decision
82  label "p2_NEQ_p3" = (d2!=d3); // p2's decision is not equal to p3's decision
83
84  module Process_1
85  s1 : [0..N+3] init 1;  // Process 1 current state
86  RN1: [0..RN] init 0; // Current round
87  v1 : [0..N] init v1_ini; // Process 1 value
88  d1 : [0..N] init 0; // Process 1 decision
89  m1 : [1..N] init 1;
90  c1 : [0..N] init 0;
91
92  // Status of the message from the other processs
93  n1_nf2 :[0..1] init 1; // Process 1 has not received the message of Process 2
94  n1_nf3 :[0..1] init 1; // Process 1 has not received the message of Process 3
95  // Process 1 view of other processes
96  w1_v2 :[0..1] init 0; // Process 1 has the view of Process 2
97  w1_v3 :[0..1] init 0; // Process 1 has the view of Process 3
98  // Process 1 has confirmation that other processes have complete view
99  w1_c1 :[0..1] init 0; // Process 1 has confirmation from Process 1
100 w1_c2 :[0..1] init 0; // Process 1 has confirmation from Process 2
101 w1_c3 :[0..1] init 0; // Process 1 has confirmation from Process 3
102
103 // Process 1 sends its message;
104 [] s1=1 & token=N1 & RN1<RN -> 1:(s1'=2) & (token'=next) & (v1_ext'=v1) & (w1_v2_ext'=w1_v2) &
        (w1_v3_ext'=w1_v3) & (RN1'=RN1+1);
105 // Process 1 receives or loses the message of Process 2
106 [] s1=2 & token=N1 & RN1<=RN -> (1-Q): (s1'=3) & (n1_nf2'=1) + Q: (s1'=3) & (n1_nf2'=0);
107 // Process 1 receives or loses the message of Process 3
108 [] s1=3 & token=N1 & RN1<=RN -> (1-Q): (s1'=4) & (n1_nf3'=1) + Q: (s1'=4) & (n1_nf3'=0);
109 // Not last round, Process 1 computes the messages of other processes: updates its value, views
        and confirmations;
110 [] s1=N+1 & token=N1 & RN1<RN -> 1: (s1'=1) & (v1'=v1_new) & (w1_v2'=w1_v2_new) &
        (w1_v3'=w1_v3_new)& (token'=next);
111 // Last round, Process 1 computes the messages of other processes: updates its confirmations and,
        only for OP, updates value and views;
112 [] s1=N+1 & token=N1 & RN1=RN -> 1: (s1'=N+2) & (w1_c2'=w1_c2_new) & (w1_c3'=w1_c3_new) &
        (token'=next);
113 [] s1=N+2 & token=N1 & RN1=RN -> 1: (s1'=N+3) & (c1'=c1_new) & (m1'=m1_new) & (token'=next);
114 // Process 1 decides -> agree or abort
115 //[] s1=N+2 & token=N1 -> 1: (s1'=0) & (token'=next) & (d1'= decision_Orc);
116 // Process 1 decides -> agree or abort &
117 [] s1=N+3 & token=N1 & (m1 >= (c*o1)) & (c1 >= (c*(o1)))->1: (s1'=0) & (token'=next) & (d1'=v1);
118 [] s1=N+3 & token=N1 & ( (m1 < (c*o1)) | (c1 < (c*(o1))) )-> 1: (s1'=0) & (token'=next) & (d1'=0);
119 endmodule
120 module Process_2=Process_1 [N1=N2, s1=s2, m1=m2, c1=c2, o1=o2, v1=v2, d1=d2, RN1=RN2,
        n1_nf2=n2_nf3, n1_nf3=n2_nf1, w1_v2=w2_v3, w1_v3=w2_v1, w1_c1=w2_c2, w1_c2=w2_c3,
        w1_c3=w2_c1, v1_ext=v2_ext, v2_ext=v3_ext, v3_ext=v1_ext, w1_v2_ext=w2_v3_ext,
        w1_v3_ext=w2_v1_ext, w2_v1_ext=w3_v2_ext, w2_v3_ext=w3_v1_ext, w3_v1_ext=w1_v2_ext,
        w3_v2_ext=w1_v3_ext, v1_ini=v2_ini] endmodule
121 module Process_3=Process_1 [N1=N3, s1=s3, m1=m3, c1=c3, o1=o3, v1=v3, d1=d3, RN1=RN3,
        n1_nf2=n3_nf1, n1_nf3=n3_nf2, w1_v2=w3_v1, w1_v3=w3_v2, w1_c1=w3_c3, w1_c2=w3_c1,
        w1_c3=w3_c2, v1_ext=v3_ext, v2_ext=v1_ext, v3_ext=v2_ext, w1_v2_ext=w3_v1_ext,
        w1_v3_ext=w3_v2_ext, w2_v1_ext=w1_v3_ext, w2_v3_ext=w1_v2_ext, w3_v1_ext=w2_v3_ext,
        w3_v2_ext=w2_v1_ext, v1_ini=v3_ini, not_last=last] endmodule
```

Listing A.44: *Prism Model for a system of four processes executing the 1-of-* selection algorithm with the moderately pessimistic* decision criterion, part (2)*

```
dtmc
const N=4;      // Number of processes in the network (N cannot be modified)
const RN=2;         // Number of rounds in the protocol (RN>=2)
const double Q;     // Probability of losing a message (0<=q<=1)
const double c=0.5;     //Crrection parametr
const N1=1;         // Identity number of Process 1
const N2=2;         // Identity number of Process 2
const N3=3;         // Identity number of Process 3
const N4=4;         // Identity number of Process 4
const v1_ini=1;     // Initial value of Process 1
const v2_ini=2;     // Initial value of Process 2
const v3_ini=3;     // Initial value of Process 3
const v4_ini=4;     // Initial value of Process 4
const not_last=1;   // Auxiliary constant to define the next process
const last=0;       // Auxiliary constant to define the next process
const o1=4;     //Oracle value is given randomely
const o2=4;     //Oracle value is given randomely
const o3=4;     //Oracle value is given randomely
const o4=4;     //Oracle value is given randomely
global v1_ext : [0..N] init 0;  // Message value of Process 1
global v2_ext : [0..N] init 0;  // Message value of Process 2
global v3_ext : [0..N] init 0;  // Message value of Process 3
global v4_ext : [0..N] init 0;  // Message value of Process 4

global w1_v2_ext : [0..1] init 0;   // Process 1 view of Process 2
global w1_v3_ext : [0..1] init 0;   // Process 1 view of Process 3
global w1_v4_ext : [0..1] init 0;   // Process 1 view of Process 4

global w2_v1_ext : [0..1] init 0;   // Process 2 view of Process 1
global w2_v3_ext : [0..1] init 0;   // Process 2 view of Process 3
global w2_v4_ext : [0..1] init 0;   // Process 2 view of Process 4

global w3_v1_ext : [0..1] init 0;   // Process 3 view of Process 1
global w3_v2_ext : [0..1] init 0;   // Process 3 view of Process 2
global w3_v4_ext : [0..1] init 0;   // Process 3 view of Process 4

global w4_v1_ext : [0..1] init 0;   // Process 4 view of Process 1
global w4_v2_ext : [0..1] init 0;   // Process 4 view of Process 2
global w4_v3_ext : [0..1] init 0;   // Process 4 view of Process 3
```

Listing A.45: *Prism Model for a system of four processes executing the 1-of-* selection algorithm with the pessimistic* decision criterion, part (1)*

```
41 global token : [1..N] init 1;           // Token used to coordinate the processes
42 formula next = N1*not_last+1; // Define the next Process in the network
43
44 formula m1_new= 1 + w1_v2 + w1_v3 + w1_v4; // length of view vector of process 1
45 formula m2_new= 1 + w2_v1 + w2_v3 + w2_v4; // length of view vector of process 2
46 formula m3_new= 1 + w3_v1 + w3_v2 + w3_v4; // length of view vector of process 3
47 formula m4_new= 1 + w4_v1 + w4_v2 + w4_v3; // length of view vector of process 4
48
49 formula c1_new=  w1_c2 + w1_c3 + w1_c4; // length of view vector of process 1
50 formula c2_new=  w2_c1 + w2_c3 + w2_c4; // length of view vector of process 2
51 formula c3_new=  w3_c1 + w3_c2 + w3_c4; // length of view vector of process 3
52 formula c4_new=  w4_c1 + w4_c2 + w3_c3; // length of view vector of process 3
53
54 formula v1_new = max (v1 , (n1_nf2 * v2_ext) , (n1_nf3 * v3_ext), (n1_nf4 * v4_ext));  // Process
       1 compute new value
55 formula w1_v2_new = max (w1_v2 , n1_nf2, (n1_nf3 * w3_v2_ext), (n1_nf4 * w4_v2_ext)); // Process 1
       update its view of Process 2
56 formula w1_v3_new = max (w1_v3 , n1_nf3, (n1_nf2 * w2_v3_ext), (n1_nf4 * w4_v3_ext)); // Process 1
       update its view of Process 3
57 formula w1_v4_new = max (w1_v4 , n1_nf4, (n1_nf2 * w2_v4_ext), (n1_nf3 * w3_v4_ext)); // Process 1
       update its view of Process 4
58
59 // Process 1 knows that Process 2 view is complete
60 formula w1_c2_new = ((w1_c2=1)|(n1_nf2*(w2_v1_ext+w2_v3_ext+w2_v4_ext + 1)) >=(c*o1))?1:0;
61 // Process 1 knows that Process 3 view is complete
62 formula w1_c3_new = ((w1_c3=1)|(n1_nf3*(w3_v1_ext+w3_v2_ext+w3_v4_ext + 1)) >=(c*o1))?1:0;
63 // Process 1 knows that Process 2 view is complete
64 formula w1_c4_new = ((w1_c4=1)|(n1_nf4*(w4_v1_ext+w4_v2_ext+w4_v3_ext + 1)) >=(c*o1))?1:0;
65
66 // Label definitions for property specifications
67 label "p1_AB"= (d1=0); //Process p1 aborts
68 label "p2_AB"= (d2=0); //Process p2 aborts
69 label "p3_AB"= (d3=0); //Process p3 aborts
70 label "p4_AB"= (d4=0); //Process p4 aborts
71
72 label "p1_DC"= (d1!=0); //Process p1 decides
73 label "p2_DC"= (d2!=0); //Process p2 decides
74 label "p3_DC"= (d3!=0); //Process p3 decides
75 label "p4_DC"= (d4!=0); //Process p4 decides
76
77 label "p1_NEQ_p2" = (d1!=d2); // p1's decision is not equal to p2's decision
78 label "p1_NEQ_p3" = (d1!=d3); // p1's decision is not equal to p3's decision
79 label "p1_NEQ_p4" = (d1!=d4); // p1's decision is not equal to p4's decision
80 label "p2_NEQ_p3" = (d2!=d3); // p2's decision is not equal to p3's decision
```

Listing A.46: *Prism Model for a system of four processes executing the 1-of-* selection algorithm with the pessimistic* decision criterion, part (2)*

```
label "p2_NEQ_p4" = (d2!=d4); // p2's decision is not equal to p4's decision
label "p3_NEQ_p4" = (d3!=d4); // p3's decision is not equal to p4's decision

label "p1_EQ_p2" = (d1=d2); // p1's decision is equal to p2's decision
label "p1_EQ_p3" = (d1=d3); // p1's decision is equal to p3's decision
label "p1_EQ_p4" = (d1=d4); // p1's decision is equal to p4's decision
label "p2_EQ_p3" = (d2=d3); // p2's decision is equal to p3's decision
label "p2_EQ_p4" = (d2=d4); // p2's decision is equal to p4's decision
label "p3_EQ_p4" = (d3=d4); // p3's decision is equal to p4's decision

module Process_1
s1 : [0..N+3] init 1;        // Process 1's current state
RN1: [0..RN] init 0;         // Process 1's current round
v1 : [0..N] init v1_ini;     // Process 1's value
d1 : [0..N] init 0;          // Process 1's decision
m1 : [1..N] init 1;      // Size of Process 1's view vector
c1 : [0..N] init 0;

// Status of the messages sent from the other processs
n1_nf2 :[0..1] init 0;       // Process 1 has not received the message of Process 2
n1_nf3 :[0..1] init 0;       // Process 1 has not received the message of Process 3
n1_nf4 :[0..1] init 0;       // Process 1 has not received the message of Process 4
// Process 1's view of other processes
w1_v2 :[0..1] init 0;        // Process 1 has the view of Process 2
w1_v3 :[0..1] init 0;        // Process 1 has the view of Process 3
w1_v4 :[0..1] init 0;        // Process 1 has the view of Process 4
// Process 1 has confirmation that other processes have complete view
w1_c1 :[0..1] init 0; // Process 1 has confirmation from Process 1
w1_c2 :[0..1] init 0; // Process 1 has confirmation from Process 2
w1_c3 :[0..1] init 0; // Process 1 has confirmation from Process 3
w1_c4 :[0..1] init 0; // Process 1 has confirmation from Process 4
// Process 1 sends its message;
[] s1=1 & token=N1 & RN1<RN -> 1:(s1'=2) & (token'=next) & (v1_ext'=v1) & (w1_v2_ext'=w1_v2) &
      (w1_v3_ext'=w1_v3) & (w1_v4_ext'=w1_v4) & (RN1'=RN1+1);
// Process 1 receives or loses the message of Process 2
[] s1=2 & token=N1 & RN1<=RN -> (1-Q): (s1'=3) & (n1_nf2'=1) + Q: (s1'=3) & (n1_nf2'=0);
// Process 1 receives or loses the message of Process 3
[] s1=3 & token=N1 & RN1<=RN -> (1-Q): (s1'=4) & (n1_nf3'=1) + Q: (s1'=4) & (n1_nf3'=0);
// Process 1 receives or loses the message of Process 4
[] s1=4 & token=N1 & RN1<=RN -> (1-Q): (s1'=5) & (n1_nf4'=1) + Q: (s1'=5) & (n1_nf4'=0);
// Not last round, Process 1 computes the messages of other processes: updates its value, views
      and confirmations;
[] s1=N+1 & token=N1 & RN1<RN -> 1: (s1'=1) & (v1'=v1_new) & (w1_v2'=w1_v2_new) &
      (w1_v3'=w1_v3_new) & (w1_v4'=w1_v4_new) & (w1_c2'=w1_c2_new) & (w1_c3'=w1_c3_new) &
      (w1_c4'=w1_c4_new) & (token'=next);
// Last round, Process 1 computes the messages of other processes: updates its confirmations and,
      only for OP, updates value and views;
[] s1=N+1 & token=N1 & RN1=RN -> 1: (s1'=N+2) & (v1'=v1_new) & (w1_c2'=w1_c2_new) &
      (w1_c3'=w1_c3_new) & (w1_c4'=w1_c4_new) & (token'=next);
[] s1=N+2 & token=N1 & RN1=RN -> 1: (s1'=N+3) & (m1'=m1_new) & (c1'=c1_new)& (token'=next);
// Process 1 decides -> agree or abort
[] s1=N+3 & token=N1 & (m1 >= (c*o1)) & (c1 >= ( (c*o1)-1))->1: (s1'=0) & (token'=next) & (d1'=v1);
[] s1=N+3 & token=N1 & ((m1 < (c*o1))| (c1 < ((c*o1)-1)))-> 1: (s1'=0) & (token'=next) & (d1'=0);
endmodule
module Process_2=Process_1 [N1=N2, s1=s2, m1=m2, c1=c2, o1=o2, v1=v2, d1=d2, RN1=RN2,
      n1_nf2=n2_nf3, n1_nf3=n2_nf4, n1_nf4=n2_nf1, w1_v2=w2_v3, w1_v3=w2_v4, w1_v4=w2_v1,
      w1_c1=w2_c2, w1_c2=w2_c3, w1_c3=w2_c4, w1_c4=w2_c1, v1_ext=v2_ext, v2_ext=v3_ext,
      v3_ext=v4_ext, v4_ext=v1_ext, w1_v2_ext=w2_v3_ext, w1_v3_ext=w2_v4_ext ,
      w1_v4_ext=w2_v1_ext, w2_v1_ext=w3_v2_ext, w2_v3_ext=w3_v4_ext, w2_v4_ext=w3_v1_ext,
      w3_v1_ext=w4_v2_ext, w3_v2_ext=w4_v3_ext, w3_v4_ext=w4_v1_ext, w4_v1_ext=w1_v2_ext,
      w4_v2_ext=w1_v3_ext, w4_v3_ext=w1_v4_ext, v1_ini=v2_ini] endmodule
module Process_3=Process_1 [N1=N3, s1=s3, m1=m3, c1=c3, o1=o3, v1=v3, d1=d3, RN1=RN3,
      n1_nf2=n3_nf4, n1_nf3=n3_nf1, n1_nf4=n3_nf2, w1_v2=w3_v4, w1_v3=w3_v1, w1_v4=w3_v2,
      w1_c1=w3_c3, w1_c2=w3_c4, w1_c3=w3_c1, w1_c4=w3_c2, v1_ext=v3_ext, v2_ext=v4_ext,
      v3_ext=v1_ext, v4_ext=v2_ext, w1_v2_ext=w3_v4_ext, w1_v3_ext=w3_v1_ext ,
      w1_v4_ext=w3_v2_ext, w2_v1_ext=w4_v3_ext, w2_v3_ext=w4_v1_ext, w2_v4_ext=w4_v2_ext,
      w3_v1_ext=w1_v3_ext, w3_v2_ext=w1_v4_ext, w3_v4_ext=w1_v2_ext, w4_v1_ext=w2_v3_ext,
      w4_v2_ext=w2_v4_ext, w4_v3_ext=w2_v1_ext, v1_ini=v3_ini] endmodule
module Process_4=Process_1 [N1=N4, s1=s4, m1=m4, c1=c4, o1=o4, v1=v4, d1=d4, RN1=RN4,
      n1_nf2=n4_nf1, n1_nf3=n4_nf2, n1_nf4=n4_nf3, w1_v2=w4_v1, w1_v3=w4_v2, w1_v4=w4_v3,
      w1_c1=w4_c4, w1_c2=w4_c1, w1_c3=w4_c2, w1_c4=w4_c3, v1_ext=v4_ext, v2_ext=v1_ext,
      v3_ext=v2_ext, v4_ext=v3_ext, w1_v2_ext=w4_v1_ext, w1_v3_ext=w4_v2_ext ,
      w1_v4_ext=w4_v3_ext, w2_v1_ext=w1_v4_ext, w2_v3_ext=w1_v2_ext, w2_v4_ext=w1_v3_ext,
      w3_v1_ext=w2_v4_ext, w3_v2_ext=w2_v1_ext, w3_v4_ext=w2_v3_ext, w4_v1_ext=w3_v4_ext,
      w4_v2_ext=w3_v1_ext, w4_v3_ext=w3_v2_ext, v1_ini=v4_ini, not_last=last] endmodule
```

Listing A.47: *Prism Model for a system of four processes executing the 1-of-* selection algorithm with the pessimistic* decision criterion, part (3)*

A.6 Results for *1-of-** Selection Algorithms

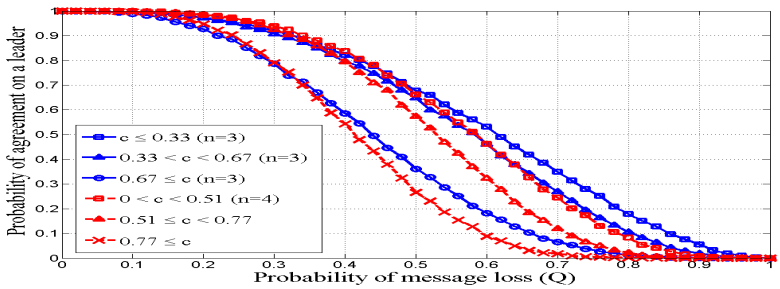
A.6.1 Varying the correction parameter (c)

Fig. A.1 shows the probability of agreement on a leader and on abort for the *optimistic** selection algorithm as a function of Q with different settings of the *correction* parameter (c) in the range of 0 to 1 for systems of $n = 3$ and $n = 4$ processes and $R = 2$ and correct oracles. As we see from the results given in A.1(a), the probability of agreement on a leader for similar ranges of c in general is higher for system of three processes (i.e. $n = 3$) compare to the system of four processes (i.e. $n = 4$).

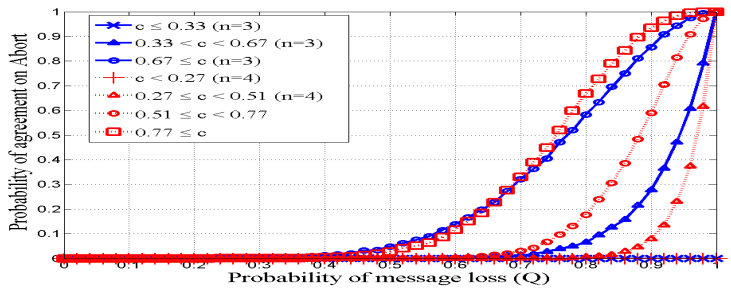
From the results given in A.1(b), the probability of agreement on abort for similar ranges of c in general is lower for the system of three processes (i.e. $n = 3$) compare to the system of four processes (i.e. $n = 4$).

Fig. A.2 shows the probability of having disagreement among processes for systems of three and four processes executing the *1-of-** selection in two rounds as a function of Q using the *optimistic** algorithm. The given results are for the same settings of the c values as given in Fig. A.1 and correct oracles.

Fig. A.2(a) shows the probability of unsafe disagreement and Fig. A.2(b) shows the probability of safe disagreement for both systems of three and four processes. From the results in Fig. A.2(a), we see that the probability of unsafe disagreement for similar ranges of c values is higher for the system of four processes compare to the system of three processes. From the results given in Fig. A.2(b) we see that in most of the cases we have lower probabilities of safe disagreement for similar ranges of the c parameter for a system of four processes compare to the system of three processes.

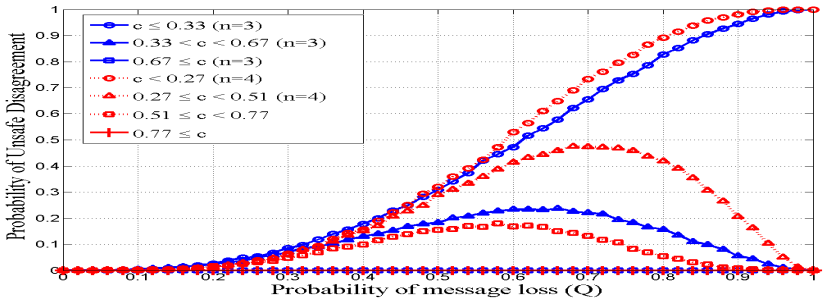


(a) Agreement on a leader

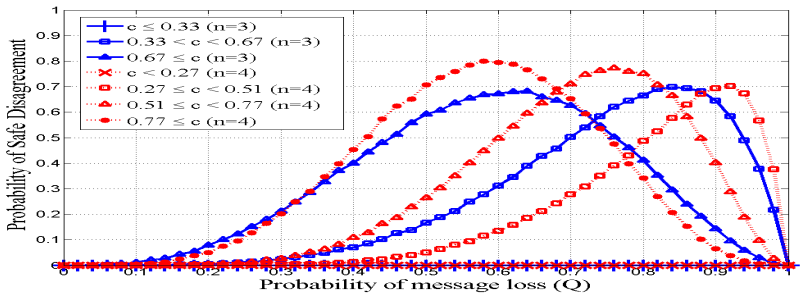


(b) Agreement to abort

Figure A.1: *Probability of agreement for the* ***optimistic**** *algorithm as a function of Q with different values of c for systems of $n = 3$ and $n = 4$ processes and $R = 2$ and $o_i = n$.*

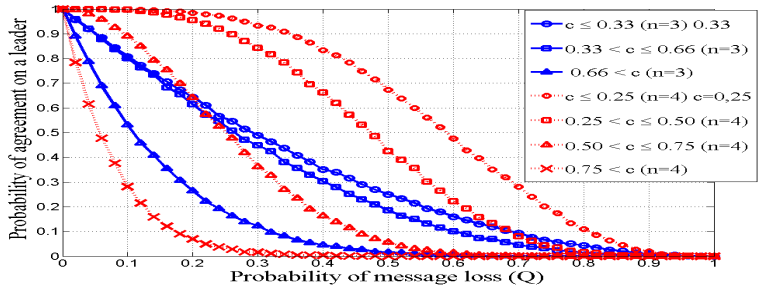


(a) Unsafe Disagreement

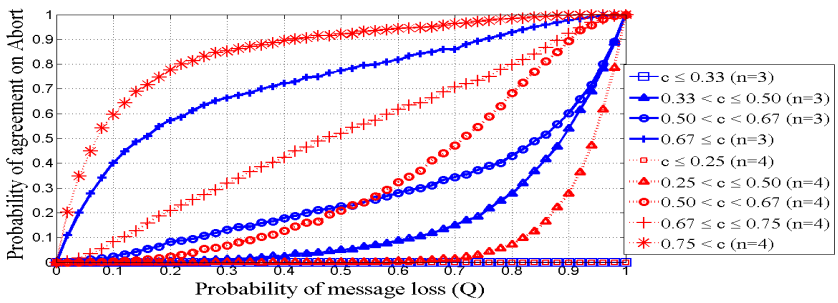


(b) Safe Disagreement

Figure A.2: *Probability of disagreement for the* ***optimistic**** *algorithm as a function of Q with different values of c for systems of* $n = 3$ *and* $n = 4$ *processes and* $R = 2$ *and* $o_i = n$.

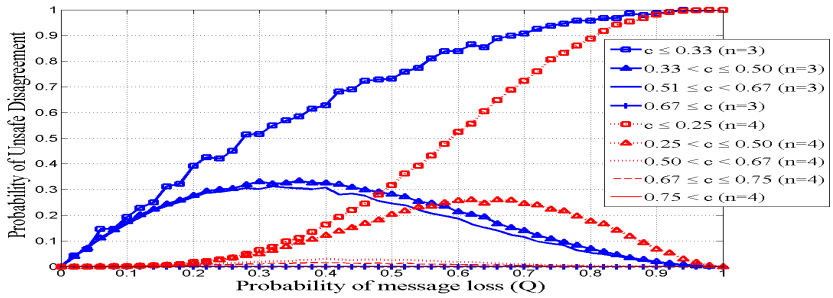


(a) Agreement on a leader

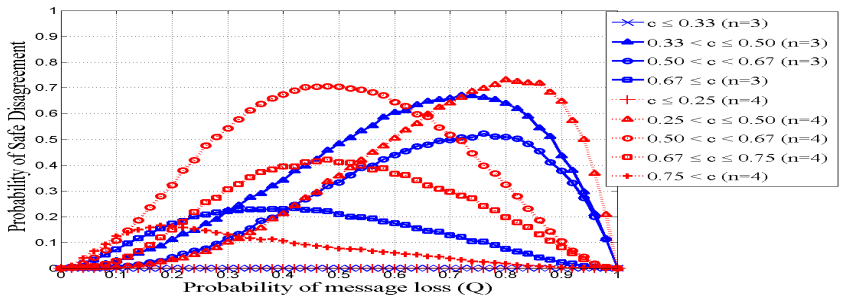


(b) Agreement to abort

Figure A.3: *Probability of agreement for the* ***moderately pessimistic**** *algorithm as a function of Q with different values of c for systems of $n = 3$ and $n = 4$ processes and $R = 2$ and $o_i = n$.*

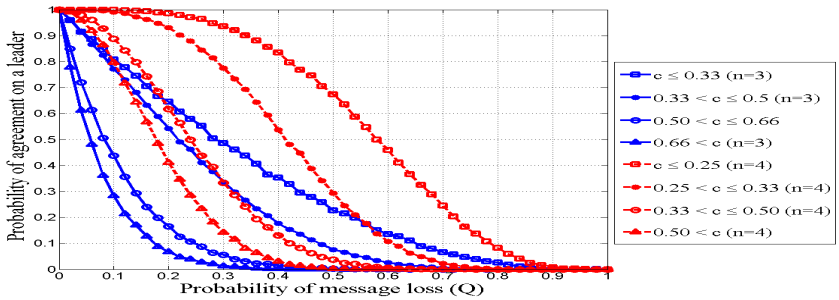


(a) Unsafe Disagreement

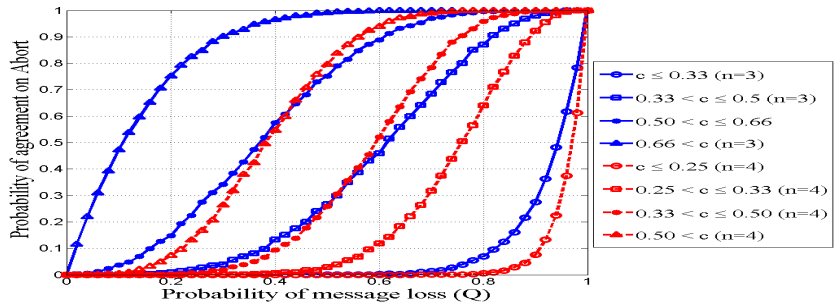


(b) Safe Disagreement

Figure A.4: *Probability of disagreement for the* ***moderately pessimistic**** *algorithm as a function of Q with different values of c for systems of* $n = 3$ *and* $n = 4$ *processes and* $R = 2$ *and* $o_i = n$



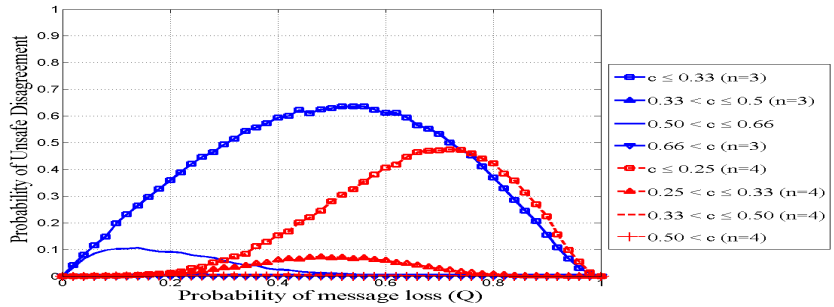
(a) Agreement on a leader



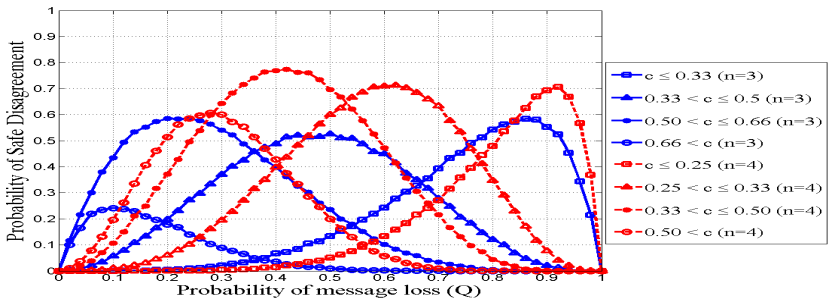
(b) Agreement to abort

Figure A.5: *Probability of agreement for the* ***pessimistic**** *algorithm as a function of Q with different values of c for systems of* $n = 3$ *and* $n = 4$ *processes and* $R = 2$ *and* $o_i = n$

For the explanation of the results refer to Fig. 4.5 and Fig. 4.6 in Section 4. $R = 2$ is a specific case for the *pessimistic* and *moderately pessimistic* decision criteria. For $R = 2$, increasing the size of the system we have lower probabilities of disagreement for the *1-of-n* selection algorithm. Here, we see the same trend for the *1-of-** selection algorithm. So, it must be interesting to compare the outcomes of the *1-of-** selection algorithm for larger values of R and different sizes of the system.



(a) Unsafe Disagreement



(b) Safe Disagreement

Figure A.6: *Probability of disagreement the* ***pessimistic**** *algorithm as a function of Q with different values of c for systems of $n = 3$ and $n = 4$ and $R = 2$ and $o_i = n$*