Aalto University
School of Science
Degree Programme in Computer Science and Engineering

MARCOS U. TONG ALVAREZ

Weak models of wireless distributed computing

Comparison between radio networks and population protocols

Master's Thesis
Espoo, August 2, 2016

Supervisor:	Jukka Suomela
Advisor:	Remberto Martinez

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Aalto University

ABSTRACT OF MASTER'S THESIS

Author:	MARCOS U. TONG ALVAREZ		
Title:	Weak models of wireless distributed computing Comparison between radio networks and population protocols		
Date:	August 2, 2016	**Pages:**	112
Major:	Foundations of Advanced Computing	**Code:**	T-110
Supervisor:	Jukka Suomela		
Advisor:	Remberto Martinez		

This thesis compares weak distributed computing models that are suitable for extremely limited wireless networks. The comparison is mainly between multiple variations of radio networks and population protocols. The analysis is based on model features, computability and algorithmic complexity. The thesis analyses essential and optional model features, and organizes the models accordingly. It discusses the applicability of results from stronger models to radio network models, including impossibility results, algorithms and their runtime. It analyzes different radio network algorithms for the classical problems in terms of their features, and it discusses their applicability to other radio network models. It reviews the fundamental differences between population protocols and radio networks. Lastly, the comparative analysis summarizes fundamental differences and separating features.

Keywords: theory of computing, distributed algorithms, computational complexity, weak models, radio networks, population protocols

Language: English

Acknowledgements

This thesis defines a major milestone in my life and professional career. It culminates the achievement of my master's degree in Foundations of Advanced Computing. It is the result of a large amount of dedication and effort. I would like to thank the persons that contributed to this work.

Thanks to my supervisor Jukka Suomela: For your professional work in supervising the topic, contents and results of the thesis. For your time and for making yourself available whenever was necessary. For your enlightening feedback and extensive corrections. For motivating me with your passion for making computer science.

Thanks to the lecturers and professors of the master's degree: For your contribution to develop my knowledge, skills and critical thinking.

Thanks to my advisor Remberto Martinez: For your valuable support and feedback.

Thanks to my parents Juan Manuel Tong Gastélum and Yolanda Álvarez Tamayo: For your constant encouragement and motivation for doing great things.

Thanks to my wife Svitlana Holyk: For your unconditional support at all times. For your encouragement to continue. For your patience in busy times. For being my front passenger in this journey.

Thank you all for being part of this work.

Sincerely,

Marcos U. Tong Alvarez

Espoo, August 2, 2016

Abbreviations and Acronyms

Abreviations

adv.	Adversary or adversarial
e.g.	For example
i.e.	It is; in other words
ref.	References
s.t.	Such that
std.	Standard
u.a.r.	Uniformly at random
w.p. 1	With probability precisely 1.
w.h.p.	With high probability. The probability goes to 1 as n goes to infinity.

Acronyms

BC	Broadcasting problem
BE	Beeping model
CD	Collision detection
FSM	Finite state machine
LE	Leader election problem
MIS	Maximal independent set problem
MP	Message passing model class
nFSM	Networked finite state machines model
PN	Port numbering model
PP	Population protocol model class
RN	Radio network model

Contents

Chapter 1

Introduction

This chapter introduces the thesis motivation and context. The first section presents the broader research area, motivation and basic terminology. The second section describes the context of the topic.

1.1 Motivation

Distributed computing networks are a relevant research area with applications in telecommunications and biology. Models of distributed computing are essential for understanding the computational power of distributed networks. Studies of distributed networks evolved from parallel network of processors into distributed networks of computing units. Gradually, with the development of the Internet and telecommunication networks there was an increasing interest in understanding distributed computability. Recently the attention has turned to wireless networks, in which communication conditions and node capabilities are dynamic and limited. Modeling computation on these networks is relevant for practical and novel applications.

Different distributed computing models have been developed, motivated by the need to understand computation in real-world networks. Numerous models have been developed and modified to reflect more closely the reality of these networks. In this regard, distributed computing models have been defined in terms of limitations and assumptions, which are referred to as *model features* in this work. A precise definition of model features is crucial, since the research findings in a particular model could differ with any small change to model features. Conversely, certain assumptions about the underlying network are optional modeling choices, needed both for abstracting irrelevant aspects or for narrowing down the computability analysis.

Model features and assumptions are the basis for the comparative analysis of this thesis. This thesis studies two classes of wireless models, namely radio networks (RN) and population protocols (PP). Both model classes were designed considering the limitations of real wireless networks. The two models are fairly different at first glance. Separating their computational power is neither evident nor trivial. Multiple variations of these classes have been used to study distributed algorithms. Comparing the results from these studies is challenging, since they have considered different features and assumptions. Previous literature [26, 32, 54] has made isolated comparisons based on particular features. However, literature on comparing and classifying the models shall develop further and this work contributes in this regard.

1.2 Context

This thesis is placed in the context of theoretical computer science, computational complexity and distributed algorithms. The focus of this thesis is comparing the computational power of two *weak models* of distributed computing. It is pertinent to clarify that strong models make stronger assumptions, which in fact are more restrictive from the modeling and computational point of view. Conversely, "weak" models are said to relax those assumptions and limitations. Consequently, weaker models are preferred over stronger models. This particularly holds for networks of agents with limited computation and communication capabilities.

Concretely, this thesis deals with wireless distributed computing. A clarification of this aspect is justifiable, since there are different connotations of the word "wireless". It can evoke the lack of physical communication medium, such as cables and other transmitting material or substances. In this thesis, *wireless* refers to its meaning in the context of theoretical distributed computing and modeling, namely having unlabeled connections. This has implications for the model capabilities, removing the ability to transmit messages directly to particular recipients. In radio networks, computing units have to send the same message to other reachable units (i.e. broadcast). Similarly, in population protocols computing units do not send messages directly to others. Instead they exchange state information during interactions that happen dynamically.

The remaining chapters are organized as follows: Chapter 2 includes definitions of main concepts for audience that is not familiar with the topic. It also includes definitions for the model features and problems studied in this thesis. In addition, it presents the main research questions of this thesis. Chapter 3 reviews results for the port numbering model. This stronger wired model is a natural starting point for the analysis for the weaker radio networks models. Chapter 4 reviews results for multiple variations of radio networks models. It includes tables summarizing the computational complexity results. Additionally, it discusses the applicability of each algorithm to other models. Chapter 5 reviews results for population protocol models. Chapter 6 compares the studied models based on the main model features. Finally the Chapter 7 provides an overall summary of results, contributions and open questions.

Chapter 2

Definitions

This chapter provides definitions that are used throughout this thesis. Section 2.1 defines basic concepts of distributed computing. Section 2.2 defines in more detail the elements of distributed computing models. Readers that are familiar the topic can safely proceed to Section 2.3, which includes definitions of the essential and optional model features. Section 2.4 and Section 2.5 provide definitions of the two major model classes. Section 2.6 defines the distributed computing problems studied in this thesis. Lastly, Section 2.7 presents the research questions and Section 2.8 describes the methodology.

2.1 Core concepts

A *computing network* is a system of independent computing units and communication links between them. Computing units, also known as nodes, have local computation capabilities and the ability to communicate with a set of neighboring units. Communication links are the channels or connections through which computing units exchange information. In *distributed computing networks* each individual computing unit has a limited or partial view of the entire network. These units have access to local memory only, as opposed to parallel computing networks, in which computing units have access to a shared or global memory. *Distributed computing* studies computation in distributed computing networks. Two central concepts in distributed computing are distributed computing problem and distributed algorithm. A distributed computing *problem* defines a computing problem in distributed networks, along with a particular network or family of networks to be solved on. A distributed *algorithm* or protocol defines the steps, operational rules and conditions for solving a problem using a particular model. The definition of an algorithm includes the inputs, computational steps, state transitions and outputs for each computing unit.

A *model* of distributed computing is the abstract representation of the elements and operation of distributed computing networks. A model is used to define the initialization, communication and protocols running in a network. Throughout this thesis, a distributed computing model is simply referred to as a "model". Models are formally defined in terms of *modeling elements* using a particular notation. Typically the elements of a model include the sets of inputs, states, outputs, transitions and symbols of an alphabet for encoding information. *Inputs* represent the input data received by the protocol, and they are an optional part of a problem. *Outputs* are the results of the computation. Each node has local inputs and outputs, which all together comprise the global inputs and outputs, respectively. Both inputs and outputs are encoded in some alphabet, which may be the same or different.

Nodes run a finite state machine (FSM), which is represented by a set of *states* and a set of *transitions*. A state is a snapshot of a node's memory at any given time. A node can only be in one state at a time. For flexibility, models also define input states and output states, which are the subsets of the state set in which nodes are able to accept inputs and produce outputs, respectively. A *transition* function defines the rules for choosing the new state of the node. A transition is usually a function of the information exchanged and the current states of a node and its neighbors. A *configuration* represents a snapshot of the network, in particular the global state, which is composed of the states of the individual nodes. The sequence of configurations from the start of an algorithm for solving a problem instance is called an *execution* or a *computation*.

Each distributed computing network has an underlying graph. The graph representation of a network has one node per computing unit and one edge per communication link. In this thesis, the term "*node*" refers to a computing unit. An example of a graph is shown in Figure 2.1. The same graph can be represented in set notation as $G = (V, E)$, where $V = \{1, 2, 3, 4\}$ and $E = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{2, 4\}\}$. A graph family is a set of graphs sharing a particular property. Examples of graph families are the family of cycle graphs, the family of bipartite graphs and the family of complete graphs. Both in previous literature and in this thesis, solving a problem in a graph family is equivalent to solving a problem in networks with underlying graph belonging to such family. Lastly, graph properties are commonly used in the study of distributed algorithms. These include the network size n (i.e. the number of nodes), the maximum degree Δ of any node in the network and the network diameter D, which is the number of connections separating the most distant nodes. The properties for the example graph of Figure 2.1 are $n = 4$, $D = 2$ and $\Delta = 3$. Appendix A includes relevant concepts of graph theory.

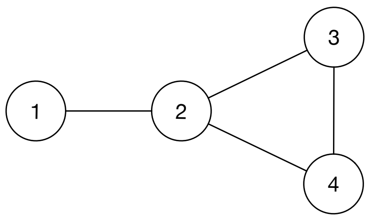

Figure 2.1: Example of a graph-based network.

2.2 Model elements and capabilities

The computational power of a model includes both the *computability* and *complexity* of algorithms for solving specific problems. Computability refers to whether a given problem can be solved at all in a certain model. Complexity refers to the resources such as time, memory or state set size, required by algorithms to solve a problem in a model. In this thesis, both computability and complexity aspects of a model are collectively called *model capabilities*. Complexity analysis of distributed algorithms is represented in a fashion similar to the complexity of centralized algorithms. An upper bound on the algorithmic complexity is represented as $\mathcal{O}(f)$, a lower bound as $\Omega(f)$ and a tight bound as $\Theta(f)$. Typically f is a function of n, D and/or Δ.

A key research interest of previous literature and of this thesis is the separation and characterization of model capabilities. Model separation is a technique for analysing and highlighting the differences in model capabilities, by finding problems that are solvable (efficiently) in one model and not solvable (efficiently) in other models. Model characterization is the formal and precise delimitation of model capabilities, including computability of mathematical function families and runtime bounded by specific functions.

A relevant distinction exists between *deterministic* and *non-deterministic* distributed algorithms. This is similar to the corresponding distinction for centralized algorithms. Given the same inputs and network, deterministic algorithms are guaranteed to follow the same computation and reach the same output. Conversely, non-deterministic algorithms might follow different executions and produce different outputs. This is reflected in the number of outputs the transition function has for a given input. For each source state there is only one target state in deterministic algorithms, whereas for non-deterministic algorithms there may be multiple target states. *Randomized* algorithms are a particular type of non-deterministic algorithms. Probabilistic analysis of algorithms studies the transitions of non-deterministic algorithms. Sometimes assumptions are made with regard to the probability distribution of such transitions, typically a uniform distribution is assumed. A typical acceptance criteria for randomized algorithms is with high probability (w.h.p.), which has a formal meaning of $1 - 1/n^C$ for some constant C. Finally, two types of randomized algorithms are Las Vegas and Monte Carlo algorithms. The former produces correct outputs w.p. 1 after an expected probabilistic runtime. The later might produce correct outputs with some probability while it has a characterized runtime w.p. 1.

2.3 Model features

This thesis makes instrumental use of the concept of *model feature* for comparing models. A model feature is a distinctive attribute or aspect of a model. Models have both essential features and optional features. Essential features are an intrinsic part of the model definition. Optional features are choices that specialize a model for particular purposes.

2.3.1 Essential features

The **communication mechanism** is the way nodes exchange information. Two different mechanisms are messaging and interactions. Messaging is representative of those networks where symbols are sent and received through communication links. Interactions are fundamentally different from messaging in the sense that communication links are not predefined, they are rather established dynamically.

The **scheduling policy** defines the timing in which communication happens. Two separating policies are probabilistic and adversarial. Probabilistic scheduling is based on a probability distribution, a typical choice is a uniform distribution. Adversarial scheduling defines an adversary controlling the scheduling of transitions.

2.3.2 Optional features

The **synchrony** feature defines the timing in which communication happens either synchronous or asynchronous. In a synchronous model all nodes communicate at the same discrete time slots (rounds). In an asynchronous model nodes can arbitrarily communicate at different times. An alternative definition of this feature classifies models with global and local synchronization. In global synchronization, also known as global clock, the round counters of all nodes are aligned. In local synchronization, also known as local clock, nodes keep a (different) local round counter.

The **collision detection** (CD) feature applies to models in which communication happens through a shared channel, which is exposed to collisions. This is the case of radio networks and more generally broadcast networks, in which nodes are not able to direct messages to particular neighbors. Collision detection defines whether nodes are able to distinguish collisions from background noise or not. In the later case, the node cannot distinguish a round with collision from a round with no messages received. Carrier sensing is an additional class defined by this feature, in which nodes are only able to detect whether transmission is ongoing or not, for instance by detecting presence of energy in the communication channel.

The **activation policy** feature defines the mechanism for enabling of nodes, such that they start their execution. Two activation aspects considered in previous literature are spontaneity and simultaneity. Spontaneity defines whether nodes are enabled by themselves or require some external factor to enable them. Typically models in previous literature have defined non-spontaneous activation. Simultaneity refers to whether nodes are enabled at the same time or not. A fully synchronous model defines both global clocks and simultaneous activation. Non-simultaneous models are further separated depending on whether nodes are activated by incoming messages or by an adversary. Adversarial scheduling implies adversarial activation, since for any node the adversary may decide not to enable communication until an arbitrary later time. Conversely, models that define adversarial activation do not necessarily define adversarial scheduling.

2.3.3 Model assumptions

In addition to model features, there are model assumptions particular to each model. The following assumptions may restrict both the comparability of results and their applicability to other models:

1. The **anonymity** assumption refers to whether nodes have unique IDs or are anonymous.
2. The **access to random bits** assumption refers to nodes being able to generate random bits.
3. The **graph family** assumption restricts the model to corresponding networks. Appendix A provides a list of graph families.
4. A set of **bounded resources** assumptions refer to limitations on the global or local resources. Models define them either as parameters or as functions of other properties such as the network size n. Local attributes include local memory, message queue size and state set size. Global attributes include bandwidth, message size and communication alphabet size.

2.3.4 Modeling dimensions

Modeling dimensions define a containment order of classes, similar to classes of complexity theory. The containment order indicates that the set of solvable problems in certain subclass is a subset of the set of solvable problems in its containing class. Different dimensions have been defined in previous literature, including uniformity [60], correctness [29], class of state machine [37] and adversarial strength [34].

The **uniformity** dimension refers to the local knowledge of global attributes, in particular the network size. Uniform protocols are those that are described independently of the network size. Both positive and negative results were proven by Chalopin et al. in [18], depending on complete or partial knowledge of the network size. Different model classes where characterized by Yamashita and Kameda in [60] based on the local knowledge. The classes include local knowledge of:

1. Nothing about the network size nor its topology (uniformity) D_{no}.
2. An upper bound on the network size D_{upb}.
3. The accurate network size D_{acc}.
4. The topology of the network D_{top}.

The network topology does not include node IDs. This means that nodes may know the structure of the network, while still being unaware of the particular node they represent and neighborhood they belong to. A containment relation was proven by Yamashita and Kameda in [60]:

$D_{no} \subseteq D_{upb} \subseteq D_{acc} \subseteq D_{top}$. Depending on the problem there is strict a containment or equivalence relationship in model capabilities.

The **correctness** dimension refers to restrictions in the correctness of global outputs. Four classes have been defined based on the correctness requirement. The strongest and weakest classes were studied by Itai and Rodeh in [38]. The two intermediary classes were presented by Emek in [29]. Starting from the strongest requirement, the classes are:

1. Terminal correctness (TE) requires nodes to terminate at the time they produce the first outputs and these outputs must form a correct and stable global output.
2. Write once outputs (WO), in which the first local outputs must form a correct and stable global output, while the nodes are allowed to continue communicating and transitioning states.
3. Rewritable outputs (RW) does not require nodes to terminate. Nodes can write and rewrite local outputs until a first global output is produced, which must be correct and stable.
4. Eventual correctness (EC) neither requires nodes to terminate nor the first global output to be correct. Nodes can write and rewrite local outputs as long as the global output eventually converges to a correct solution.

A separation of problems based on this dimension was presented in [29], along with this containment order: $TE \subseteq WO \subseteq WR \subseteq EC$.

The **classes of state machine** presented by Hella et al. in [37] separate state machines according to the collection of inputs and outputs. Input collection classes include a vector, a set and a multiset of input symbols. Similarly, output collections classes include a vector of symbols (a symbol per outgoing connection) and a single symbol broadcasted to all reachable neighbors. Different model classes were characterized in [37] based on the combination of input and output collection types. These are weaker models derived from the port numbering (PN) model, referred to as class VV_c. The classes are:

1. VV input ports and output ports are not necessarily connected to the same neighbor.
2. MV input ports are not numbered and algorithms receive a multiset of messages.
3. SV input ports are not numbered and algorithms receive a set of messages.

Feature	Message Passing	Population Protocols
Communication Mechanism	Port connection forming a topology, messages sent through ports.	Dynamic interactions typically in pairs.
Correctness Dimension	Either terminal or eventual correctness.	Eventually converge to a correct solution.
Scheduling Policy	Randomized scheduling is the typical choice in MP models. Uniform distribution is the typical choice.	Adversarial scheduling is inherently defining the interactions. Fairness condition on reachable configurations.

Table 2.1: Essential model features in MP and PP model classes.

4. VB output ports are not numbered and algorithms broadcast the same message to all ports.
5. MB is the combination of MV and VB.
6. SB is the combination of SV and VB.

A containment order was presented by Hella et al. in [37], such that $SB \subsetneq MB = VB \subsetneq SV = MV = VV \subsetneq VV_c$.

The **adversary strength** dimension was motivated by the study of link reliability in wireless radio networks. This feature is out of the scope of this thesis.

Two major model classes from which many others derive are message-passing (MP) and population protocols (PP). Essential features are embodied in the two major classes, and they are summarized in Table 2.1. The following two sections introduce definitions for the two major model classes and derived models.

2.4 Message-passing models

In **message-passing** (MP) models nodes communicate by exchanging messages through communication ports. Early MP models [4, 16, 55] were defined in terms of parallel computing in networks of processors, however computability results have been extended to distributed computing. A network is represented as an undirected graph $G = (V, E)$ with static topology. Each processor i is connected to a set of neighbors

V_i, with which communication is enabled. These models described networks of processors running the same protocol, possibly of infinite states Z_i. Processors communicate by exchanging (passing) messages with their neighbors and communication happens in synchronous rounds. In a single round each processor i receives messages m_ℓ from its neighbors $\ell \in V_i$, does some processing, sends a (possibly different) message $m_{i\ell}$ to each neighbor and transitions to another state. Message delivery is assumed to follow order in which they were sent, guaranteed by an incoming message queue. Nodes are activated simultaneously upon receiving a special "START" message. In the plain MP model processors have an identity and know the connected links, however they do not know the neighbor identities nor any other global attributes, such as the size of the network.

The **Port Numbering** (PN) model introduced in [4] is probably the most typical anonymous MP model from previous literature. In the PN model networks are represented as a graph $G = (V, E)$ with locally labeled connections. The connections are defined by a *port numbering* function $f_v \colon \{u, v\} \to Z_d$ for each node $v \in V$ of degree $d = \deg_G(v)$, that maps incident edges $\{v, u\} \in E$ to labels $Z_d = \{1, 2, \ldots, d\}$. Given a set of states Q and alphabet A, the communication function $M \colon Q \times Z_d \to A \times A \times Q$ defines for $(q, i) = (a, b, q')$ the possible transitions from state q to q' by exchanging message a by b on port i. In this model, the family of graphs is unrestricted. Inputs and outputs are defined as vectors ordered by the port numbers. Furthermore nodes are anonymous, communication is synchronous and correctness is terminal. Most literature of MP models assume a simple PN network, which is a PN network without self loops and without multiple connections between any two nodes. Figure 2.2 shows an example of simple PN network, which has as underlying graph the one from Figure 2.1. In this example, the port numbering function is defined as $f_1 \colon \{\{1, 2\} \to 1\}$, $f_2 \colon \{\{2, 1\} \to 2, \{2, 3\} \to 1, \{2, 4\} \to 3\}$, $f_3 \colon \{\{3, 2\} \to 2, \{3, 4\} \to 1\}$ and $f_4 \colon \{\{4, 2\} \to 2, \{4, 3\} \to 1\}$. A relevant aspect to highlight is that nodes were labeled for illustrative purposes, however they are anonymous in the PN model. Variants of PN include: (a) the LOCAL model, which adds unique identifiers, (b) the CONGEST model, which limits the message size, and (c) the *randomized PN*, in which nodes have access to randomness.

Wireless models of distributed computing were motivated by wireless networks including local area networks and packet radio networks. The following subsection presents definitions for the most typical RN models in previous literature.

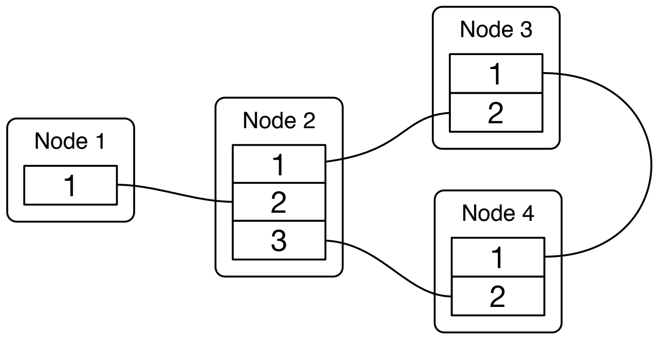

Figure 2.2: Example of a simple PN.

2.4.1 Radio network models

The **Radio Network** (RN) models are based on a shared channel over which all communication is carried. As opposed to PN, in RN nodes are unable to direct messages to specific neighbors, instead nodes broadcast the same message to all neighbors (nodes in the transmission range). Nodes either transmit or receive at any given time. A collision occurs in case two nodes in the same transmission range decide to transmit at the same time.

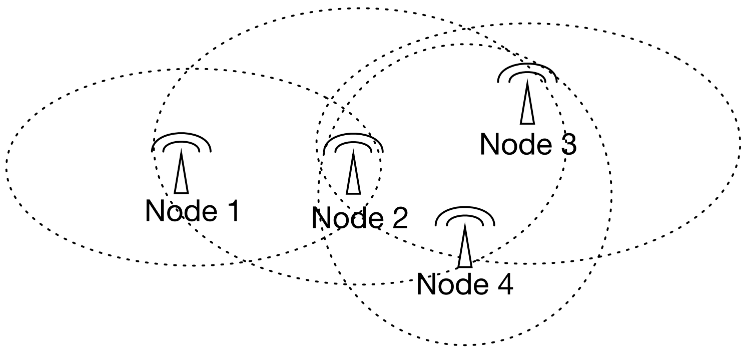

Figure 2.3: Example of a simple multi-hop RN.

RN models are classified according to transmission range, collision detection (CD), synchrony and activation policy. Depending on the transmission range nodes can send messages directly to any other node or require message routing [36]. In *single-hop* networks all nodes are in the same transmission range. Single-hop RN networks are representative of complete graphs. Conversely, in *multi-hop* networks there are nodes out of the range of other nodes. Figure 2.3 shows an example of multi-hop RN, which has as underlying graph the one from Figure 2.1. Another classification of RN is based on collision detection. In RN with CD collisions are represented as nodes receiving a “noise” message, whereas in RN without CD no message is received (i.e. the channel is silent). The *standard* RN considers multi-hop networks with CD [20]. Conversely, other models consider the case without CD [13, 28], which is referred to as *silent* RN model in this thesis. Both of these variations assume synchronous communication and simultaneous activation. In addition, asynchronous and non-simultaneous models have been studied in [25, 30, 49]. Figure 2.4 depicts a feature map of RN models.

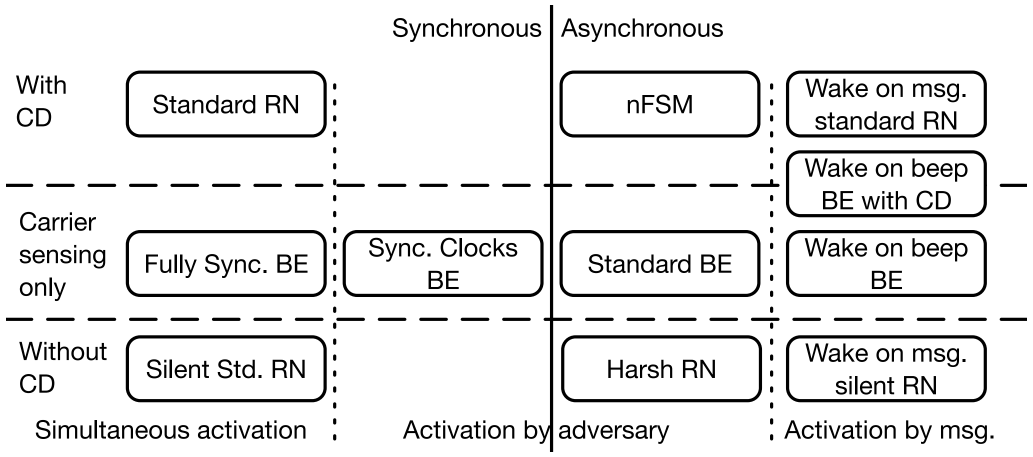

Figure 2.4: Feature map of RN models.

A weaker version of the RN model known as *harsh* RN model, was defined in [49]. This is a multi-hop asynchronous RN without CD. In this model an adversary activates nodes, as opposed to nodes being activated by incoming messages. In this model nodes do not know the network size and the message size is limited to $\mathcal{O}(\log n)$. This model has been applied to restricted graph families, with the objective of representing more closely the reality of RN, including: (a) *Unit disk* graphs ($\mathcal{UD}$), in which nodes are placed in an Euclidian space (i.e. two-dimensional) such that there is a connection between the nodes that are within one unit distance of their circular transmission range [49]. (b) *Bounded growth* graphs ($\mathcal{BG}$), which are a generalization of $\mathcal{UD}$ graphs based on a metric space with constant doubling dimension [53]. (c) *Bounded independence* graphs ($\mathcal{BI}$), which are another generalization of $\mathcal{UD}$ graphs such that the size of the 1-hop and 2-hop maximal independent sets (MIS) are bounded by two variables κ_1 and κ_2 [50].

The **networked FSM** (nFSM) model introduced in [30] is a type of RN model. It defines a graph based $G = (V, E)$ broadcast network, where each node $v \in V$ runs an instance of the same FSM and communicate asynchronously with a set of neighbors $N(v) \subseteq V$. There is a communication alphabet Σ defining the possible transmitted messages $\sigma \in \Sigma$, a finite set of states Q and subsets of input states $Q_I \subseteq Q$ and output states $Q_O \subseteq Q$. Messages sent in step t from a neighbor $u \in N(v)$ to v are stored as a multiset of inputs $\psi_u(v)$, such that they are guaranteed to be delivered in the same order sent by u after a delay $D_{v,t,u}$. Asynchrony is represented by the length $L_{v,t}$ of a step $t \in \mathbb{Z}_{>0}$ of a node v. This model is defined with adversarial scheduling, in which an adversary controls the transmission delays and step lengths. States are mapped to a "query letter" by a querying function $\lambda\colon Q \to \Sigma$, such that at each state nodes are only able to utilize the received messages matching the corresponding letter. A bounding parameter b is a key characteristic of nFSM, along with a bounded counter function $\sharp(\sigma)$ of the symbol σ in the multiset of received messages of v. The transition function $\delta\colon Q \times B \to 2^{Q\times(\Sigma\cup\{\epsilon\})}$ maps the current state Q and bounded counter $B = \sharp(\lambda(Q))$ to the next states and query letters. The authors of [30] presented a synchronizer protocol and utilized it in their algorithmic solutions.

Lastly, the **beeping model** (BE) introduced in [25] is another type of RN model. The BE model is a subtype of RN in which node communication is entirely based on beeps. Nodes can either be silent or beep. Collision detection is not available as such, nodes can only distinguish between no neighbor beeping or at least one neighbor beeping. The model does not assume unique IDs nor knowledge of global attributes. This model is closely related to nFSM, as noted in [30], the BE model is equivalent to nFSM with bounding parameter $b = 1$. This model considers adversarial node activation and asynchronous local clocks. The model considers discrete time intervals, such that beeps occur at discrete slots and last precisely for the duration of a slot. A variant of BE with a global clocks was presented in [1]. A parameterized variant of BE was presented in [35]. It includes parameters for the probabilistic precision, an lower bound of the network size and an upper bound on the number of states. This variant was used to study state complexity, and a trade off was found to exist between state set size and error precision. In general BE is one of the weakest RN models.

2.5 Population protocol models

Although **Population protocol** (PP) models are a wireless model, they are significantly different from MP models. In PP models nodes communicate through pair interactions that happen on proximity. A population is represented as a directed interaction graph $G = (V, E)$, with a set of agents V and a set of possible interaction pairs $E \subseteq V \times V$. The edge direction indicates that one agent is the initiator and a different one is the responder. Edges do not represent established communication connections, since interactions happen dynamically. PP models were introduced in [7] with a formal definition that included input alphabet X, output alphabet Y and finite set of sates Q. A input function $I\colon X \to Q$ maps inputs to states, and an output function $O\colon Q \to Y$ maps states to outputs. Before an interaction, the pair of participating agents are in a pair states (p, q), where p is the state of the initiator and q the state of the responder. And after the interaction, the initiator and responder change to another pair of states (p', q'). A transition function $\delta\colon Q \times Q \to Q \times Q$ defines all possible interactions. A configuration is a function $C\colon V \to Q$ that maps agents to current states. The original model considered passively mobile agents, using the word "passive" to indicate that nodes have no control on their mobility. A fairness condition is inherent in the definition of PP, such that every possible interaction has a chance to occur infinitely often. The number of states is $\mathcal{O}(1)$. Since nodes run the same FSM, and they are anonymous (i.e. do not have unique IDs). The standard PP model assumes a complete interaction graph. A global signal activates all the sensors simultaneously. An adversary decides the scheduling of interactions. Figure 2.5 shows an example PP, with four nodes and their interactions during five times after initial activation t_0. The underlying interaction graph of this network corresponds to a complete graph.

The **stabilizing inputs** PP model was introduced by Angluin et al. in [5] with the objective of representing real-world scenarios more closely. Instead of having inputs defined at the beginning of the computation, inputs may change until they stabilize. This allows chaining protocol outputs as inputs of other protocols, facilitating the modeling of algorithmic reductions. Configurations and transitions are extended to include both a state and a symbol. The configuration function becomes $C\colon V \to (Q \times X)$ and the transition function becomes $\delta\colon (Q \times X) \times (Q \times X) \to Q \times Q$. An auxiliary projection function is used to obtain the state $\pi_1(C(v))$ and the input $\pi_2(C(v))$. Inputs stabilize

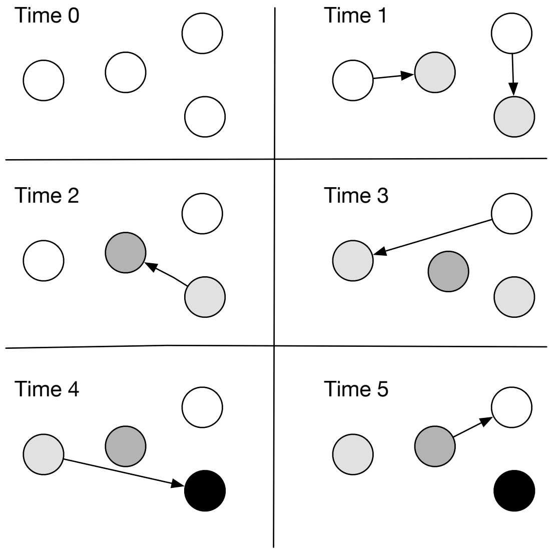

Figure 2.5: Example of a PP network and node interactions by proximity.

after a finite step k, if for any further step ℓ the inputs remains exactly the same $\pi_2(C_k) = \pi_2(C_l)$ for $l \geq k$. Additional concepts for the self-stabilizing PP model were introduced by Angluin et al. in [11]. A *trace* $T(Z)$ is an infinite sequence of assignments $\lambda_i \colon V \to Z$ for $i = 0, 1, \ldots$ and Z the trace alphabet. There are input traces for $Z = X$ and output traces for $Z = Y$. A *behavior* $B(Z)$ is a set of traces that have the same alphabet. Similar to traces, there is input behavior $B_{in}(X)$ and output behavior $B_{out}(Y)$, which ensure respectively that every input or output traces are contained in the behavior. A stable behaviour $B^S(Z) = ZB(Z)$ is a behavior preceded by a finite sequence prefix. Lastly, the *repetition closure* of a trace $T(Z)$ allows one or more repetitions of each assignment λ_i^+, and an *elastic behavior* is the repetition closure of a set of traces. The later concept is particularly useful for problem reductions, which can be defined in terms of self-stabilizing inputs and protocol composition.

The **probabilistic** PP model (also known as conjugating automata) was defined in [7]. As opposed to adversarial scheduling, it defines a probabilistic interaction scheduling with uniform distribution, also known as randomized interactions. Randomized interactions allow a fairness condition w.p. 1 as well as algorithms in which the output is correct only w.h.p. Numerous variations of probabilistic PP models have been defined in literature, some are of them are described Appendix B

Figure 2.6 depicts a hierarchy of PP models, including the standard, stabilizing and probabilistic PP. Subtypes of these models and the other PP model types are out of the scope of this thesis.

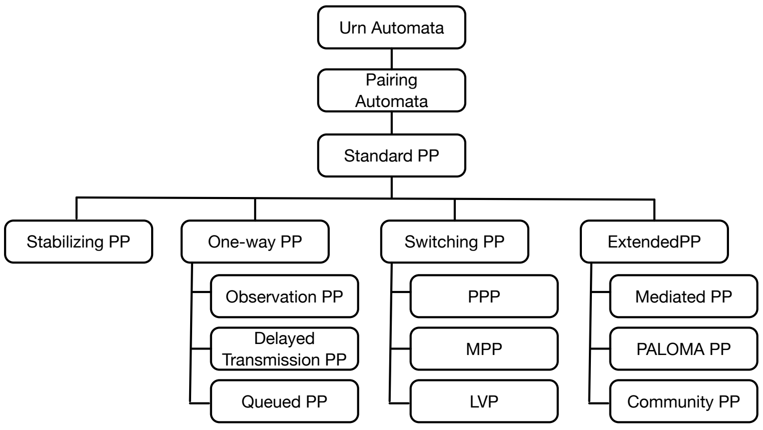

Figure 2.6: PP models hierarchy.

2.6 Problem definitions

Classical distributed computing problems have been repeatedly studied in previous literature. Table 2.2 defines the set of classical problems studied in this thesis. An essential part of a problem definition is the graph family in which the problem has to be solved. The reader can refer to Appendix A for a list of graph families.

2.7 Research questions

The main research question of this thesis is: *How comparable are the model capabilities of RN and PP?*

Concrete questions include:

1. Are the results from the different types of RN models applicable to other RN models?
2. How do model features affect the capabilities of RN?
3. What are the fundamental differences between RN and PP models in terms of model features?

Problem	**Input**	**Output**
Maximal independent set (MIS)	$I(v) = \emptyset$	$O(v) \in \{0,1\}$ s.t. $I = \{v : O(v) = 1\}$ is independent: $\{u,v\} \in E \to u \notin I$ or $v \notin I$ and maximal: $\nexists I'$ s.t. I' is independent and $I \subsetneq I'$
$(\Delta+1)$-coloring	$I(v) = \emptyset$	$O(v) \in \{1, 2 \ldots \Delta+1\}$ s.t. $\{u,v\} \in E \to O(v) \neq O(u)$
2-hop coloring	$I(v) = \emptyset$	$O(v) \in \{1, 2 \ldots \Delta(\Delta-1)+1\}$ s.t. $\{u,v\}, \{v,w\} \in E$ and $u \neq v$ and $u \neq w \to O(w) \neq O(u)$
Parity	$I(v) \in \{0,1\}$	$O(v) = 1$ if there are an odd number of 1's in the input, and $O(v) = 0$ if there are an even number of 1's in the input.
Majority	$I(v) \in \{0,1\}$	$O(v) = (\sum_{u \in V} I(u))/\lvert V \rvert$
Leader election (LE)	IDs	$O(v) = \mathsf{ID}_l$ for all $v \in V$ s.t. $I(u) = \mathsf{ID}_l$ for some $u \in V$
Anonymous LE	$I(v) = \emptyset$	$O(v) \in \{0,1\}$ s.t. $(\sum_{u \in V} O(u)) = 1$
Single source global broadcast (BC)	$I(s) = m$ for precisely one source node s, and $I(u) = \varepsilon$ for $u \neq s$, $u \in V$	All nodes must have received m. Nodes may terminate, and do not need to know whether BC has completed
Acknowledged broadcast	$I(s) = m$ for precisely one source node s, and $I(u) = \varepsilon$ for $u \neq s$, $u \in V$	All nodes must have received m. All nodes know that BC has completed before terminating

Table 2.2: Problem definitions.

2.8 Methodology

The present thesis performs a survey through a systematic literature review. The research process is outlined next:

1. Data collection. Searching by keywords in relevant sources such as journals of distributed and theoretical computing and academic publication search engines. The main keywords included the two major model classes: “radio networks” and “population protocols”.
2. Paper selection. Filtering papers based on the model features and problems they are solving. This thesis focuses in BC, MIS, LE and coloring. Studies for other problems such as maximal matching, vertex cover and consensus were left out. Adversary strategies were left out of scope. Refining search keywords by combining model features and problems with the two major model classes: “maximal independent set in radio networks”, “asynchronous radio networks”, ”leader election in population protocols”.
3. Initial data processing. Reading the abstract, introduction and model definition section. Extracting the addressed problems and model features, including collision detection, synchrony and activation policy. Listing the underlying model assumptions such as knowledge of global properties and graph family. Summarizing the information in tables.
4. Detailed data processing. Studying the algorithm and computational complexity results. Analysing the algorithm, transcribing it and complementing it with the following clarification types: (a) state variables, (b) constants, (c) state transitions, (d) flow control (conditions and loops). Complementing the tables created in the previous step with complexity results and references.
5. Analyse applicability. Assessing its applicability to related models, such as other RN and PP models.
6. Iterative data collection. Looping through related articles from the authors of the most relevant publications. Deep diving into relevant references from these publications.
7. Summarize and discuss. Creating comparative tables per model feature summarizing the computational complexity results. Discussing the underlying challenges derived from the choices of model features and assumptions.

Chapter 3

Port numbering model

This chapter provides a review of previous literature in the port numbering model (PN). PN is a natural starting point for studying the capabilities of message-passing models (MP). Section 3.1 reviews symmetry breaking aspects, which are central in the study of MP model capabilities. Two symmetry breaking mechanisms are discussed, namely unique IDs and randomization. Sections 3.2, 3.3 and 3.4 cover algorithms for solving maximal independent set (MIS), vertex coloring and leader election (LE), respectively. In general, these three sections focus on randomized algorithms.

3.1 Symmetry breaking

The concept of symmetry plays a crucial role in understanding computability limitations of MP models. The ability to distinguish different nodes is crucial for problem solvability. Since nodes in MP models run the same FSM, nodes that are given the same input are identical from the perspective of each node. As the execution of a given algorithm progresses, nodes are able to gather information from the network. However depending on the input arrangement and topology symmetry, nodes may or may not be able to break such symmetry (i.e. become distinguishable). The ability to break symmetry is impossible for anonymous MP models, in which nodes do not have labels (i.e. IDs) nor access to random bits. In distributed computing **local symmetry** can be defined as two nodes having isomorphic radius-t neighborhood at time t in the execution of a protocol. Given a pair of nodes with identical inputs and local symmetry all their state transitions as well as their output at time t will be identical. Consequently, it is impossible to solve problems in which solutions require nodes to output different values if local symmetry holds. A relevant research area in previous literature is characterizing problems and graph families in which it is possible to break symmetry.

A seminal study on symmetry breaking was presented by Angluin [4], which defines the problem in terms of finding a "centered" configuration in the PN model. Graph coverings are used for proving whether it is possible or not to reach a centered configuration in particular topologies and graph families. Findings in [4] include positive results for trees and complete graphs. For trees it is possible to find a centered configuration by following a bottom-up approach, in which leaf nodes send acknowledge messages and intermediary nodes wait for collecting the acknowledge messages from all children. Conversely, [4] includes negative results for the same problem in the family of cycle graphs, the family of graphs containing a triangle and graphs having a proper covering. In these graphs nodes are not able to break local symmetry nor to propagate a graph anomaly fast enough. These results apply for problems requiring a centered configuration, including vertex coloring, MIS and LE. Moreover the results extend to other MP models, including RN and BE models. The reader may refer to other symmetry breaking studies in [18, 38], which are also based on graph theory isomorphisms, such as isomorphism, automorphism and homomorphism (see Appendix A on graph theory concepts).

As opposed to anonymous MP, symmetry breaking for MP models with unique IDs is not a crucial challenge. Computability of MP models with unique IDs has been studied extensively in [44, 51, 55, 59]. A popular model representative of this feature is the LOCAL model, which extends the PN model with unique IDs. An upper bound of $\mathcal{O}(D)$ for any function in LOCAL was presented by Linial in [44]. That is the time it takes for all nodes to gather the entire information about the entire network, including node IDs and connections. Additionally, Linial presented in [44] three algorithms for solving the problems of MIS in directed cycles, 3-coloring directed cycles and coloring trees. Each of these three algorithms has runtime $\mathcal{O}(\log^* n)$. The algorithms followed the approach of collecting all data locally before proceeding with the processing. This reduces the analysis to combinatorial problems, for which the graph chromatic number is a relevant parameter. The chromatic number of a graph indicates the least number of colors for which a valid coloring is possible in the graph.

Randomization is another symmetry breaking mechanism. For complete graphs Angluin proved in [4] that it is possible to find a centered configuration using a randomized protocol, in which every node starts offering a coin-toss with all neighbors, expanding branches in a breadth first approach, and keeping track of the beaten nodes. Randomization can be as powerful as unique IDs when the network size n is known, since it can be used to build unique IDs w.h.p. [58]. The following sections review randomized algorithms for solving classical problems. These include Luby's algorithm for MIS [45], Johansson's $(\Delta+1)$-coloring algorithm [39], Itai and Rodeh's algorithm for LE in cycles [38] and Matias and Afek's algorithm for LE in arbitrary graphs [46]. All these algorithms rely on the assumption that n is known. In this regard, Yamashita and Kameda presented in [60] a separation of graph families, depending on their level of symmetry and the model uniformity (see modeling dimensions in Section 2.3.4). The level of graph symmetry is defined in [60] based on p-factorizations, e.g. cycle graphs have a 2-factor. Lastly, Yamashita and Kameda proved in [60] that an upper bound of the network size N, as opposed to the actual value, is enough to solve LE and spanning tree in tree graphs.

The concept of *locally checkable* problems introduced in [51] is closely related to local symmetry. The solution to these problems can be validated locally. Locally checkable algorithms running in constant time are defined as *local algorithms* in [59]. In that survey the author summarizes both positive and negative results, also known as *non-local* problems. The positive results include a local algorithm for weak 2-coloring. Impossibility results for cycles and other graph families include constructing a spanning tree and finding a stable matching. Classical problems of distributed computing include locally checkable problems such as MIS and coloring, as well as a non-local problems such as LE. The rest of this chapter reviews results for solving these problems in the PN, highlighting their applicability to RN. The complexity results are summarized in Table 3.1.

3.2 Maximal independent set

MIS algorithms were originally developed for parallel networks of processors [2, 45]. These algorithms consist of phases of two steps: (a) a choice step, in which processors become part of the MIS with a certain joining probability, and (b) a validation step, which defines criteria to prevent two adjacent processors joining the MIS. At each phase, an auxiliary independent set is built and then combined with a set holding the partial result. A Las Vegas algorithm was presented by Alon et

Problem	**Assumptions**	**Runtime**	**Ref.**
MIS	Parallel network of processors	$\mathcal{O}(\log n)$	[2, 45]
MIS	Synchronous PN	$\mathcal{O}(\log n)$	[48]
Coloring	Synchronous PN, memory $\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	[47]
LE	Known n, synchronous PN	$\mathcal{O}(n \log n)$	[12]
LE	Known n, asynchronous PN	$\mathcal{O}(n^2)$	[12]
LE	Known n, asynchronous PN, eventual correctness	$\mathcal{O}(n \log n)$	[12]
LE	Known n	$\mathcal{O}(n \log^2 n)$	[38]
LE	Known n, random bits, generated IDs of size $\mathcal{O}(\log n)$	$\mathcal{O}(D)$ w.p. $> 1 - \varepsilon$	[46]

Table 3.1: Complexity of classical problems in PN.

al. [2]. Each node u with $d_u > 0$ is marked w.p. $1/d_u$. Nodes with $d_u = 0$ are always marked. In case an edge is incident to two marked nodes, one of them u is unmarked w.p. $d_u/(d_u + d_v)$. A Monte Carlo algorithm was presented by Luby in [45]. It is also known as Luby's permutation algorithm, and it follows an approach that is suitable for distributed networks: processors generate a random priority number locally and nodes with local minima are chosen to join the MIS. However a different approach is required, since distributed networks do not have a shared memory.

Luby's permutation algorithm has been adapted to work in the PN model [1, 48]. The auxiliary and partial result vertex sets stored in shared memory are replaced by a set of active neighbors stored in local memory. This allows nodes to perform both the validation and the choice steps locally. Transmitting large priority numbers between nodes represents another challenge for the distributed adaptations of Luby's algorithm. A direct application of Luby's permutation algorithm in the PN model requires making the following strong assumptions: (a) Message size $\mathcal{O}(\log n)$ for exchanging priorities, (b) knowledge of (a bound of) the network size n for generating the priorities, and (c) synchronous node activation and round execution.

An improved adaptation presented by Métivier et al. [48] drops the first two assumptions. Informally, the algorithm simulates exchanges of real number priorities through exchanges of bit strings. Algorithm C.2 is an abstraction of the original algorithm. Figure 3.1 depicts the state transitions, with state set $\{C, I, M, S\}$ corresponding to candidate, ineligible, MIS and slave. Nodes generate a random bit $b \sim \{0, 1\}$ per step. The concatenation of the generated bits forms a bit string equivalent to a random real number. However the algorithm does not require nodes to hold the bit string in memory. Instead, nodes only keep the latest bits exchanged and the current states of the neighbors. Nodes detect local minima in case they generate $b = 0$ and do not receive any message with $b = 0$. These nodes acknowledge their neighbors before joining the MIS. Nodes that receive such acknowledge move to state 'slave'. Nodes that generate $b = 1$ and receive a $b = 0$ realize they are not local minima. These nodes move to the ineligible state, and they wait for the next phase in order to compete again. Each node requires memory of size Δ, to store the state of its neighbors. The algorithm is applicable to broadcast networks, since at each step nodes send the same message to all neighbors. However the algorithm is not directly applicable to RN networks, since it relies on a fully synchronous and collision free model. The small message size requirement makes this algorithm adaptable to models that assume small memory and/or message size, including the nFSM and BE models. However, special considerations are required for coping with non-simultaneous activation and asynchronous scheduling. Non-simultaneous activation requires MIS nodes to keep sending their state, in order to prevent newly active neighbors from joining the MIS. Lastly, the algorithm relies on nodes being able to gather all neighbor bits generated at the same round. Although this works naturally for models with global clocks, it is not the case for asynchronous models.

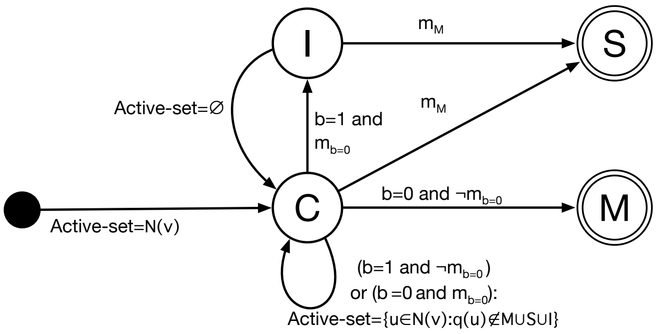
I
S
C
M
m_M
m_M
Active-set=∅
b=1 and $m_{b=0}$
b=0 and $\neg m_{b=0}$
Active-set=N(v)
(b=1 and $\neg m_{b=0}$) or (b =0 and $m_{b=0}$): Active-set={u∈N(v):q(u)∉M∪S∪I}

Figure 3.1: MIS in the PN model [48].

3.3 Leader election

Early LE algorithms were designed for parallel networks of processors. LE in anonymous cycles was studied by Attiya et al. in [12] for both synchronous and asynchronous models with runtime $\mathcal{O}(n \log n)$ and $\mathcal{O}(n^2)$ respectively. A variant of the asynchronous model with eventual termination was presented in [12]. This algorithm has the same runtime than the synchronous model, and it highlights the trade off between synchronization and eventual termination. Similarly, Itai and Rodeh presented in [38] two algorithms for LE in cycles:

1. LE with known n and terminal correctness. The algorithm works in phases numbered incrementally. At each phase active nodes choose a random candidate number from $\{1, \ldots, n\}$, and they submit their candidacy by sending a wave with the phase number and candidate number. In addition, the wave has a binary flag indicating whether the active node is known to be unique or not. Nodes with maximal candidate number advance to next phases. The procedure repeats until there is only one wave, which then reaches the originating candidate from the other side of the cycle. The algorithm has runtime $\mathcal{O}(n \log^2 n)$ w.h.p.

2. LE with unknown n and eventual correctness. The algorithm calculates an estimate of n and then proceeds as the first algorithm. Each node calculates a local estimate, sends it to one side of the ring and waits for it come back. This algorithm does not terminate, it eventually converges to a correct solution for LE. To succeed w.h.p. each node repeats the process r times, where $r = 1/\varepsilon$ is a model parameter. A binary flag is sent along with the wave to facilitate distinguishing waves from other nodes. Waves are removed in case the estimate is known to be invalid. Otherwise they are forwarded with an updated estimate. The estimate is incremented by one in case the node is certain that the estimation is both smaller than n and originated by another node.

The LE problem for arbitrary graphs was studied by Matias and Afek in [46] under three variants of anonymous asynchronous PN by building spanning trees. The first algorithm assumes that nodes know n (i.e. there is uniformity) and it has runtime $\mathcal{O}(D)$ w.p. $> 1 - \varepsilon$. Nodes are ranked based on a locally generated ID of length $\mathcal{O}(\log n)$. The ID is a pair of (a) the number of coin tosses the node makes until it gets heads, and (b) a number chosen randomly from $1, \ldots, d$, where $d = \mathcal{O}(r \ \log \ r)$

and $r = 1/\varepsilon$. Each node marks the port that links to its parent, the one from which it received the highest ID. Algorithm C.4 is an abstraction of the original algorithm. States are represented by the maximum received ID, and nodes move sequentially from one state to the next one until converging.

The other two algorithms require a lower bound of network size L. They differ in the termination detection condition. For the case without termination detection, nodes become candidates (transmit their ID) only in case that $\mathsf{ID} < L$. This variant reduces the length of IDs to $\mathcal{O}(\log r)$, thus reducing message complexity. However time complexity remains $\mathcal{O}(D)$ w.p. $> 1 - \varepsilon$. In addition to a known L, the case with termination detection requires a known upper bound of the network size $N = k \cdot L$ The same runtime applies for this case, the only extra step is to verify termination. For $k = 2$ termination can be confirmed as long as the tree size is greater than L since there can only be one such tree in networks with size $L < n \leq 2L$. For calculating the size of the tree, the root node sends a broadcast wave indicating that it is the root, and it waits for the acknowledge of all children. All three algorithms are adaptable to broadcast networks since the same message is sent to all neighbors. The exchange of large IDs prevents its applicability to models with limited memory and/or message size, such as nFSM and BE. Lastly, although these algorithms are adaptable for RN networks the runtime complexity would differ. Additional rounds are required to broadcast IDs, since algorithms in RN have to cope with collisions.

3.4 Vertex coloring

A randomized algorithm for $(\Delta + 1)$-coloring in the PN model was presented by Métivier et al. in [47]. The algorithm works in a fully synchronous model. Nodes are not required to know the actual network size n nor an upper bound N of the network size. The algorithm has two parts, the first one colors with an arbitrary number of colors and the second part reduces the number of colors to $\Delta + 1$. Algorithm C.3 enriches the original algorithm with state transitions. Figure 3.2 depicts the state transitions, with state set $\{A, W, C\}$ corresponding to active, waiting and colored. In the first part each node u builds a color iteratively, by generating a random bit b_u and concatenating it to the current color $c_u \leftarrow c_u \parallel b_u$. Nodes keep in memory a set of labels indicating the active set neighbors. Nodes exchange their bit at each step, and remove the nodes with different bit from the active set. This is repeated until all nodes have emptied their active set, yielding a valid coloring. At the same time, nodes build local sets of incoming and outgoing neighbors, by comparing the bit which differs from their neighbor b_u and b_v. At the end of part one, nodes have built an orientation of the graph. The second part consists of recoloring waves starting from local maxima nodes. Nodes choose a color u.a.r. from the $(\Delta + 1)$ palette, such that it does not conflict with incoming neighbors, and they acknowledge the chosen color. This technique was previously presented by Johansson [39]. The algorithm completes coloring in $\mathcal{O}(\log n)$ w.h.p. The algorithm does not apply to nFSM, since nodes need to keep large color values in memory for the first part. The synchrony requirement restricts its applicability to synchronous models. In particular, the algorithm requires distinguishing messages coming from different ports, which is not applicable to any broadcast MP model.

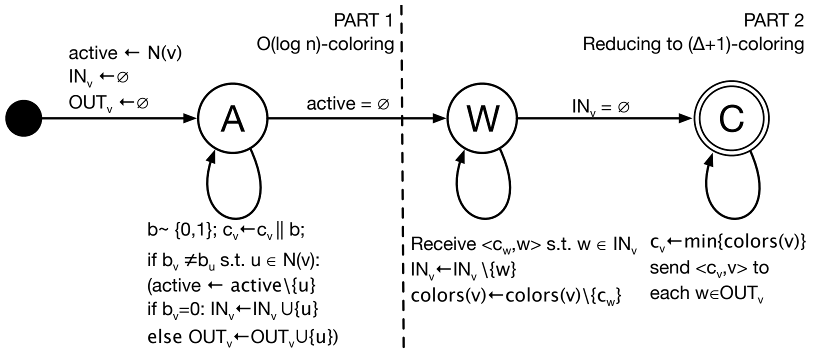

Figure 3.2: $(\Delta + 1)$-coloring in the PN [47].

Chapter 4

Radio networks

This chapter reviews the model capabilities of different RN models presented in previous literature in terms of specific problems. Section 4.1 reviews the key problem of deterministic and randomized broadcast (BC) under the radio network model (RN) with and without collision detection (CD). Sections 4.2, 4.3 and 4.4 cover algorithms for solving maximal independent set (MIS), leader election (LE) and vertex coloring in different RN and beeping (BE) models.

4.1 Broadcasting

This section reviews single source broadcasting algorithms in RN. Broadcasting consist of delivering a message from a single source to all other nodes (see Table 2.2 for a formal definition). BC is a key operation used as a building block for solving other problems. It has been utilized both as a subroutine and as a complexity metric T_{BC} (i.e. measuring complexity in terms of times to broadcast). Among other applications, BC is used as subroutine to solve LE [13]. The crucial challenge of BC in RN without CD is to find the optimal scheduling of transmissions that reduces collisions at the uncovered (i.e. uninformed) nodes. In this regard, collision detection and determinism play crucial roles in the computational complexity of BC. Tables 4.1 and 4.2 summarize results presented in previous literature for randomized and deterministic BC in RN respectively. The rest of this section reviews different BC algorithms in detail.

A “decay” approach was presented by Bar-Yehuda et al. in [13] for BC in the wake on message silent RN model. Nodes compete with neighboring nodes to become the only transmitter. Their chances of succeeding

Model	**Know-ledge**	**Graph family**	**Runtime**	**Ref.**
Std. RN	Topology	$\mathcal{G}_U$	$\Omega(\log^2 n)$	[3]
Std. RN	N, D	$\mathcal{G}$	$\mathcal{O}(D + \log^6 n)$	[33]
Silent RN	None	$\mathcal{G}_U$	$\Omega(D \log n/D)$	[43]
Silent RN	None	$\mathcal{G}_D$	$\Omega(D \log n/D)$	[24]
Silent RN	$D \leq N^{1-\varepsilon}$	$\mathcal{G}$	$\Omega(D \log n)$	[43]
Silent RN	n and D	$\mathcal{G}_U$	$\mathcal{O}(D \log n + \log^2 n)$	[13]
Silent RN	N	$\mathcal{G}_U$	$\mathcal{O}(D \log N/D + \log^2 N)$	[40]
Silent RN	n	$\mathcal{G}_D$	$\mathcal{O}(D \log n/D + \log^2 n)$	[27]
Silent RN	None	$\mathcal{G}_D$	$\mathcal{O}(D \log n/D + \log^2 n)$	[24]
Silent RN	IDs	$\mathcal{BG}$	$\Omega(D \log n/D)$	[54]

Table 4.1: Complexity of randomized BC in RN.

Model	**Know-ledge**	**Graph family**	**Runtime**	**Ref.**
Std. RN	IDs	$\mathcal{BG}$	$\mathcal{O}(D \log n)$, $\Omega(D + \log n)$	[54]
Std. RN	IDs	$\mathcal{G}_D$	$\mathcal{O}(n \cdot D)$	[21]
Silent RN	IDs	$\mathcal{G}_D$	$\Omega(D \log n)$	[21]
Silent RN	IDs	$\mathcal{G}_D$	$\Omega(n \log D)$	[24]
Silent RN	IDs	$\mathcal{G}_U$	$\Omega(n)$	[13, 42]
Silent RN	IDs	$\mathcal{G}_U$	$\Omega(D \log n)$	[17]
Adv. RN	IDs	$\mathcal{G}_U$	$\Omega(n \log n)$	[42]
Silent RN	IDs, N	$\mathcal{G}_U$	$\mathcal{O}(n \log n)$, $\Omega(n \frac{\log n}{\log(n/D)})$	[40]
Silent RN	IDs	$\mathcal{BG}$	$\mathcal{O}(n \log n)$, $\Omega(n \log_{n/D} n)$	[54]
Silent RN	IDs	$\mathcal{G}_D$	$\mathcal{O}(n^{11/6})$	[21]
Silent RN	IDs	$\mathcal{G}_D$	$\mathcal{O}(n \log^2 n)$	[23]
Silent RN	IDs, n, D	$\mathcal{G}_D$	$\mathcal{O}(D\Delta \log(n/\Delta))$	[24]
Silent RN	IDs, n	$\mathcal{G}_D$	$\mathcal{O}(D\Delta \log n \log(n/\Delta))$	[24]
Silent RN	IDs	$\mathcal{G}_D$	$\mathcal{O}(D\Delta \log(n/\Delta) \log^{1+\alpha} n)$	[24]

Table 4.2: Complexity of deterministic BC in RN.

decrease as the rounds of a phase progress. The decay BC algorithm requires nodes to know the network diameter D and a size estimate N. Algorithm C.5 extends the original algorithm with state transitions. Figure 4.1 depicts the state transitions, with state set $\{W, A, C, P\}$ corresponding to waiting, active, competing and passive. Nodes start and remain in waiting state W until they receive the message then become active. The active state has the sole purpose of synchronization. The algorithm proceeds in phases. Nodes use the value of D to calculate phase limits, in which they move into competing state at the same time. The value of N defines the number of phases [13]. The algorithm accomplishes broadcast in time $\mathcal{O}(D \log n + \log^2 n)$. Knowledge of N is a necessary condition for the algorithm to terminate [24], otherwise nodes would have to run forever in case new nodes are activated. The algorithm is not directly applicable to the harsh RN model due to the activation by incoming message assumption and the requirement of known graph properties. It is not applicable to the nFSM model either, due to the use of large counters. However, the decay approach has been largely influential for developing extensions and adaptations that are applicable to different RN models [27, 32, 40].

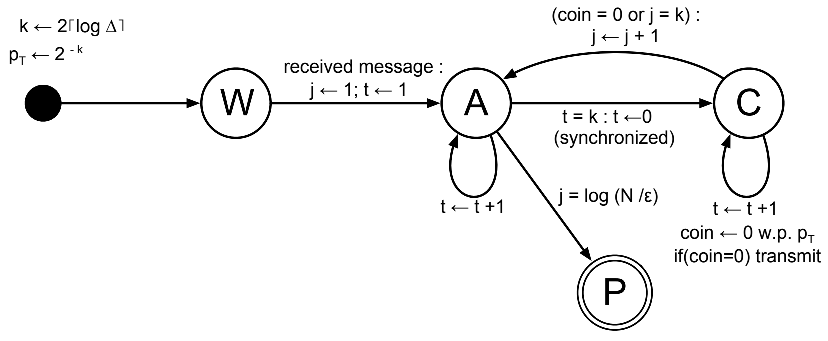

Figure 4.1: BC in silent RN – decay approach [13].

An improvement to decay BC in silent RN was presented and named “optimal” BC by Kowalski and Pelc [40]. It is based on the observation that uncovered nodes with a large number of covered neighbors cannot be informed fast w.h.p. The optimized BC proceeds in phases of $\log N/D + 2$ steps, where N is a known estimate of the network size and D the network diameter. During the first $\log N/D + 1$ nodes transmit w.p. 2^q and in the last step w.p. p_i. The value $q = \log N/D$ is chosen such that nodes with at most N/D informed neighbors receive the message w.h.p. The value of p_i is chosen based on the *universal probability* *sequences* defined in [40], such that nodes with more than N/D informed neighbors get informed in the last step. Nodes are not required to know D, instead they use a "doubling technique" to estimate D in exponential increments. This estimate is used to decide whether to execute optimal BC for $D > 32N^{2/3}$, or the basic decay BC otherwise. The algorithm is not applicable to nFSM due to the counters needed for iterating over the steps, however it is applicable to BE. Algorithm C.7 is an abstraction of the original algorithm. Figure 4.2 depicts the state transitions, with state set $\{W, A_i, P_i\}$ corresponding to active, competing and passive, and $i \in \{d, o\}$ to decay and optimal BC subroutines.

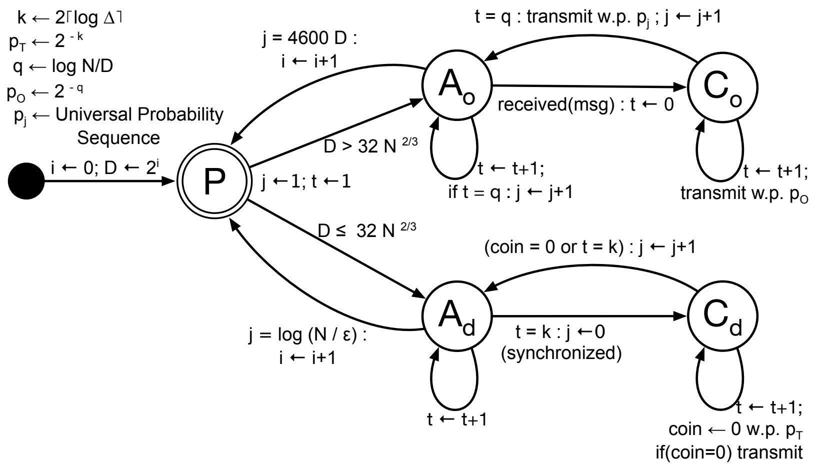
k ← 2⌈log Δ⌉
p_T ← 2^-k
q ← log N/D
p_O ← 2^-q
p_j ← Universal Probability Sequence
i ← 0; D ← 2^i
P
j ←1; t ←1
j = 4600 D : i ← i+1
D > 32 N^2/3
D ≤ 32 N^2/3
j = log (N / ε) : i ← i+1
A_o
C_o
t = q : transmit w.p. p_j ; j ← j+1
received(msg) : t ← 0
t ← t+1; if t = q : j ← j+1
t ← t+1; transmit w.p. p_O
A_d
C_d
(coin = 0 or t = k) : j ← j+1
t = k : j ←0 (synchronized)
t ← t+1
t ← t+1; coin ← 0 w.p. p_T if(coin=0) transmit

Figure 4.2: Randomized broadcasting in RN without CD [40].

Improved BC algorithms for silent RN were presented by Czumaj and Rytter in [27]. They presented the original decay algorithm in terms of "selecting sequences" for the transmission probabilities. This clarifies the actual probability with which nodes transmit at each round during the decay process. At a given round $i \in \{1, \ldots, \log n\}$ of each phase, nodes transmit with probability $2^{i-\log n}$. The improved algorithm replaces the original sequence with a parameterized sequence based on $\lambda = \log n/D$. Algorithm C.6 is a transcription of Czumaj and Rytter's linear randomized BC algorithm. Although the algorithm assumes that nodes know the network size n, the authors make reference to an estimation technique for removing this assumption.

Deterministic BC algorithms were presented by Chlebus et al. in [21] for both the standard RN and the silent RN. The algorithm requires unique IDs, since deterministic symmetry breaking is impossible in anonymous networks [4]. For the case with CD, Chlebus et al. presented an encoded BC algorithm with runtime $\mathcal{O}(D|\sigma|)$ for transmitting a message with size σ. Informally, the algorithm encodes the message into two consecutive bit streams through carrier sensing. A silent round represents $\sigma_i = 0$ and a non-silent round (i.e. with one message received or a collision detected) represents $\sigma_i = 1$. In addition, it was proven that with CD it is possible to accomplish acknowledged BC, whereas for the case without CD it was proven to be impossible. Acknowledged BC means that the source is required to know that BC has completed successfully. Furthermore, Chlebus et al. in [21] presented algorithms for acknowledged BC in (a) symmetric graphs with runtime $\mathcal{O}(n)$, and (b) strongly connected graphs with runtime $\mathcal{O}(n|\sigma|)$. For the case without CD, Chlebus et al. presented an algorithm with runtime $\mathcal{O}(n^{11/6})$. Informally, waves of informing nodes progress one hop per phase. Nodes calculate the transmission schedules in advance by computing a (n, k)-*selective* family $\mathcal{F}$. An family $\mathcal{F}$ of subsets of $[n]$ is (n, k)-*selective* if for every non empty subset Z of $[n]$, s.t. $|Z| \leqslant k$, there is a set $F \in \mathcal{F}$ with $|Z \cap F| = 1$ [24]. An common arbitrary ordering is established. All nodes have the same subsets and ordering. The algorithms proceed in j phases, each with $k = |\mathcal{F}|$ rounds. Nodes transmit only in rounds for which their label belongs to the j-th subset.

The previous approach was named "dovetail" and further developed by Clementti in [24]. The study presented three variations of Chlebus algorithm, for different assumptions of known graph properties. Algorithm A assumes known n and Δ. It proceeds in phases $i = 1, 2, \ldots$ with an ordered set of families $\mathcal{F}_i$. Nodes start uninformed in the inactive state. Nodes may only become active at round j of a given phase i, provided that: (a) its label belongs to the jth set of $\mathcal{F}_i$ and (b) they have been informed for the first time during the previous phase $i - 1$. This algorithm was proven to complete broadcast in $\mathcal{O}(D\Delta \log(n/\Delta))$. For the case of known n only, algorithm B proceeds in phases $h = 1, 2, \ldots$ of $\lceil \log n \rceil$ rounds. At round ℓ of phase h each node runs round h of algorithm A with $\Delta = 2^{\ell}$, s.t. $1 \leqslant \ell \leqslant \lceil \log n \rceil$. In addition, nodes that become inactive while executing algorithm A will move to inactive state for the remaining of algorithm B. The runtime of algorithm B is $\mathcal{O}(D\Delta \log n \log(n/\Delta))$. Lastly, for the case of no global properties known, algorithm C executes algorithm B with $n = 2^{\ell}$, for $\ell = 1, 2, \ldots$ serving as estimate. This algorithm has runtime $\mathcal{O}(D\Delta \log(n/\Delta) \log^{1+\alpha} n)$.

4.2 Maximal independent set

The MIS problem has been extensively studied in a large variety of models for different graph families. Results and lower bounds vary depending on the graph, model features and assumptions. This section reviews results for MIS in RN and BE models. The complexity of these algorithms is summarized in Table 4.3.

Model	**Know-ledge**	**Graph family**	**Runtime**	**Ref.**
Harsh RN	n	$\mathcal{UD}$	$\mathcal{O}(\log^2 n)$ w.h.p.	[49]
nFSM	None	$\mathcal{G}$	$\mathcal{O}(\log^2 n)$ w.h.p.	[30]
BE	Known N	$\mathcal{G}$	$\mathcal{O}(\log^3 n)$ w.h.p.	[1]
Wake on beep BE	None	$\mathcal{G}$	$\mathcal{O}(\log^3 n)$ w.h.p.	[1]
Wake on beep BE with CD	None	$\mathcal{G}$	$\mathcal{O}(\log^2 n)$ w.h.p.	[1]
Synchronous clocks BE	None	$\mathcal{G}$	$\mathcal{O}(\log^2 n)$ w.h.p.	[1]

Table 4.3: Complexity of randomized MIS in RN.

Model	**Know-ledge**	**Graph family**	**Runtime**	**Ref.**
Std. RN	IDs, n	$\mathcal{BG}$	$\mathcal{O}(\log n)$	[54]
Std. RN non-simultaneous	IDs, n	$\mathcal{BG}$	$\mathcal{O}(\log n)$, $\Omega(\log n)$	[54]

Table 4.4: Complexity of deterministic MIS in RN.

An algorithm to solve MIS in $\mathcal{UD}$ graphs in the harsh RN model was presented by Moscibroda and Wattenhofer in [49]. Algorithm C.11 is an abstraction of the original algorithm. Figure 4.3 depicts the state transitions, with state set $\{W, A, C, M, S\}$ corresponding to waiting, active, competing, MIS and slave (i.e. the node decided not join the MIS). The algorithm assumes that nodes have the ability to store and exchange large counters (large memory and message size) and know the network size. Different counter thresholds are used as node “filters” before they

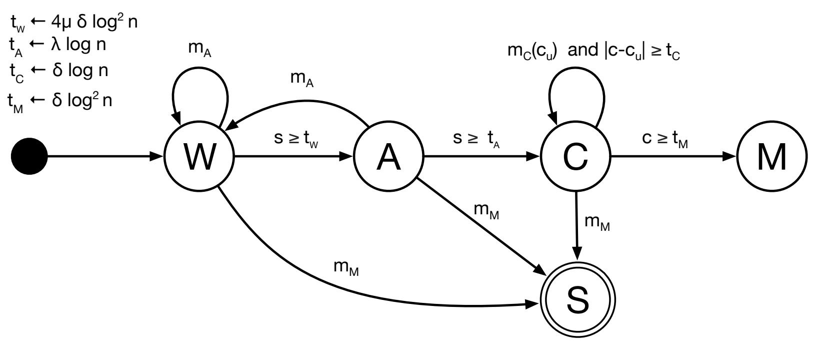
$t_W \leftarrow 4\mu\ \delta \log^2 n$
$t_A \leftarrow \lambda \log n$
$t_C \leftarrow \delta \log n$
$t_M \leftarrow \delta \log^2 n$
m_A
m_A
$m_C(c_u)$ and $|c-c_u| \geq t_C$
W
$s \geq t_W$
A
$s \geq t_A$
C
$c \geq t_M$
M
m_M
m_M
m_M
S

Figure 4.3: MIS in the harsh RN model [49].

progress to the next state, including t_W, t_A and t_M. The threshold t_W has the purpose of avoiding interference with competing nodes from newly woken nodes. Thresholds t_A and t_M are used to break node symmetry, as opposed to random priority values used by Luby [45] and Métivier et al. [48]. The threshold t_C is used as safety condition before joining the MIS. Nodes calculate the difference between its counter c and each neighbor counter c_u, such that $|c - c_u| \geq t_C$ holds in order to join the MIS. An important aspect to highlight is that nodes may only terminate in 'slave' state, and nodes that joined the MIS run forever claiming their decision. This algorithm is not applicable to the nFSM model due to the use of large counters. The complexity is different in the BE model, which requires transmitting messages as bit streams.

A randomized algorithm for solving MIS in the nFSM model was presented by Emek and Wattenhofer in [30]. This is a RN model with asynchronous communication and adversarial scheduling. Additionally, nodes have a small constant memory size, making them unable to hold large counters. Therefore a different approach is required for synchronizing nodes, keeping a fairness condition and producing valid outputs. The algorithm leverages a synchronizer protocol [30] for soft alignment. This is a partial alignment in which the local time of two neighboring nodes differs by at most one round. As a trade off, the model requires a larger state size equal to the square of the original size. In this partly aligned model, rounds are referred to as *turns*. In addition, a pulse variable $t \in \{0, 1, 2\}$ is used to cope with asynchronous activation. Algorithm C.10 is an abstraction of the original algorithm. Figure 4.4 depicts the state transitions, with state set $\{W, A, C_t, M, S\}$ corresponding to waiting, active, competing, MIS and slave, and t corresponding to the pulse variable. At each turn, nodes broadcast their state information, and they receive a multiset of states from their neighbors. Nodes have access to a bounded counter function $\beta(\sharp(q))$, that obtains the occurrences of state q in the multiset received. Nodes start in active state and they move to competing state in case no neighbor is in active state. Depending on a randomly generated bit b and the multiset of received states nodes either join the MIS or move to waiting state. Lastly, nodes in waiting state move to the slave state in case a neighbor has joined the MIS, otherwise they move back to active state. For each node the number of competing rounds is modeled as an independent variable, which follows a geometric probability distribution $\mathrm{G}(p)$ with parameter $p = 1/2$. The sum of all these n variables is at most $\mathcal{O}(\log n)$ w.h.p. Active nodes decrease by $\mathcal{O}(\log n) + NB(\mathcal{O}(\log n), 1 - p)$ w.h.p. Combining these results, the algorithm has runtime $\mathcal{O}(\log^2 n)$ w.h.p. Since the nFSM model assumes guaranteed delivery, this algorithm is not directly applicable to RN models neither with CD nor without CD. Lastly, an adaptation is required for applying it to the BE model, in order to efficiently translate large messages (synchronized version of state messages) into bit streams.

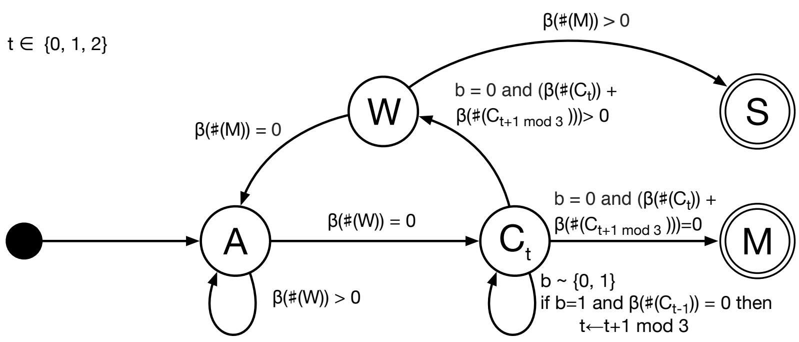
t ∈ {0, 1, 2}
β(#(M)) > 0
W
S
b = 0 and (β(#(C_t)) + β(#(C_{t+1 mod 3})))> 0
β(#(M)) = 0
A
β(#(W)) = 0
C_t
b = 0 and (β(#(C_t)) + β(#(C_{t+1 mod 3})))=0
M
β(#(W)) > 0
b ~ {0, 1}
if b=1 and β(#(C_{t-1})) = 0 then t←t+1 mod 3

Figure 4.4: MIS in the nFSM model [30].

A template for solving MIS in BE was presented in [1], along with multiple instantiations for variations of the BE model. These variations depend on the uniformity and sender side CD. The template requires an estimate of the network size, and makes use of counters for iterating over phases and rounds. A default instantiation of the template considers the standard BE model with a known upper bound of the network size N, and other variations utilize an estimate obtained by the algorithm. Algorithm C.13 is a modification of the original version, to highlight state transitions. Figure 4.5 depicts the state transitions, with state set $\{W, C, M\}$ corresponding to waiting, competing and MIS.

Nodes remain in the waiting state for $t_W = c\log^2 n$ rounds. At each round some neighbor beeps w.p. at most $1/2$. Any node that hears a beep restarts its counter and moves back to waiting state. It follows that a node beeps alone every $t_C \cdot t_M = \mathcal{O}(\log^2 N)$ rounds w.p. $1/e$, and $\mathcal{O}(\log n)$ of these events are required to build the MIS. Combining these results, the algorithm runtime is $\mathcal{O}(\log^3 n)$ w.h.p. The algorithm does not terminate but eventually converges. MIS nodes keep beeping w.p. $1/2$ to claim their place. Since these algorithms use large counters, they are not applicable to the nFSM model. Lastly, it does not apply to models without CD.

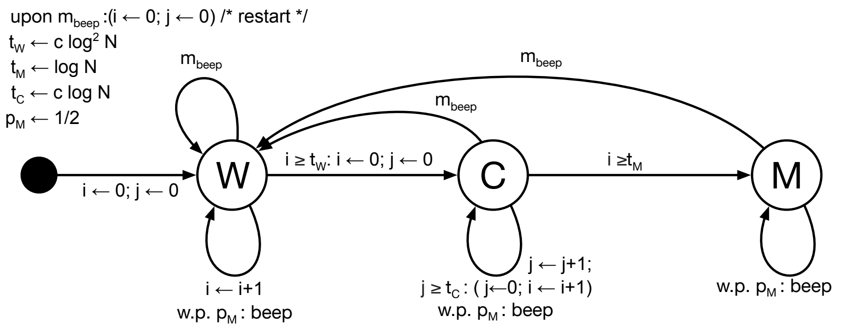

Figure 4.5: MIS in BE with known N [1].

Two instantiations of the template for solving MIS were studied by Afek et al. in [1] for *wake on beep* variations of BE with and without sender side CD. Wake on beep refers to assuming that nodes are woken up by incoming beeps. Algorithms C.14 and C.15 are abstractions of the original algorithms. Figures 4.6 and 4.7 depict the state transitions, with states $\{W, A, C, M, I, S\}$ corresponding to waiting, active, competing, MIS, inactive and slave. Since both algorithm assume wake on beep, they have these commonalities: (a) Replace a known upper bound of the network size N with an estimate $N = 2^x$, where x is the current phase counter. (b) Create a partial alignment by defining a preliminary waiting state W, in which nodes wake up, beep and wait for one round. (c) Perform k exchanges or "challenges" with a number of rounds each. Since nodes are only partially aligned, at least two rounds are required for moving into MIS state in case neighbors are one round behind. Informally the difference between the two is that the variant with sender side CD is restrictive, while the variant without sender side CD is permissive. With sender-side CD, nodes make sure no neighbor has beeped recently before they join the MIS. Nodes move from active to competing only with a probability p_C inversely proportional to the network size estimate. Without sender-side CD nodes that joined the MIS may later leave the MIS in case of conflicts. A silent round is enough for nodes to advance state to competing and to MIS. This difference is reflected in the termination condition. In the model without CD nodes can only converge to the correct output, whereas in the model with CD nodes terminate. Lastly, the algorithm with CD requires $k = 4$ challenges and has runtime $\mathcal{O}(\log^2 n)$ w.h.p. The algorithm without CD requires only $k = 3$ challenges, however it has runtime $\mathcal{O}(\log^3 n)$ w.h.p.

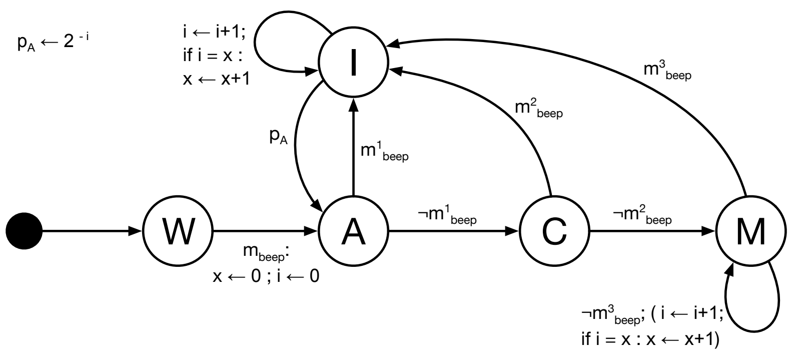
$p_A \leftarrow 2^{-i}$
$i \leftarrow i+1$; if $i = x$: $x \leftarrow x+1$
I
p_A
m^1_{beep}
m^2_{beep}
m^3_{beep}
W
A
C
M
m_{beep}: $x \leftarrow 0$; $i \leftarrow 0$
$\neg m^1_{beep}$
$\neg m^2_{beep}$
$\neg m^3_{beep}$; ($i \leftarrow i+1$; if $i = x$: $x \leftarrow x+1$)

Figure 4.6: MIS in BE with wake on beep, without sender side CD [1].

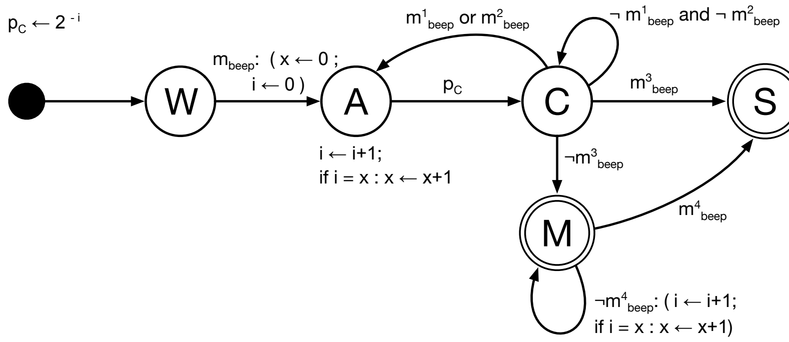
$p_C \leftarrow 2^{-i}$
m^1_{beep} or m^2_{beep}
$\neg m^1_{beep}$ and $\neg m^2_{beep}$
m_{beep}: ($x \leftarrow 0$; $i \leftarrow 0$)
W
A
p_C
C
m^3_{beep}
S
$i \leftarrow i+1$; if $i = x$: $x \leftarrow x+1$
$\neg m^3_{beep}$
M
m^4_{beep}
$\neg m^4_{beep}$: ($i \leftarrow i+1$; if $i = x$: $x \leftarrow x+1$)

Figure 4.7: MIS in BE with wake on beep and sender side CD [1].

Another instantiation of the template for solving MIS was presented by Afek et al. in [1] for the synchronized clocks BE. This algorithm implements the MIS algorithm template based on counters. The algorithm is an extension of Métivier et al. [48] algorithm, which is based Luby's permutation algorithm [45]. In this algorithm, state exchange is accomplished by designating two time slots, corresponding to competing and MIS states. This model assumes synchronized local clocks throughout the execution of the algorithm. Similarly to the wake on beep MIS algorithms, it replaces the need for a known upper bound of the network size with an estimate k of the priority sizes (i.e. number of bits). Although the communication is synchronous, node activation is only decided by an adversary (i.e. nodes are not waken up by beeps). For coping with adversary activation, this algorithm designates every third time slot for a restart bit. The restart bit is used to ensure that nodes have the same estimate k throughout the algorithm. Algorithm C.16 is an abstraction of the original algorithm. Figure 4.8 depicts the state transitions, with state set $\{I, C, M\}$ corresponding to inactive, competing and MIS. It was proven that for each node its $\mathcal{O}(\log n)$ neighborhood is stable during $\Omega(\log^2 n)$ rounds. The algorithm runtime is $\mathcal{O}(\log^2 n)$ w.h.p., since it performs at most $\mathcal{O}(\log n)$ executions of Luby's algorithm, which has runtime $\mathcal{O}(\log n)$. This algorithm is not applicable to the harsh RN model nor to the nFSM model. Firstly, because it assumes synchronous clocks and secondly, because it relies on large counters.

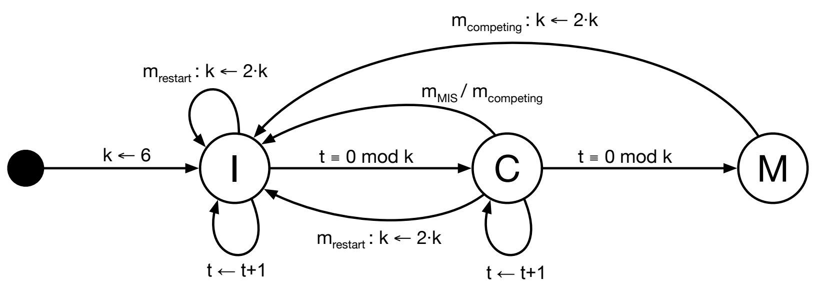

Figure 4.8: MIS in BE with synchronous clocks [1].

Lastly, a deterministic algorithm for solving MIS in the synchronous BE was presented by Schneider and Wattenhofer in [54]. It makes two assumptions: (a) nodes have unique IDs, and (b) networks have underlying $\mathcal{BI}$ graphs, such that a polynomial function $f(r)$ defines the size of the maximum independent set in the r radius. Algorithm C.12 is an abstraction of the original algorithm. Figure 4.9 depicts the state transitions, with state set $\{A, C, I, M, S\}$ corresponding to active, candidate, inactive, MIS and slave. The algorithm iterates for $f(f(2)+2)$ stages of $f(2)$ phases of $\log^* n + 2$ competitions. Nodes start in active state. Each competition consists of two parts. In the first part nodes transmit their IDs as bit streams. Nodes detect that another node has higher ID in case a beep is heard during a silent round. The second part consists of an update state subroutine with two rounds. In case an active node does not have a neighbor with a higher ID during the first phase, it joins the MIS, it listens for one round and it beeps in the next one. In case an active node does not have a neighbor with a higher ID during a later phase, it moves to candidate state, it beeps for one round and listens in the next one. Otherwise, they listen for two rounds, and they move to inactive and to slave states in case they hear beep in the first and second rounds respectively. The algorithm runtime is $\mathcal{O}(\log n)$, since it takes $f(f(2)+2)$ stages of $\log n$ rounds. The algorithm was extended for the asynchronous wake on beep BE, without runtime overhead. It defines a pulse beat, such that every six rounds it executes a step of the synchronous algorithm. Upon waking up, nodes wait for seven silent rounds before starting execution. In case they hear two consecutive beeps they move to slave state, since that means a neighbor has joined the MIS. MIS nodes beep in the last two rounds (i.e. fifth and sixth). Other nodes only beep in the first round. The algorithm does not terminate, it rather converges to a correct output.

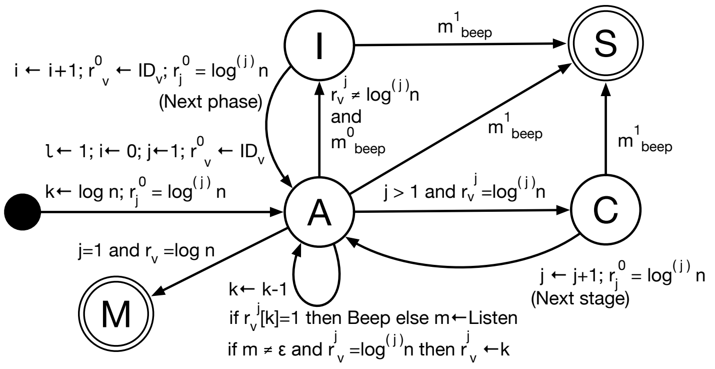
I
S
A
C
M
m^1_{beep}
$i \leftarrow i+1$; $r^0_v \leftarrow ID_v$; $r^0_j = \log^{(i)} n$
(Next phase)
$r^j_v \neq \log^{(i)} n$
and
m^0_{beep}
m^1_{beep}
m^1_{beep}
$l \leftarrow 1$; $i \leftarrow 0$; $j \leftarrow 1$; $r^0_v \leftarrow ID_v$
$k \leftarrow \log n$; $r^0_j = \log^{(i)} n$
$j > 1$ and $r^j_v = \log^{(i)} n$
$j=1$ and $r_v = \log n$
$j \leftarrow j+1$; $r^0_j = \log^{(i)} n$
(Next stage)
$k \leftarrow k-1$
if $r^j_v[k]=1$ then Beep else $m \leftarrow$ Listen
if $m \neq \varepsilon$ and $r^j_v = \log^{(i)} n$ then $r^j_v \leftarrow k$

Figure 4.9: Deterministic MIS in fully synchronous BE [54].

4.3 Leader election

LE is a highly relevant problem in distributed computing, since it can be leveraged for solving other problems. Similarly to BC, LE has been extensively studied under various RN models. Decisive model features include determinism, collision detection and synchrony. A recent review of LE algorithms for different models was presented by Ghaffari and Haeupler in [32]. In this study, LE is proven to be as hard as BC. For asynchronous RN, both with and without CD, it was shown that LE lower bounds are precisely the lower bounds for BC in the corresponding setting. The BC lower bounds in Tables 4.2 and 4.1 are closely related to the LE lower bounds in Tables 4.5 and 4.6.

Model	**Know-ledge**	**Graph family**	**Runtime**	**Ref.**
Silent RN	n, D	$\mathcal{G}_U$	$\mathcal{O}(D\log n + \log^2 n) \cdot \log n$ w.h.p.	[14]
Silent RN	n, D	$\mathcal{G}_U$, $\mathcal{G}_D$	$\mathcal{O}(D\log n/D + \log^2 n) \cdot \sqrt{\log n}$ w.h.p.	[26]
Std. RN or BE	n,D	$\mathcal{G}_D$	$\mathcal{O}(D\log n)$ (expected time)	[26]
Std. RN or BE	N, D	$\mathcal{G}_U$	$\mathcal{O}(D + \log n) \cdot \mathcal{O}(\log^2 n \log n)$ w.h.p.	[32]
Silent RN	N, D	$\mathcal{G}_U$	$\mathcal{O}(D\log n/D + \log^3 n) \cdot \min\{\log\log n, \log n/D\}$ w.h.p.	[32]
Silent RN	N	$\mathcal{G}_U$	$\mathcal{O}(n)$ w.h.p., $\Omega(n)$	[22]

Table 4.5: Complexity of LE in randomized RN.

Model	**Know-ledge**	**Graph family**	**Runtime**	**Ref.**
BE and wake on beep BE	IDs	$\mathcal{G}_U$	$\mathcal{O}(D\log n)$	[31]
Standard RN	IDs, N	$\mathcal{G}_U$	$\Theta(n)$	[41]
Silent RN	IDs, N	$\mathcal{G}_U$	$\Omega(n\log n)$	[41]
Silent RN	IDs, N	$\mathcal{G}_U$	$\mathcal{O}(n\log^{3/2} n\sqrt{\log\log n})$	[22]

Table 4.6: Complexity of LE in deterministic RN.

Multiple approaches to LE in RN have been presented in previous literature. A “simulation” approach to solve LE in the silent RN was introduced by Bar-Yehuda et al. in [14]. The authors provided a simulation of single-hop RN with CD from multi-hop RN without CD. The simulation is applicable to anonymous RN. However it only works for synchronous RN and not for asynchronous RN. It simulates a round of a single-hop RN in T_{BC}. It makes use of decay and broadcast protocols, thus requiring nodes to know the network size n and diameter D.

The actual algorithm for LE was not presented but rather conjectured in [14]. The claim was based on a previous result of $\mathcal{O}(\log\log n)$ for single-hop LE with CD and a multiplying factor of T_{BC}. Two different approaches to LE in RN were presented recently, namely the validating and clustering approaches. The rest of this section reviews these two approaches and Table 4.5 summarizes the complexity results.

4.3.1 Validating approach for LE

The "validating" approach to solve LE in RN was presented by Czumaj and Davies [26]. It assumes that nodes know the network size n and diameter D. Informally the approach consists of iteratively attempting to select a single leader and validating the selection. Initially a set of candidates S is selected at random. Each candidate generates an ID with length logarithmic in n. Candidates use optimized BC to transmit their IDs, and they remove their candidacy in case they detect a higher ID. Other nodes remain in active state forwarding candidate IDs. A validation condition is defined such that certain "witness" nodes detect whether there are zero, one or more than one candidates. These nodes play the special role of informing the rest of the network, such that other nodes either terminate or start a new iteration (i.e. a new attempt is made). Two algorithms for solving LE using this approach were presented in [26], corresponding to the models of silent RN and wake on beep BE. The remaining of this subsection reviews these two algorithms.

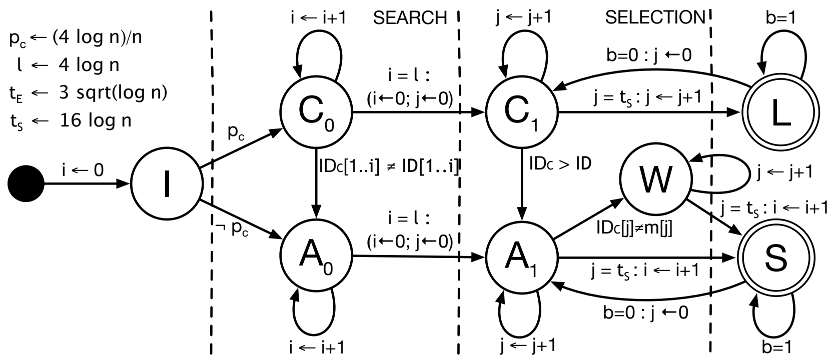

Figure 4.10: Optimal LE for directed graphs in RN [26].

1. Validating algorithm for LE in the silent RN. Nodes move to the 'candidate' state w.p. $(4\log n)/n$. A multi-BC subroutine is used to transmit multiple candidate IDs that are $\ell = \log n$ bits long, such

that recipients interpret the first received message. It iterates over two phases: a *search* phase for finding the highest ID and a *selection* phase to validate the existence of a single leader. The search phase performs multi-BC of IDs bit by bit; only the bits of the highest ID are forwarded. Candidates that receive a higher ID remove their candidacy. The selection phase includes two parts: Firstly, the remaining candidate nodes disseminate their ID using multi-BC, such that non-candidate nodes keep the first ID_c received. Secondly, nodes validate that a single candidate remains. The validation consist of $i \in \{1, \dots, 16 \log n\}$ rounds of decay, such that at the i-th round nodes with $ID_c[i] = 1$ perform decay. It was proven that four iterations of decay are enough for at least one witness node to detect multiple IDs (i.e. a witness with $ID_c[i] = 0$ receives a bit 1 in decay rounds). Thereafter witness nodes perform multi-BC to acknowledge the validation result to rest of the network. Both the search and the selection phase have runtime $\mathcal{O}(T_{BC} \cdot \sqrt{\log n})$ each. The overall runtime of this algorithm is $\mathcal{O}(T_{BC} \cdot \sqrt{\log n})$ w.h.p. Algorithm C.17 is an annotated version of the algorithm. Figure 4.10 depicts the state transitions, with state set $\{I, A_q, C_q, W, L, S\}$ corresponding to inactive, active, candidate, witness, leader and slave and $q \in \{0, 1\}$ to the search and selection phases, respectively.

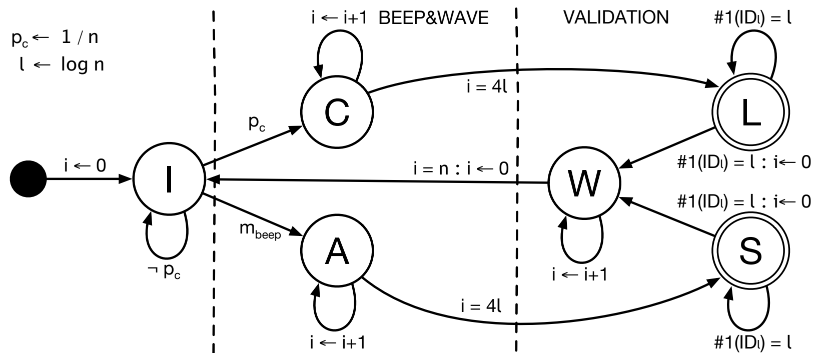

Figure 4.11: Optimal LE for directed graphs in BE [26].

2. Validating algorithm for LE in the wake on beep BE. Initially, nodes become candidate w.p. $1/n$. The algorithm includes a *beep-and-wave* subroutine, in which candidate nodes transmit their IDs as a beep sequence and other nodes repeat the received sequence, which possibly includes multiple superimposed IDs. These transmissions are performed every 3rd round. Candidates choose an

ID with $4 \log n$ bits, including $\log n$ 1's precisely. Candidate IDs are broadcasted using beep-and-wave. Witness nodes are able to detect multiple candidate IDs, in case they hear more than $\log n$ 1's. Thereafter witness nodes acknowledge the rest of the network by sending a validation flag using beep-and-wave. Both the beep-and-wave subroutine and the entire algorithm have an expected runtime of $\mathcal{O}(D + \log n)$. Algorithm C.18 is an annotated version of the algorithm. Figure 4.11 depicts the state transitions, with state set $\{I, A, C, W, L, S\}$ corresponding to inactive, active, candidate, witness, leader and slave.

Figure 4.12 shows an example of the validating approach requiring two iterations of the selection subroutine to find a single leader. The label inside each node represents its current state. The initial candidates are numbered sequentially as they appear in first part of the diagram. In addition, the diagram includes ID labels associated to each node, representing a candidate ID stored in memory. Each label $ID(C_i)$ denotes the ID of the i-th candidate node. Labels ID_c denote the first candidate ID received by a non-candidate node. It depicts five phases: (a) *Phase 0* shows the initial candidates with chosen IDs of length $\ell = \lceil \log_2(18) \rceil = 4$ and the result of the search subroutine as first $\sqrt{\log n}$ digits underlined, (b) *Phase 1* shows the first multi-BC with all nodes receiving some candidate ID, (c) *Phase 2* shows the result of the first witness detection, (d) *Phase 3* shows the second multi-BC (new candidate IDs) and the result of the second witness detection, and (e) *Phase 4* shows the final configuration after all nodes have the received the only remaining candidate ID and no more witness are detected.

4.3.2 Clustering approach for LE

A different "clustering" approach to LE in multi-hop RN was presented by Ghaffari and Haeupler in [32] as an algorithm template. Algorithm C.19 is modification of the original template to highlight state transitions. Figure 4.13 depicts the state transitions, with state set $\{W, A_q, C_q, U, B, L, S\}$ corresponding to waiting, active, candidate, unclustered, boundary, leader and slave, and $q \in \{0, 1\}$ to the clustering and elimination phases. The algorithm makes the following assumptions: (a) nodes know the network size n, (b) the message size is large enough $\mathcal{O}(\log n)$ to hold candidate IDs, (c) nodes are activated by incoming messages, (d) communication is synchronous through the establishment of global clocks in T_{BC} with a wake up wave, and (e) nodes

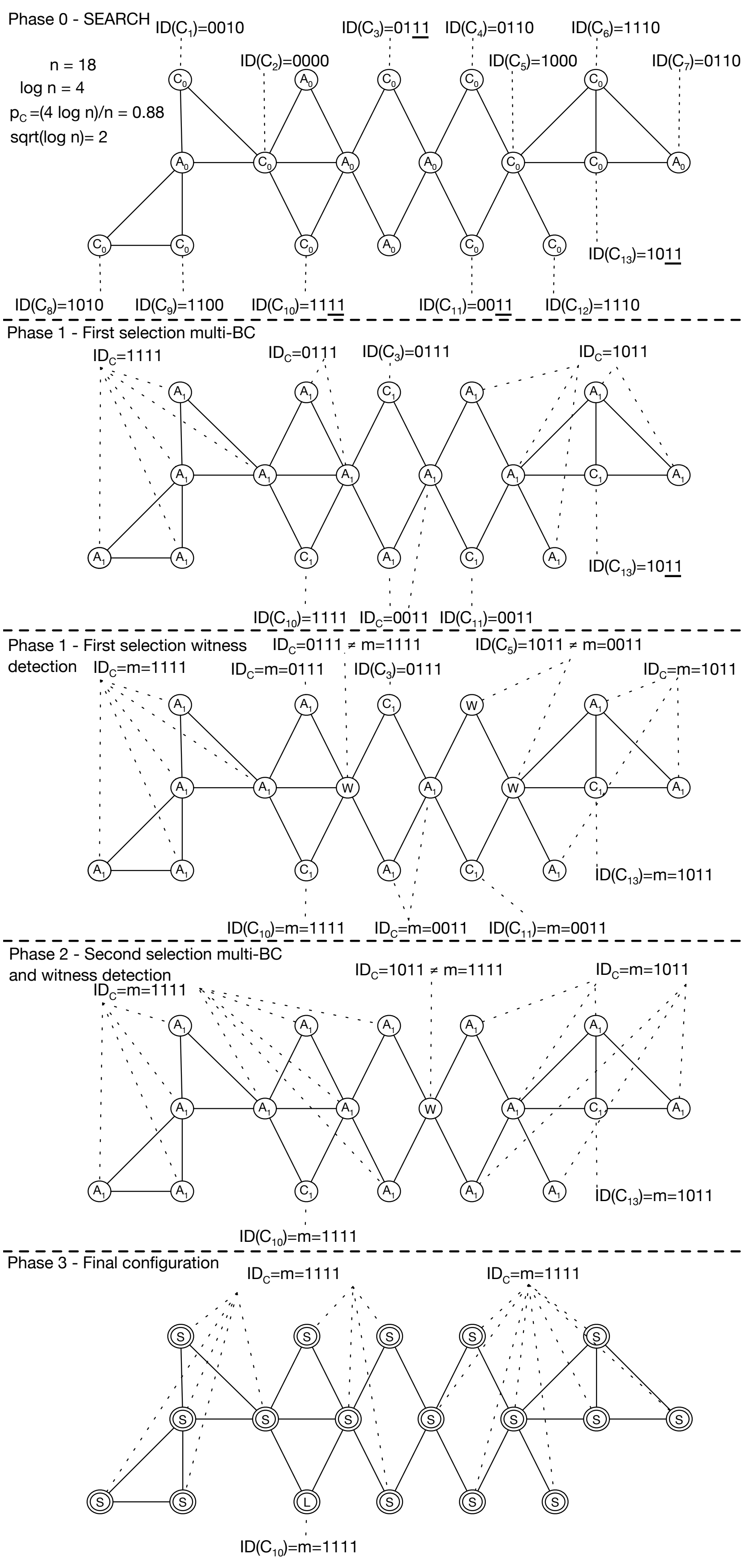

Figure 4.12: Example execution of validating based LE.

can calculate a unique ID by sampling $\Theta(\log n)$ bits. The algorithm template works on an overlay graph of clusters, abstracted from the original graph. It consists of two parts:

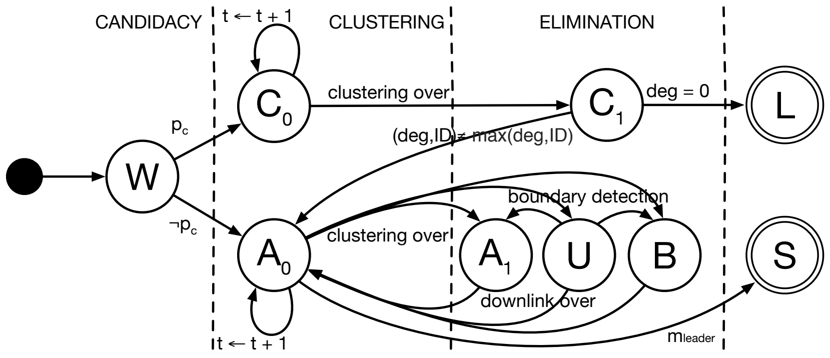

Figure 4.13: Clustering LE in RN and BE [32]

1. *Clustering*. A small set of candidate nodes are self-elected randomly w.p. $(10 \log n)/n$, and they become cluster centers. Clusters are formed by broadcasting candidate IDs, such that all clusters expand at the same speed. Non-candidates are assigned to the nearest candidate, and nodes with more than one nearest candidate are left unassigned. Boundary nodes play a special role in the algorithm. These nodes are adjacent to nodes from different clusters or unassigned nodes.
2. *Elimination part*. Candidates iterate over "debates", in which they exchange IDs, calculate their degree and mark themselves for elimination in case their degree or ID is dominated (i.e. it is smaller than another candidate's degree or ID respectively). A constant number of candidates is eliminated in each debate. After $\Theta(\log \log n)$ debates precisely one candidate remains and broadcasts its ID to claim the leadership. Debates are the central part of the template. They are enabled by three inter cluster operations: (1) uplink, sending messages from candidates to boundary nodes, (2) downlink, sending messages from boundary nodes to the candidates, and (3) intercommunication, exchanging messages between boundary nodes.

Figure 4.14 shows an example of this approach requiring 2 debates to find a single leader. It depicts four phases: (a) *Phase 1* shows the initial candidates with their chosen IDs of length $\ell = \lceil \log_2(18) \rceil = 4$,

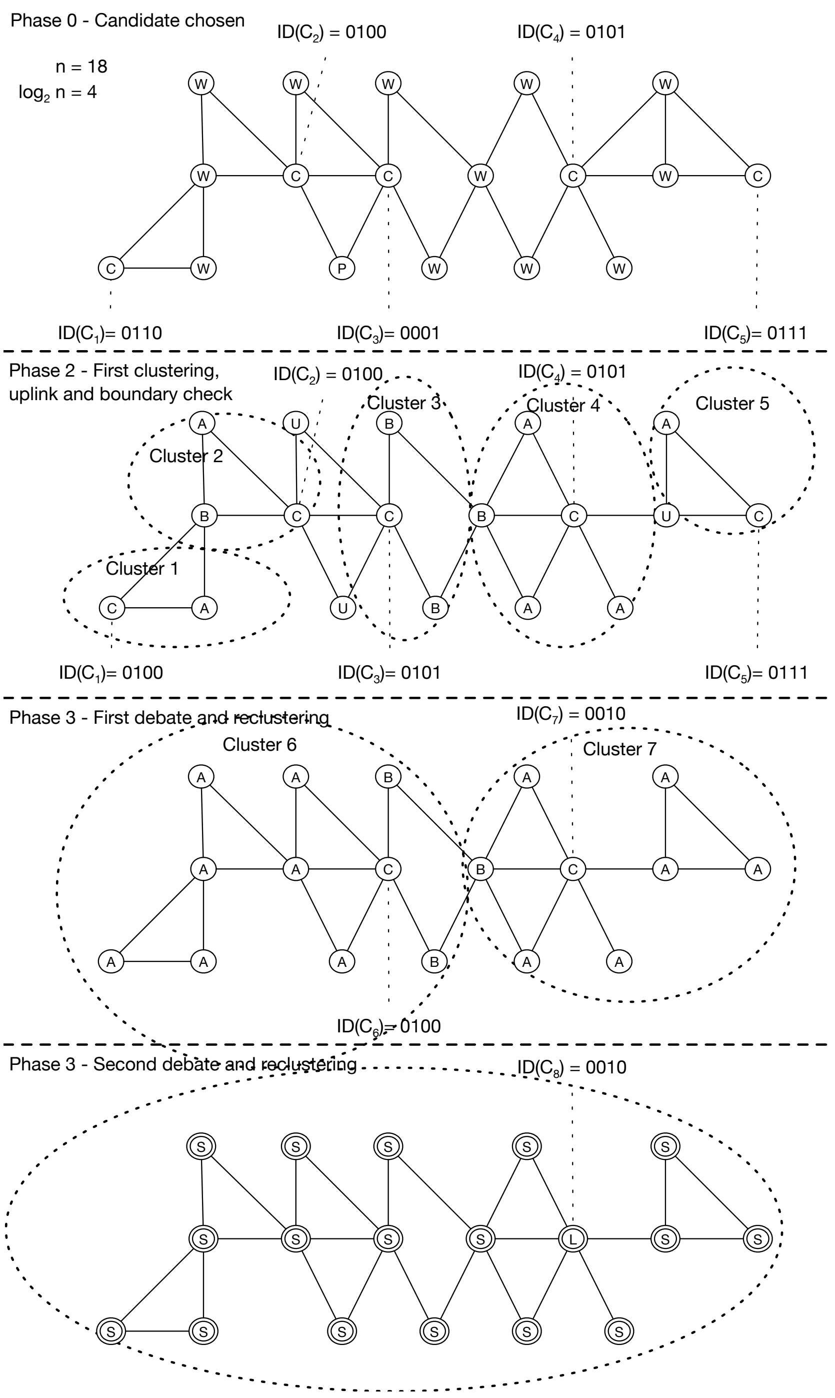

Figure 4.14: Example execution of clustering based LE.

(b) *Phase 2* shows the first clustering with some unclustered nodes, (c) *Phase 3* shows the result of the first debate: Clusters 2, 3 and 4 have degree two, clusters 3 and 4 tie in IDs. After the second clustering there are new clusters 6 and 7. (d) *Phase 4* shows the result of the second debate: Cluster 8 remains, since it has higher ID.

Two instantiations of this template were presented in [32] for the silent RN and BE models. Both the clustering and elimination parts were adapted to the conditions of each model.

1. A clustering based LE algorithm for RN without CD was presented by Ghaffari and Haeupler in [32]. The algorithm makes use of BC algorithms for both parts of the template. Initially a small set of candidates are elected randomly. Clusters are formed by performing multiple broadcasts of candidate IDs. Non-candidate nodes join the cluster of the first candidate ID they receive. There are different BC algorithms, as reviewed in Section 4.1. This algorithm requires clusters to have rather regular shapes. In particular this algorithm does not perform fast enough in case the clustering has "spikes", in which all adjacent nodes belong to different clusters. The trade off between two BC approaches was explained by Ghaffari and Haeupler in [32]. The *long decay BC* of Bar-Yehuda et al. [13] allows controlled growth while the fast decay is faster. The *fast decay BC* of Czumaj and Rytter [27] has optimal runtime, however it could produce spike-shaped clusters. The algorithm combines the two BC approaches to ensure a controlled cluster growth of one hope per phase. Once clustering is completed, the debate part is executed on the overlay graph of clusters. Candidates debate by exchanging their IDs, and they eliminate themselves in case they receive a higher ID. The exchanging of IDs require cluster operations. The uplink and downlink operations consist of multiple broadcasts initiated from the candidates and from the boundary nodes, respectively. As per the debate part, the fast decay BC is suitable for the uplink and downlink operations, whereas basic decay BC is enough for the intercommunication operation. The clustering total runtime is $\mathcal{O}(T_{BC}) + \mathcal{O}(\log^2 / \log(n/D)) \cdot \mathcal{O}(\log^2 n) = \mathcal{O}(D + \log n/D + \log^3 n)$ w.h.p. In addition, the uplink and downlink operations have runtime $\mathcal{O}(T_{BC})$, and the intercommunication operation has runtime $\mathcal{O}(D + \log n/D + \log^3 n)$. Lastly the debate runtime for RN without CD is further improved by limiting the number r of competing candidate IDs to $r = 5$, which guarantees a constant fraction $2/5$

of candidates is eliminated while remaining at least one.

2. Another clustering based LE algorithm for BE was presented by Ghaffari and Haeupler in [32]. The algorithm uses beeping waves for building clusters, in which nodes are numbered with the distance to the closest candidate. Waves proceed one hop per round, such that the elapsed time equals the distance to the candidate. Algorithm C.20 is an annotated version of this clustering algorithm for BE. Nodes use *superimposed codes* for estimating degrees by distinguishing the number of different messages received. Superimposed codes map a set of elements to a binary *codeword* such that any superimposition of k or less codewords is unique and distinct from any superimposition of more than k codewords. Candidates use superimposed codes for comparing degree and ID pairs, and deciding whether to mark itself for elimination or not. All three clustering operations are redefined such that nodes receive the superimposition of messages, as opposed to separate messages. Algorithms C.21, C.23 and C.22 are annotated versions the original algorithms. For improved efficiency, the algorithm uses $(1+\delta)-$*approximate* k*-counting* superimposed codes of length $\ell = \Theta(\log M \log k/\delta^3)$, with logarithmic dependency on M and k, and polynomial dependency on $1/\delta$. For calculating candidate degrees, this debate algorithm requires $\mathcal{O}(D+\ell)$ rounds to transmit IDs of length $\mathcal{O}(\log n)$ encoded with $\ell = \Theta(\log n \log\log n)$ bits. The number of required debates is $\min\{\log\log n, \log n/D\}$. Combining the two results, the runtime for LE is $\mathcal{O}(D+\log n)\cdot\mathcal{O}(\log^2 n \log n)$ w.h.p. Using approximation $\delta = 0.1$ and code length $k = \mathcal{O}(\log n)$, the number of different candidate IDs is correct by a $1+\delta = 11/10$ factor. Lastly, nodes compare their degree and ID with others by transmitting them as a bit stream, and they mark themselves in case they hear beep.

4.3.3 LE in the BE model

The time and state complexity to solve LE in a parameterized BE model was discussed by Gilbert and Newport in [35]. Firstly, the state complexity is obtained using a reduction to the $(1,k)$-loneliness detection problem w.p. $1-\varepsilon$, where k is the network size and ε an error bound. Loneliness defines two output states: q_a (i.e. “I am alone”) and q_c (i.e. “I am in a crowd”). For a network with size $n = 1$ the single node is required to enter the q_a state, and for networks with size $1 < n < k$ all nodes are required to eventually enter q_c. A lower bound is obtained based on the probability of violating such requirement, for the case of $n > 1$ this is the probability of some node reaching state q_a. This could happen after a sequence of state transitions of a given size r. Each node transitions w.p. $1/q$, where q is a precision parameter as defined in Subsection 2.4.1. The probability that all k nodes follow the same transition is $(1/q)^k$. The probability of all nodes following the entire sequence of r state transitions is $((1/q)^k)^r$. The error lower bound $(1/q)^{k \cdot r} \leq \varepsilon$ is obtained by making the previous probability smaller than the error ε. The inequality $q^{k \cdot r} \geq (1/\varepsilon)$ follows, along with $r \cdot k \ \log \ q \geq \log(1/\varepsilon)$. A lower bound of r is obtained from the previous inequality $r \geq \log(1/\varepsilon)/(k \ \log \ q)$. This yields a $\Omega(\log_q(1/\varepsilon)/k)$ lower bound for the number of states to solve loneliness in BE. The reduction of LE to loneliness is based on the pairs of rounds required for electing and announcing the leader, followed by a last round for terminating. In this last round any non-leaders should beep. The round is silent in case the node is alone, which then moves to state q_a. Otherwise all nodes move to state q_c. A lower bound of $s = \Omega(\log_q(1/\varepsilon)/\tilde{N})$ for the number of states to solve LE in the BE was obtained by Gilbert and Newport in [35] based on the reduction to the loneliness problem. The value of $\tilde{N}$ is a network size lower bound constant.

Secondly, a template algorithm for solving LE in the parameterized BE model was presented by Gilbert and Newport in [35]. The template consist of iterations of an elimination and a termination subroutines. Initially all nodes are active. During the elimination subroutine active nodes beep w.p. $1 - 1/\hat{q}$, otherwise they listen and become inactive in case they hear beep. The termination subroutine receives the active and knockout flags as parameters, and it returns a termination flag indicating that it is time to output. After the termination subroutine only one node remains active w.p. $1 - \varepsilon$, and it elects itself the leader. A safety condition is proven such that after $R = 4\log_{\hat{q}}(\max\{n, 1/\varepsilon\})$ silent knockout steps the probability of ending up with more than one leader is $\leq \varepsilon/2$. By union bound, the total error probability is $\leq \varepsilon$ [35]. The time complexity of the entire LE algorithm is $\mathcal{O}(tR)$, where t is the number of rounds required to run the termination subroutine. Three variants of the termination subroutine were presented by Gilbert and Newport in [35], which highlight the trade off between state and time complexity:

1. A *state optimal termination* has runtime $\mathcal{O}(\lceil \log_q(1/\varepsilon)/\tilde{N} \rceil)$. It disregards the active and knockout parameters. Instead, nodes try to beep w.p. $1/2$ for $\delta = \lceil c \ \log_{\hat{q}}(1/\varepsilon) \rceil$ rounds, for some constant $c \geq 1$.

After δ silent rounds nodes return true, otherwise they return false. The runtime grows exponentially with the network size n.

2. A *fast termination* fixes the precision parameter to the minimum $q = 2$ and it requires $\Theta(\log(1/\varepsilon))$ rounds. In the first round nodes that are knocked out beep to validates that there is at least one node. During $\lceil\log(2/\varepsilon)\rceil$ rounds active nodes beep w.p. $1/2$. Each node initializes an "alone" flag to true, and it switches it to false in case it hears a beep. In the final round, the active node with $solo = 1$ beeps. This validates there is at least one leader. The combined runtime for LE with a fast termination is $\mathcal{O}(\log(n + 1/\varepsilon)\log(1/\varepsilon))$ w.h.p.

3. A *constant termination* requires $\mathcal{O}(1)$ states and has runtime $\mathcal{O}(1)$ w.h.p. Similarly to the fast termination subroutine, nodes keep a "alone" flag, and they beep w.p. $1/2$. However, the phases last for a constant number of rounds. The final round is similar to the last round of the fast termination subroutine. Each attack iteration takes $\Theta(\log n)$ rounds. The error probability is $1/n^c$, for a constant c and the overall runtime is $\mathcal{O}(\log^2 n)$ w.h.p.

4.4 Vertex Coloring

This section reviews algorithms for vertex coloring in RN. Vertex coloring is a locally checkable problem. A typical application of coloring is exclusive assignment of shared resources, such as time and frequency. In particular, time slot assignment is useful in RN to avoid collisions. The rest of this section reviews coloring algorithms for three different RN models. Table 4.7 summarizes the complexity results.

Model	**Know-ledge**	**Graph family**	**Runtime**	**Ref.**
Harsh RN	n	$\mathcal{BG}$	$\Omega(\Delta + \log n)$	[54]
Harsh RN with CD	n	$\mathcal{BG}$	$\Omega(\Delta + \log n)$	[54]
nFSM	None	$\mathcal{BD}$	$\mathcal{O}(\log n)$	[30]
Harsh RN	n, Δ	$\mathcal{BI}$	$\mathcal{O}(\Delta \log n)$	[50]
Discrete BE	Q	$\mathcal{G}_U$	$\mathcal{O}(\log n)$	[25]

Table 4.7: Complexity of vertex coloring in RN.

An algorithm for $(\Delta + 1)$-coloring $\mathcal{BD}$ graphs in the nFSM model was presented by Emek et al. in [30]. Each node estimates its degree up to b, which is a model parameter. The algorithm proceeds in phases of $b + 1$ rounds: $b - 1$ rounds for estimating the degree, a round for broadcasting this estimate and a final round for running a randomized coloring subroutine. Only active nodes execute the subroutine, which consist of three steps: (a) choosing a color u.a.r. from its available palette, (b) broadcasting the chosen color, and (c) validating that the chosen color does not conflict with the ones chosen by its neighbors. Algorithm C.25 corresponds to the description presented in [30]. Figure 4.15 depicts the state transitions, with state set $\{A, W, C\}$ corresponding to active, waiting and colored. Similar to the MIS algorithm for nFSM, this coloring algorithm leverages the synchronizer defined in [30]. The algorithm has runtime $\mathcal{O}(\log n)$ w.h.p., since a constant number of nodes are colored at each round. The algorithm is not directly applicable neither to the harsh RN nor to the BE models.

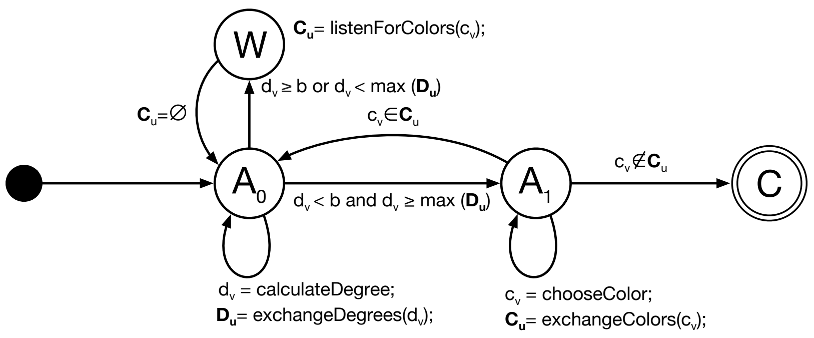
W
$\mathbf{C_u}$= listenForColors(c_v);
$d_v \geq b$ or $d_v <$ max ($\mathbf{D_u}$)
$\mathbf{C_u}=\varnothing$
$c_v \in \mathbf{C_u}$
A_0
A_1
$c_v \notin \mathbf{C_u}$
C
$d_v < b$ and $d_v \geq$ max ($\mathbf{D_u}$)
d_v = calculateDegree;
$\mathbf{D_u}$= exchangeDegrees(d_v);
c_v = chooseColor;
$\mathbf{C_u}$= exchangeColors(c_v);

Figure 4.15: $(\Delta + 1)$-coloring undirected trees in the nFSM model [30].

A randomized algorithm for $\mathcal{O}(\Delta)$ coloring $\mathcal{BI}$ graphs in an extended harsh RN was presented by Moscibroda and Wattenhofer in [50]. This model inherits the features of the harsh RN model, and it adds the assumption that nodes know the network size n and maximum degree Δ. In addition the model requires that nodes either have a unique ID or are able to generate a random ID that is unique w.p. $1/n$. In bounded independence $\mathcal{BI}$ graphs, two parameters κ_1 and κ_2 define the largest independent sets in the 1-hop and 2-hop neighborhoods respectively. The algorithm follows a similar approach to the MIS algorithm for harsh RN presented by the same authors in [49]. Both the MIS and the coloring algorithms use counters and critical ranges for breaking symmetry. Algorithms C.26, C.27 and C.28 corresponds to algorithms

1–3 in [50]. Figure 4.16 depicts the state transitions, with state set $\{R, A_i, C_i\}$ corresponding to requesting, active and colored, and i the chosen color. Upon wake up all nodes are in state A_0. The algorithm proceeds in three stages. In the first stage, a MIS of leaders is chosen and these move to state C_0. Other nodes are assigned to them forming clusters. These nodes move to requesting state R, in which they request an intra-cluster color from the respective leader. In the second stage, active nodes compete for the assigned color with their neighbors. Each node keeps track of its neighbors' counters and resets its own counter in case a neighbor's counter is within the critical range of it. In the third stage, colored nodes have moved to some colored state C_i, and transmit the chosen color i w.p. $1/(\kappa_2\Delta)$. This algorithm is not applicable to nFSM due to the requirement of keeping large counters in memory.

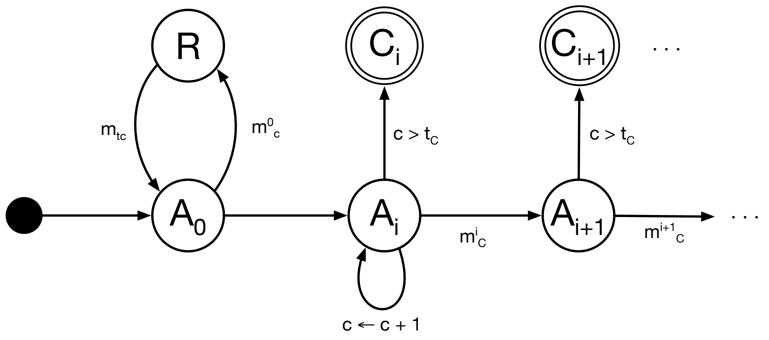

Figure 4.16: Coloring unstructured RN [50].

A randomized Las Vegas algorithm for $\Omega(T/\Delta)$-interval coloring in the standard BE model was presented by Cornejo and Kuhn [25]. In this model an adversary activates the nodes. The model assumes that nodes have local clocks with aligned boundaries, such that they proceed at the same rate without drifting. The algorithm proceeds in phases of Q slots of length μ, such that $Q \geq \Delta$ and each phase lasts for $T = Q\mu$. The $\Omega(T/\Delta)$-interval coloring problem consist of assigning disjoint time intervals to neighboring nodes. This problem was proven to be as hard as $\mathcal{O}(\Delta)$-vertex coloring in [25]. Nodes calculate an estimate of their degree $\tilde{d}_v$ by counting the number of beeps heard in a time period Q. Informally the algorithm assigns larger intervals to nodes with higher degree. The algorithm simulates collision detection by adding a random jitter, such that beeps are delayed by a random offset. In other words, nodes do not beep exactly within the time slot boundaries, rather the beeps span two time slots. Algorithm C.29 is a modification of the original algorithm. Figure 4.17 depicts the state transitions, with state set $\{U, C\}$ corresponding to uncolored and colored. Nodes start uncolored and switch to colored back and forward, depending on whether a neighbor has beeped within a safety interval b_v. The safety interval is three time slots, going from one slot before the currently assigned time slot to two slots after. This actual length of the safety interval is an inverse factor of the estimated degree. Uncolored nodes search for a free time intervals (i.e. without any beep) that are at least as large as the safety interval. They choose one free intervals u.a.r., and they try to claim it by beeping at that interval. The algorithm correctness was proven with the argument that uncolored nodes can find a free interval with constant probability. The algorithm does not terminate, rather it eventually converges to a correct interval assignment. The algorithm produces a $\Omega(T/\Delta)$-interval coloring in $\mathcal{O}(\log n)$ time, which matches the lower bound presented in [25].

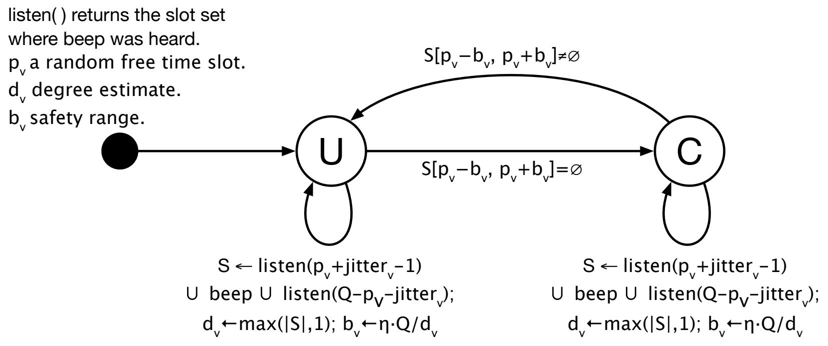
listen( ) returns the slot set
where beep was heard.
p_v a random free time slot.
d_v degree estimate.
b_v safety range.
$S[p_v-b_v, p_v+b_v]\neq\varnothing$
U
C
$S[p_v-b_v, p_v+b_v]=\varnothing$
$S \leftarrow$ listen(p_v+jitter$_v$−1) $\cup$ beep $\cup$ listen(Q−p_V−jitter$_v$); $d_v \leftarrow \max(|S|,1)$; $b_v \leftarrow \eta \cdot Q/d_v$
$S \leftarrow$ listen(p_v+jitter$_v$−1) $\cup$ beep $\cup$ listen(Q−p_V−jitter$_v$); $d_v \leftarrow \max(|S|,1)$; $b_v \leftarrow \eta \cdot Q/d_v$

Figure 4.17: Coloring in discrete BE [25].

Chapter 5

Population protocol model

Population protocol models (PP) are distinct from message-passing models. They eventually reach a correct solution, and they are not required to terminate. Analysing PP requires considering not only reachability but also convergence of correct configurations. Seminal research by Angluin et al. [5, 7–9, 11] characterized the capabilities of standard PP, and multiple variations of them. The self-stabilizing PP model is particularly relevant, since it allows modeling chaining of protocols. Sections 5.1, 5.2 and 5.3 discuss the aspects of eventual computability, determinism and self-stabilization. Sections 5.4 and 5.5 review the problems of 2-hop coloring and leader election (LE) in PP.

5.1 Eventual computability

Computability in PP models is closely related to eventual computability from a multiset of inputs. Angluin et al. [10] proved that the predicates eventually computable with PP models are precisely the same as semilinear predicates. The semilinear predicates are the sets defined by first-order Presburger arithmetic. These include: (a) the Boolean terms 0 and 1, (b) the operators $<$ and $+$, and (c) the logical quantifiers and operators. Equivalently, semilinear predicates are characterized by the Boolean combinations of threshold and modulo predicates.

Based on the previous characterization, an eventually computable expression language was presented by Angluin et al. in [7]. All predicates that can be expressed in this language are computable in PP. The language is defined by the Boolean combinations of comparison predicates involving two terms. Each term consists of constants, symbols and operations. Symbol represent counters, such as the number of occurrences in the multiset of inputs. The only valid operations in the language include sum, product, quotient and modulo. The remaining of this section reviews two example problems of modulo and threshold predicates, along with their corresponding protocols.

Parity is an example of modulo predicate. In the parity problem nodes receive an input $\{0, 1\}$. All nodes must output the sum of inputs modulo 2. Nodes keep two state variables: a live bit and a data bit. The live bit indicates whether its input has been counted or not. Protocol 5.1 is a transcription of the parity protocol from [7]. In case two live nodes with different counter interact, one sets its data bit to the sum of their two data bits modulo 2, and the other one turns off its live bit flag. This repeats until there are no adjacent live nodes. Through the second rule, live nodes disseminate to the parity result to the adjacent non-live nodes. For the live nodes, the sum of data bits modulo 2 remains equal to the sum of the inputs modulo 2. The protocol can be generalized to any modulo m on the number of 1's in the input. The protocol runtime is $\mathcal{O}(n^2 \log n)$ interactions [7].

Protocol 5.1: Parity in the standard PP model [7].

Constants: $k = 1$ /* To calculate $1/(k + 1) = 1/2$ of nodes have input 1 */
State variables: $live \in \{0, 1\}$ /*Live flag*/, $d \in \{0, 1\}$ /*data bit*/
Input: $I : \{0 \to (1, 0), 1 \to (1, 1)\}$
Output: $O : \{0 \to 0, 1 \to 1\}$ /*the output is the data bit*/
Transition Rules:
$((1, d_1), (1, d_2)) \to ((1, (d_1 + d_2) \mod 2), (0, d_2))$
$((1, d_1), (0, d_2)) \to ((1, d_1), (0, d_1))$

Majority is an example of threshold predicate. In the binary majority problem nodes receive an input $\{0, 1\}$. All nodes must output 1 in case more than half of the inputs are 1. Otherwise all nodes must output 0. Nodes keep two state variables, a live flag and a counter. The live flag indicates whether its input has been counted or not. The counter has values in the range $-k, \ldots, k$. Protocol 5.2 is a transcription of the majority protocol from [7]. In case two live nodes with different counter interact, they set their counter to the sum of both counters and one turns off its live flag. This repeats until all live nodes have the same counter value. Eventually the protocol stabilizes and the solution is disseminated from live nodes to the other nodes. This protocol generalizes to calculate whether at least a fraction $1/(k+1)$ of the inputs are 1's. Throughout the entire execution the sum of counters from live nodes remains equal to the difference between total number of 1's and the total number of 0's in the input. This approach generalizes to predicates involving quotient and modulo.

Protocol 5.2: Majority in the standard PP model [7].

```
Constants: $k = 1$ /* Parameter of population fraction $1/(k+1)$, which is $1/2$ for
           majority*/
State variables: $live \in \{0,1\}$ /*Live flag*/,
$c \in \{-k, \ldots, k\}$ /*counter*/
Input: $I : \{0 \to (1,-1), 1 \to (1,1)\}$
Output: $O : \{-1 \to 0, 0 \to 0, 1 \to 1\}$
Transition Rules:
$((1,c_1),(1,c_2)) \to ((1,c_1+c_2),(0,c_1+c_2))$ /*if $c_1 \neq c_2$*/
$((1,c_1),(0,c_2)) \to ((1,c_1),(0,c_1))$
/* $\mathcal{O}(n^2 \log n)$ interactions [7].*/
```

Similarly to MP models, the computability in PP models is highly dependent on the graph family. However, authors suggest that complete interaction graphs are the weakest setup, in which nodes are effectively indistinguishable by their connections. This claim is supported by the simulation presented by Angluin et al. in [7] of complete graphs that works for any connected graph. The simulation only requires an additional state variable with 4 possible values. For comparison purposes, complete interaction graphs correspond to single-hop in the radio network model (RN).

Lastly, a simulation of counters in the probabilistic PP model was presented by Angluin et al. in [7]. Up to $\mathcal{O}(1)$ counters of size $\mathcal{O}(n)$ are simulated with a population of n nodes, holding an $\mathcal{O}(1)$ array in memory. At a given time, the value of the i-th counter is the sum over the i-th elements of all node arrays. Given a LE protocol as input, the leader decreases or increases the counter by modifying the corresponding array element of a node. In addition, a protocol for validating that a counter is greater than zero was presented in [7]. The leader node first sets a "timer" mark in a node and then starts counting up to k interactions, such that k is a small constant. In case the timer node was found within k interactions, the counter is known to be greater than zero. Otherwise, the counter is assumed to be zero w.p. $\mathcal{O}((1/n)^{k-1} \log n)$. The protocol runs in $\Theta(n^{k+1})$ interactions w.h.p.

Leader Election (Deterministic)	**Coloring (Non Deterministic)**
1. $(1,1) \to (0,1)$ 2. $(1,0) \to (0,1)$ 3. $(0,1) \to (1,0)$ 4. $(0,0) \to (0,0)$	1. $(i,i) \to (j,k)$

Table 5.1: Determinism vs. non-determinism in PP.

5.2 Determinism in PP

Population protocol models have been defined under both types of assumptions, deterministic and non-deterministic. In both cases an adversary actives the interactions with a fairness condition assumption. The non-deterministic aspect refers to nodes having access to randomization. Table 5.1 shows this difference with two example protocols. On the left side there is a deterministic protocol to obtain a single leader. The state variable is a leader flag $\ell \in \{0,1\}$, and initially all nodes are set to $\ell = 1$. The first transition rule eliminates two interacting candidates, until eventually one candidate remains. The second and third rule circulate the leader flag, ensuring that the protocol works for non-complete and directed graphs. On the right side there is a non-deterministic protocol for coloring. There is only one transition rule for breaking invalid conditions, in which two nodes have the same color. In this case both nodes choose another color u.a.r. Eventually all nodes have different colors.

The computability for non-deterministic and deterministic protocols was proven to be the same by Angluin et al. in [11]. The proof leverages the previous LE protocol, such that leaders make a deterministic choice by iterating over the possible interactions in a round-robin fashion. Protocol 5.3 is a transcription of the original determinizer protocol. The state variables include the leader token ℓ, the simulated protocol states $q, q' \in Q$ and the stabilizing inputs $x, x' \in X$.

This deterministic protocol for LE succeeds in reducing the number of leaders down to one, however it is not able to create one in case there are no leaders in the input. In other words it is not a self-stabilizing protocol. The following section reviews self-stabilization in PP.

Protocol 5.3: Determinizer protocol in the standard PP model [11].

```
Constants: $m = \Delta$ /*max degree*/
State variables: $\ell \in \{*, -\}$ /*leader flag*/, q /*the current state in the simulated
    protocol*/ $c \in \{1, \ldots, m\}$ /*choice counter*/
Initialize: $token \leftarrow *$; $c \leftarrow 1$
Transition Rules:
$((*qc, x), (*q'c', x')) \to (-qc, *q'c')$
$((*qc, x), (-q'c', x')) \to (-qc, *q'(c'+1))$
$((-q'c', x'), (*qc, x)) \to (*q'(c'+1), -qc)$
$((-qc, x), (-q'c', x')) \to (-rc, -r'c')$
/*$r = \delta[c]$ and $r' = \delta[c']$ respectively. All increments are modulo $m$*/
```

5.3 Self-stabilizing PP

The study of self-stabilization in population protocols was initially motivated by modeling interactions of a protocol with an environment, a user or other protocols. Self-stabilization allows composition (i.e. chaining) of protocol outputs as inputs to other protocols. Furthermore protocol composition is utilized for algorithmic reduction in PP. Formal definitions of the self-stabilizing PP model were presented by Angluin et al. in [5]. This utilizes the concept of *stable behavior* as defined in Section 2.5, a set of traces having a common sequence prefix after which the traces are equivalent. A trace is a sequence of assignments from agents to an alphabet (i.e. input X or output Y alphabets). A protocol that implements a stable behavior B^S is a self-stabilizing implementation of behavior B.

Protocol composition is performed by concatenating the state information of both protocols. A protocol $P1 \circ P2$ composed of P_1 and P_2 defines states with two parts, corresponding to the state of each protocol. Protocol P_2 has access to the state part that is updated by P_1. An example of self-stabilizing protocol for token circulation in rings given a LE behavior was presented by Angluin et al. in [5]. The LE behavior state has a leader flag, and the state for the token circulation behavior has a token variable and an active flag. Protocol 5.4 is a transcription of the original protocol.

Additionally, the stabilizing PP model has been utilized to study decision problems of graph properties. Positive results were presented by Angluin et al. in [5], showing that protocols with stabilizing inputs can decide whether a network topology corresponds to a directed line, cycle, star, tree or bounded degree graph. The results are proved by

Protocol 5.4: Self-stabilizing token circulation - Protocol 1 in [11].

Input: LE behavior
State variables: (t, b, ℓ) where $t \in \{\circ, \bullet\}$ /*token flag*/, $b \in \{0, 1\}$ /*active flag*/,
$\ell \in \{1, 0\}$ /*Leader flag*/
Transition Rules:
$((tb, \ell = 0), (tb, \ell = 1)) \to (\circ b, \bullet \bar{b})$
$((tb, \ell), (t\bar{b}, \ell = 0)) \to (\circ b, \bullet b)$

making use of a leader node, which places marks on neighbors and keeps track of the number of marks it has placed. Furthermore, self-stabilizing protocols were proven to solve any predicate in weakly connected graph that is also possible to solve by self-stabilizing protocols in complete graphs of the same size [5]. Additionally, they were proven to be able to compute semilinear predicates (1st order Presburger arithmetic) in complete graphs. Consequently, they are as powerful as the standard PP model.

5.4 2-hop coloring

A deterministic and a non-deterministic protocol for 2-hop coloring were presented in [11]. The protocol is designed for arbitrary graphs (i.e. not only for complete graphs). A full palette of $g = \Delta(\Delta - 1) + 1$ colors is used, where d is the maximum node degree. The key difference between the two protocols is in the color selection. Both protocols define states with two components: $color_u$ the current node color and F_u a color availability Boolean array of size g, indicating whether each color is assigned or available. The protocols define two local conditions: (1) *invalid* is the case that a node v is connected to two neighbors u and w that have the same color $color_u = color_w$, and (2) *aligned* is the case that two connected nodes have the same availability flag for the color of each other $F_u[color_v] \neq F_v[color_u]$. In addition, a global condition defines *legitimate configuration* the case in which all nodes are aligned and no node is in the invalid condition. Protocol 5.5 is a transcription of the deterministic protocol.

The protocol defines two cases for interactions, depending whether the nodes are aligned or not. In the first case, the initiator gets recolored and aligned, this ensures that nodes will get a different color. In the second case both nodes flip the availability flag corresponding to the color of the other node. Switching the availability flags has the purpose of detecting invalid conditions. Invalid conditions involve two conflicting nodes (i.e. with the same color) connected to a common neighbor. After the common neighbor and a conflicting node switch their flags, the other conflicting node becomes unaligned. The unaligned node gets realigned through case one in its next interaction, in which it chooses some other color. In the deterministic protocol, nodes iterate over the full palette incrementing the current color or not, according to control flag r_u. The control flag switches between each interaction, such that u gets recolored in case $r_u = 1$, otherwise node u simply gets aligned. The non-deterministic protocol is slightly different: nodes choose colors u.a.r. Both protocols self-stabilize to a legitimate configuration.

Protocol 5.5: Deterministic 2-hop coloring in stabilizing PP[11].

```
**Constants**: $g \leftarrow d(d-1)+1$ /* color palette */
**State variables**: $\mathsf{c}_u$ /*Color of node $u$*/, $F_u$ /*Bit array of size $g$*/
**if** $F_u[\mathsf{c}_v] \neq F_v[\mathsf{c}_u]$ **then** /*Unaligned*/
    $\mathsf{c}_u \leftarrow (\mathsf{c}_u + r_u) \mod g)$ /*For non-deterministic choose u.a.r. $\mathsf{c}_u \leftarrow \mathsf{c}'_u$*/
    $F_u[\mathsf{c}_v] \leftarrow F_v[\mathsf{c}_u]$ /* Aligned */
**end**
/* Detect conflicts */ $F_u[\mathsf{c}_v] \leftarrow \overline{F_u[\mathsf{c}_v]}$
$F_v[\mathsf{c}_u] \leftarrow \overline{F_v[\mathsf{c}_u]}$
$r_u \leftarrow 1 - r_u$/* Only applies to deterministic*/
```

5.5 Leader election

A randomized protocol for solving LE in the probabilistic PP was presented by Angluin et al. in [7]. It follows the simulated counter approach (Section 5.1), using a timer to determine that there is only one leader remaining. The state is a tuple (x, c, q, m), corresponding to the input, counter, state and mark variables. The input value is a non-negative integer. The counter is an integer c in the range $\{0, \ldots, k\}$. The state set is $q \in \{C, D, I, W, V, L\}$, corresponding to candidate, dominated, initializing, waiting, validating and leader. All states other than dominated are live states. The mark value set is $m \in \{t, i, -\}$, corresponding to timer mark, initialization mark or none. Protocol 5.6 corresponds to the description presented in [7]. All nodes start as candidates and proceed with two phases: initialization and validation. In the initialization phase, candidates set the timer mark $m \leftarrow t$ to the first node they interact with, which becomes dominated. It then proceeds to count up to k

interactions, in which the responder nodes set the initialization mark and become dominated as well. In case a dominated node had placed a timer, the candidate waits for an interaction with such timer before restarting its counter. The input variable remains unchanged during the initialization phase. In the validation phase, leader candidates simulates a counter machine by decreasing the input variable until the global counter is zero. Similarly, in case the candidate finds a live node, one gets dominated and the other restarts the initialization phase after waiting for a timer mark in case the dominated node had placed one. The protocol requires $\Theta(n^2)$ interactions to converge w.h.p. [7].

Protocol 5.6: LE in the self-stabilizing PP model [7].

```
State variables: $x$; /*input*/ $c$; /*counter*/ $q$; /*state*/ $m$; /*mark*/
Initialize: $x_i \to x_i$; $c_i \leftarrow 0$; $q \leftarrow L$; /*all nodes are live*/; $m \leftarrow -$
/*Initialization phase*/
$((x_1, c_1, L, -)(x_2, c_2, q_2, -)) \to ((x_1, c_1 + 1, \texttt{nextState}(q_2), -)(x_2, 0, D, t))$
$((x_1, c_1, I, -)(x_2, c_2, q_2, -)) \to ((x_1, c_1 + 1, \texttt{nextState}(q_2), -)(x_2, 0, D, i))$
/*Wait for a timer mark*/
$((x_1, c_1, W, -)(x_2, 0, D, t)) \to ((x_1, c_1 + 1, I, -)(x_2, 0, D, i))$
/*Validating phase*/
$((x_1, k, I, -)(x_2, 0, D, *)) \to ((x_1, 0, V, -)(x_2, 0, D, *))$
$((x_1, c_1, V, -)(x_2, 0, D, *)) \to ((x_1, 0, V, -)(x_2 - 1, 0, D, *))$ /*s.t. $x_2 > 0$*/
$((x_1, c_1, V, -)(x_2, c_2, q_2 \neq D, *)) \to ((x_1, 0, \texttt{nextState}(q_2), -)(x_2 - 1, 0, D, i))$
/*Leader found*/
$((x_1, c_1, V, -)(0, 0, D, *)) \to ((x_1, 0, L, -)(0, 0, D, *))$
nextState($q_u$)
    if $q_u = I$ then return($W$)
    else return($I$)
end
```

Lastly, a self-stabilizing PP for LE in oriented cycles of odd size was presented by Angluin et al. in [11]. Protocol C.30 is a transcription of the original protocol. The input is a value in the range $\{0, \ldots, k\}$ with $k = 2$. The protocol defines two concepts: (1) *alternating sequences* with values $\langle 0, 1, \ldots, k, 0, 1, \ldots \rangle$, and (2) *barriers*, which are edges connecting two nodes with the same input (i.e. delimiting sequences). Intuitively the protocol shifts barriers clockwise repeatedly until they reach another barrier. Two adjacent barriers cancel out, merging the corresponding sequences. Eventually a single barrier remains. This is always the case for an $k = 2$ in odd size cycles. Additionally, two auxiliary marks *probes* and *bullets* are sent around the ring clockwise from barriers and counterclockwise from leaders respectively. Probes ensure that there is at least one leader, since a new leader is generated in case a probe reaches a barrier. Bullets ensure that only one leader is elected, since a leader removes its leader mark in case it is reached by a bullet. This protocol generalizes for any $k \geq 1$ that is a relative prime to n, such that a barrier edge (u, v) connects nodes with consecutive labels $label_u + 1 \equiv label_v \mod k$.

Chapter 6

Comparative analysis

This chapter makes a comparison between the variants of radio network (RN) models, and between RN and population protocols (PP). The comparison between RN models is based on three model features. Section 6.1 compares the synchronization and activation features. It discusses the crucial challenge of non-simultaneous activation. It reviews different approaches followed by models of previous literature to circumvent the challenge. Section 6.2 compares the model capabilities with regard to collision detection. It explains the advantages of having additional information provided by collision detection and carrier sensing. Section 6.3 analyses the key aspects contributing to runtime complexity from the perspective of determinism. Lastly, Section 6.4 discusses the key differences between RN and PP from a general perspective.

6.1 Synchronization

Synchronization is a crucial feature in RN models affecting the algorithmic computational complexity. There are two interleaved aspects of synchronization: activation policy and scheduling policy. Section 2.3 provides descriptions for these two aspects. In particular, the simultaneity aspect of activation plays an important role in algorithmic complexity. In the case of non-simultaneous activation, newly activated nodes may interfere the communication between neighboring nodes, causing delays in the algorithm. Scheduling policy is closely related to activation simultaneity, since adversarial scheduling implies non-simultaneous activation. RN algorithms follow different approaches to cope with this challenge, depending on the model assumptions for the two aspects. The approaches include simulation of global clocks, "pulses" (periodic beats) and counter thresholds.

Simulating a global clock is possible for models with activation by incoming messages. A wave-based approach to simulate global clocks in was presented by Ghaffari and Haeupler in [32]. The wave carries a time counter, starting from the nodes initially active at time 0, and incrementing the counter by one per hop. As nodes receive the counter, they know how many rounds the local clock is behind time 0. It takes D rounds for the counter to reach all nodes in the network. However for BE it would require $\mathcal{O}(D^2)$, since D rounds are required to send a counter as a bit stream (i.e. one bit per round). Instead, a two wave approach for global clocks in BE was presented by Ghaffari and Haeupler in [32]. This consists of sending two waves from the initially active nodes each at different speed. The first wave progresses by one hop per round, while the second wave progresses by one hop every two rounds. Figure 6.1 shows the simulation in three example graphs. The approach leverages the additive identity of beeps, which is not applicable to RN without CD.

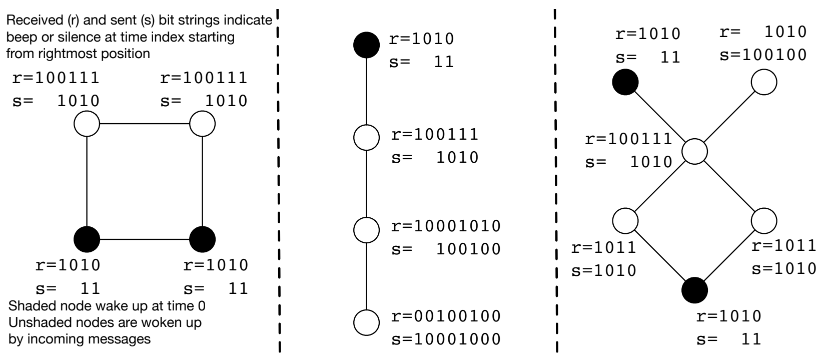

Figure 6.1: Simulating global clocks in BE in three example graphs.

A partial synchronizer based on periodic beats was presented by Emek et al. in [30] for the nFSM model. The model is representative of activation by adversary. The synchronizer obtains a soft alignment, also known as locally synchronous environment. It consist of an additional "pulse" state variable $t \in \{0, \ldots, k\}$ for $k = 3$. The pulse variable is exchanged in each communication round. Nodes use the received pulse values to decide whether to proceed with the algorithm or to wait for soft alignment. Soft alignment for a node u is defined such that any neighbor v is at most one round behind $t_v = t_u$ or $t_v = t_u - 1 \mod k$. A similar soft alignment technique for the wake on beep BE is utilized by Schneider and Wattenhofer in [54]. They presented a MIS algorithm that works in phases of six rounds. Newly woken nodes wait for seven silent rounds before starting the algorithm. In addition, a MIS algorithm based on pulse beats was presented by Afek et al. in [1] for the synchronized clocks BE. The algorithm designates every third round for a restart bit. The restart bit ensures that neighboring nodes are aligned with the same estimate of network size. Having the same estimate is a requirement for the algorithm to produce a valid MIS. In general, the goal of a partial synchronizer is similar to the goal of global clock simulation, both attempt to align local clocks.

Conversely, the counter thresholds approach attempts to separate local clocks. A MIS algorithm that follows this approach was presented by Moscibroda and Wattenhofer in [49] for the harsh RN model. The model defines adversarial activation, and the algorithm assumes $\mathcal{UD}$ graphs. The algorithm utilizes multiple counter thresholds as filters for advancing states from waiting to active, from active to competing and from competing to MIS. Section 4.2 reviews this algorithm. After nodes wake up, they wait for t_W rounds in order not to interfere with competing nodes. Another threshold t_A is used to break symmetry between active nodes for becoming candidates. Two more thresholds t_C and t_M are used to separate node clocks, such that only nodes with a clear advantage (i.e. counter delta) over the neighbors join the MIS. The threshold values are crucial for the algorithm correctness and runtime complexity.

6.2 Collision detection

This section discusses another key separating aspects of RN, namely collision detection. Collision detection (CD) is another key feature affecting algorithmic complexity of RN. Detecting a collision provides more information than not detecting it. The same applies to carrier sensing, which is the case of BE. Nodes are able know whether there is traffic in the shared channel or not. This additional information is leveraged both for encoding a single message and for detecting the number of messages transmitted.

A deterministic BC algorithm that encodes the broadcasted message with collisions was presented by Chlebus et al. in [21]. Informally, the algorithm follows the approach of simulating beeping in the standard RN (with CD). The algorithm encodes messages as a binary stream, such that silence represents a 0 and either a collision or a single message received represents a 1. The BC problem separates the runtime of deterministic algorithms for RN with CD $\mathcal{O}(n \cdot D)$ [21] and without CD

$\Omega(n)$ [13, 42]. A similar separation exists for randomized BC algorithm.

In addition, there are algorithms for the standard RN that allow sender nodes to overlap multiple messages and receiver nodes to detect the number of messages. This technique is utilized for solving LE in RN in the two approaches reviewed in Section 4.3, namely validating and clustering. For each approach, the referenced authors presented algorithms for the silent RN and wake on beep BE models. Both approaches require nodes to exchange candidate IDs. In both LE algorithms for the BE model, multiple candidate IDs are transmitted simultaneously encoded with superimposed codes. Receivers decode the superimposed codes to detect the number of IDs transmitted. Similar to BC, the LE problem separates the runtime for silent RN and wake on beep BE: (a) In the validating approach the runtime for LE in the silent RN is $\mathcal{O}(T_{BC} \cdot \sqrt{\log n})$ w.h.p., whereas the runtime for LE in wake on beep BE is $\mathcal{O}(D + \log n)$ in expectation. For this approach, the silent RN algorithm presented by Czumaj and Rytter in [26] requires multi-BC of entire IDs and a subsequent decay based bit-wise BC of IDs. Conversely, their algorithm for wake on beep BE sends the superimposition of IDs as a single bit stream. (b) In the clustering approach the runtime for LE in the silent RN is $\mathcal{O}(D \log n/D + \log^3 n) \cdot \min\{\log\log n, \log n/D\}$ w.h.p., whereas the runtime for wake on beep BE is $\mathcal{O}(D + \log n) \cdot \mathcal{O}(\log^2 n \log n)$ w.h.p. For this approach Ghaffari et al. proved in [32] it is enough for candidates to capture a subset of IDs. However, this optimization is neither applicable nor necessary for the wake on beep BE algorithm.

Typically, collision detection benefits only receivers. RN with CD and BE models assume that nodes are able to either transmit or receive at a given time, and not both at the same time. A slightly stronger BE model was defined with the additional assumption of sender side CD. In this model, transmitter nodes are able detect other transmitters. This subtle difference differentiates this model from the standard BE model in the case that n is unknown. The two MIS algorithms presented by Afek et al. in [1] highlight the difference. Sender-side CD is used to estimate the probability of a safety guarantee, such that after two silent rounds nodes joining the MIS are part of a valid solution w.h.p. Otherwise, nodes that detect a neighbor joined the MIS (i.e. hear a beep in each of the two rounds) decide not to join the MIS. Conversely, in the standard BE model, without sender-side CD, nodes are not able to detect such condition. In this case, it is likely that neighboring nodes with the same beeping sequence join the MIS. This requires nodes that joined the MIS to continue beeping at regular intervals to claim their place in the MIS.

Lastly, simulating CD in RN without CD has been studied by Bar-

Yehuda et al. in [14] and by Kowalski and Pelc in [40]. In the former the simulation is limited to simulating single-hop RN with CD by adding an overhead of $\mathcal{O}(D \log n + \log^2 n)$. In the later the simulation does not restrict the graphs, however it has a larger overhead of $\mathcal{O}(n \min\{D, \log n\})$.

6.3 Determinism

This section analyses the model capability separation due to determinism in RN. Determinism separates the solvability and computational complexity in RN. The solvability separation is clear for anonymous models. Impossibility results from the anonymous PN model apply to anonymous RN. This includes not only the classical problems of MIS, LE and coloring but also the fundamental operation of BC. Consider a 4 node cycle with a source node. The two adjacent nodes have precisely the same inputs and neighborhood. Their execution is the same at any round, both nodes either listen or transmit, causing a collision. Consequently, the message never reaches the uninformed node.

The BC and LE problems separate the computational complexity of deterministic RN. The separation for the BC problem relies in the approaches for coping with collisions. Deterministic BC algorithms avoid collisions completely by calculating an exclusive transmission schedule in advance. It is based on the concept of selective families and the assumption of unique IDs. Nodes transmit only in case their ID belongs to the family corresponding to the current round. Randomized BC algorithms make use of transmission probability sequences in order to control the amount of collisions. Section 4.1 reviews three different randomized BC algorithms. The differences in their runtime complexity derives from the different transmission probability sequences:

1. The decay BC algorithm of Bar-Yehuda [13] utilizes a sequence that doubles the probability as the number of competing nodes is halved. It does not distinguish the network diameter.
2. The “optimal” BC algorithm of Kowalski and Pelc [40] utilizes two intertwined sequences. It improves the performance of decay BC by considering separately the case of networks of large diameter. The first sequence has length $\log(N/D)$, and it is meant to inform nodes with less than N/D informed neighbors. It starts from 1 and it gets halved at each round. The second sequence is based on universal probability sequences, and it is meant to inform nodes with more than N/D informed neighbors.

3. The selecting sequences BC algorithm of Czumaj and Rytter [27] utilizes a parameterized transmission probability sequence. It specializes in “shallow” networks of small diameter $D \leq \log^3 n$.

The different probability sequences mark a difference in runtime complexity of randomized BC. The results summarized in Table 6.1 highlight the separation.

Problem	**Model**	**Deterministic**	**Non-Deterministic**
BC in $\mathcal{G}_U$	Silent RN	$\Omega(n)$ [13, 42] and $\Omega(D \log n)$ [17]	$\Theta(D \log n/D + \log^2 n)$ [3, 40, 43]
BC in $\mathcal{G}_D$	Silent RN	$\Omega(D \log n)$ [21] and $\Omega(n \log D)$ [24]	$\Theta(D \log n/D + \log^2 n)$ [24, 27]
BC in $\mathcal{G}_D$	Std. RN	$\Omega(n \cdot D)$ [21]	$\Omega(D + \log^6 n)$ [33]
LE in $\mathcal{G}_U$	Silent RN	$\Theta(n)$ [22]	$\mathcal{O}(n \log^{3/2} n \sqrt{\log \log n})$, $\Omega(n \log n)$ [22, 41]
LE in $\mathcal{G}_U$	BE	$\mathcal{O}(D \log n)$ [31]	$\mathcal{O}(D + \log n) \cdot \mathcal{O}(\log^2 n$ $\log n)$ w.h.p. [32]

Table 6.1: Runtime complexity separation by determinism in RN.

Although typically randomized algorithms have faster runtimes, sometimes they are only able to output correct solutions w.h.p. (Monte Carlo). Contrarily, deterministic algorithms are able to guarantee correct solutions all times. This can be observed in the symmetry breaking mechanism for generating unique IDs. Randomized algorithms can only guarantee unique IDs w.h.p. An example of deterministic algorithm that uses unique IDs is the algorithm for solving MIS in the standard RN, presented by Schneider and Wattenhofer in [54]. The algorithm is directly applicable to the BE model, since nodes exchange IDs bit by bit. It requires only $\mathcal{O}(\log n)$ to output a correct solution and terminate. Conversely, randomized algorithms require a different approach to solve MIS with the same guarantees (Las Vegas) in the same model. A randomized algorithm for solving MIS in the wake on beep BE with CD was presented by Afek et al. [1] The algorithm does not generate IDs but it uses counters for breaking symmetry. It requires $\mathcal{O}(\log^2 n)$ to make output a correct solution and terminate.

6.4 Communication mechanism

The last section of this chapter analyses the differences between RN and PP based on model features. The communication mechanism of PP models is fundamentally different from MP models. Particularly, aspects related to scheduling policy separate the model classes. An adversary enables communication edges dynamically. Consequently PP models are not able to terminate, rather PP models can only eventually converge to correct outputs. Another key difference is that determinism does not separate PP. Non-deterministic protocols can be simulated with deterministic protocols using a leader token. A determinizer protocol was reviewed in Section 5.2. Furthermore, there are no collisions in PP, since there is only pairwise communication.

Nevertheless, there are commonalities in the model classes from a general model features perspective. Nodes in PP models exchange state information, however this is not necessarily a limitation. State information might include not only state labels but also counters and other variables. Another potential limitation of PP models is that communication is restricted to pairs of nodes. In this regard, PP are similar to PN models, as in both models communication is directed towards a neighbor. The ability to send messages to all neighbors is provided by the fairness condition of PP. The condition guarantees that any possible interaction should be enabled repeatedly often. Lastly, in the standard PP model, nodes have limited memory, which is similar to the nFSM and BE models. However variations of PP models have considered a memory size logarithmic in n.

This concludes the comparative analysis between RN and PP. The conclusions chapter provides an overall summary of the thesis. It remarks the key results of this analysis, and it provides open questions and directions for future research.

Chapter 7

Conclusions

This thesis reviews and compares the computability of two types of wireless distributed computing models, namely radio networks and population protocols. Radio networks (RN) are a subtype of message-passing models, in which nodes communicate by sending messages through a shared channel exposed to collisions. Population protocols (PP) are a different type of wireless models, in which nodes communicate through interactions by proximity as opposed to sending messages. The thesis provides a thorough introduction to model features that have been defined in previous literature. A first contribution is a clear organization of model types and classes, including the presented feature map for RN.

The analysis starts by highlighting model capabilities that RN inherit from the stronger port numbering (PN) model. Impossibility results related to symmetry breaking hold equally for deterministic anonymous networks in both models. A crucial separation exists between deterministic and non-deterministic algorithms in anonymous networks. The separation is closely related to symmetry breaking. Unique IDs and randomization have been used in both models for overcoming graph symmetry. Typically, deterministic RN algorithms have assumed unique IDs, whereas randomization has been utilized for generating IDs that are unique w.h.p. In the later case there is a trade off between output correctness (Las Vegas) and runtime (Monte Carlo) guarantees.

The thesis further discusses the adaptability of algorithms designed for the port numbering (PN) model to RN models. PN algorithms that make use of labeled ports were excluded from the analysis, since they are not applicable to RN networks. Applicability of PN algorithms that are suitable for RN networks strives in sending the same message to all neighbors (as in broadcast), and coping with collisions. For the BE only algorithms that use bit size messages are directly applicable with similar runtime complexity. Similarly, for nFSM the bounded memory restriction discards the applicability of algorithms that utilize large messages. In addition, nodes in RN share a common channel, causing collisions in case neighboring nodes transmit in the same round. Collisions prevent successful broadcasts and affect the runtime complexity. The complexity overhead depends on the underlying model assumptions and algorithmic approaches for dealing with collisions.

Broadcasting (BC) is a fundamental operation in RN required for sending information across the network. The crucial challenge of BC is finding a transmission schedule with a controlled number of collisions. Efficient transmission scheduling is particularly relevant for broadcasting. Deterministic algorithms use unique IDs and selective families calculated in advance for deciding a transmission scheduling that avoids collisions completely. Conversely, randomization can be used for deciding whether to transmit or not at a given round, such that the number of collisions is controlled and correct results are obtained w.h.p. RN algorithms require assuming either knowledge of the network size or unique IDs to cope with collisions. The runtime complexity for deterministic algorithms is linear in the network size, whereas for randomized algorithms is logarithmic in the network size. Studying the complexity of BC is particularly relevant, since it is used as subroutine for solving other non-local problems, such as leader election (LE). The computational complexity of BC in RN is embedded in the computational complexity of LE.

Activation simultaneity and collision detection (CD) are crucial features affecting the runtime complexity in RN. Non-simultaneity creates additional overhead depending on the activating factor. In the case of activation by incoming message, it is possible to simulate global clocks with an additive D overhead. As per the case of activation by adversary, only a soft alignment is possible. Informally this approach consists on defining a schedule with periodic beats. The approach was applied in nFSM for solving locally checkable problems such as MIS and coloring. However, for the maximal matching problem it is only possible to converge to correct solutions with soft alignment.

Collision detection and carrier sensing separate the model capabilities of RN and BE models. They provide additional information that can be used for encoding a message and for detecting the number of messages received. In both the standard RN and BE models, it is possible to broadcast a message encoded with collisions. The runtime is a function of the network diameter and the message size. Furthermore, carrier sensing allows encoding information as bit streams. Using superimposed codes it is possible for transmitters broadcast multiple streams and for receivers to detect the number of received messages. LE algorithms utilize superimposed codes for detecting the presence of multiple candidate IDs. In general, algorithms for the standard RN (with CD) and BE have faster runtimes.

Population protocols are fundamentally different than RN models. An adversary enables the pair-wise communication dynamically. Consequently, protocols converge to a correct solution, as opposed to terminating. Population protocols are not exposed to collisions, since communication happens in pairs. Determinism does not impact the solvability of PP models given the determinizer approach presented in previous literature. Informally it delegates the non-deterministic choice of transition to the adversary. By definition, population protocols are provided with symmetry breaking mechanism, which is the directed interaction relation from initiators to responders. Lastly, protocol computability is focused in complete interaction graphs. Protocols running in a complete graph can simulate a complete graph with two tokens moving around the graph.

Multiple research questions are open for future research. There is room for research in relation to a comparable runtime complexity metric. This metric shall compensate that a round of RN gives all nodes a chance to communicate, whereas an interaction in PP models allows only a pair of nodes to interact. Having a comparable runtime metric, it is interesting to compare the complexity runtimes between RN and PP models for classical problems such as BC, LE, MIS and coloring. Studying state complexity of RN is another interesting area. Only one of the reviewed studies analyses state complexity. Comparing state complexity between various RN and PP models will lead to better understanding of their capabilities.

Bibliography

[1] AFEK, Y., ALON, N., BAR-JOSEPH, Z., CORNEJO, A., HAEUPLER, B., AND KUHN, F. Beeping a maximal independent set. *Distributed Computing 26*, 4 (2013), 195–208.

[2] ALON, N., BABAI, L., AND ITAI, A. A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms 7*, 4 (1986), 567–583.

[3] ALON, N., BAR-NOY, A., LINIAL, N., AND PELEG, D. A lower bound for radio broadcast. *Journal of Computer and System Sciences 43*, 2 (1991), 290–298.

[4] ANGLUIN, D. Local and global properties in networks of processors. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing* (1980), ACM, pp. 82–93.

[5] ANGLUIN, D., ASPNES, J., CHAN, M., FISCHER, M. J., JIANG, H., AND PERALTA, R. Stably computable properties of network graphs. In *Distributed Computing in Sensor Systems*. Springer, 2005, pp. 63–74.

[6] ANGLUIN, D., ASPNES, J., DIAMADI, Z., FISCHER, M. J., AND PERALTA, R. Urn automata. Tech. rep., DTIC Document, 2003.

[7] ANGLUIN, D., ASPNES, J., DIAMADI, Z., FISCHER, M. J., AND PERALTA, R. Computation in networks of passively mobile finite-state sensors. *Distributed Computing 18*, 4 (2006), 235–253.

[8] ANGLUIN, D., ASPNES, J., AND EISENSTAT, D. Stably computable predicates are semilinear. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing* (2006), ACM, pp. 292–299.

[9] ANGLUIN, D., ASPNES, J., EISENSTAT, D., AND RUPPERT, E. On the power of anonymous one-way communication. In *Principles of Distributed Systems*. Springer, 2006, pp. 396–411.

[10] ANGLUIN, D., ASPNES, J., EISENSTAT, D., AND RUPPERT, E. The computational power of population protocols. *Distributed Computing 20*, 4 (2007), 279–304.

[11] ANGLUIN, D., ASPNES, J., FISCHER, M. J., AND JIANG, H. Self-stabilizing population protocols. In *Principles of Distributed Systems*. Springer, 2006, pp. 103–117.

[12] ATTIYA, H., SNIR, M., AND WARMUTH, M. K. Computing on an anonymous ring. *Journal of the ACM 35*, 4 (1988), 845–875.

[13] BAR-YEHUDA, R., GOLDREICH, O., AND ITAI, A. On the time-complexity of broadcast in radio networks: an exponential gap between determinism randomization. In *Proceedings of the sixth annual ACM Symposium on Principles of Distributed Computing* (1987), ACM, pp. 98–108.

[14] BAR-YEHUDA, R., GOLDREICH, O., AND ITAI, A. Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection. In *Distributed Algorithms*. Springer, 1989, pp. 24–32.

[15] BERRY, G., AND BOUDOL, G. The chemical abstract machine. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (1989), ACM, pp. 81–94.

[16] BRAND, D., AND ZAFIROPULO, P. On communicating finite-state machines. *Journal of the ACM 30*, 2 (1983), 323–342.

[17] BRUSCHI, D., AND DEL PINTO, M. Lower bounds for the broadcast problem in mobile radio networks. *Distributed Computing 10*, 3 (1997), 129–135.

[18] CHALOPIN, J., DAS, S., AND SANTORO, N. Groupings and pairings in anonymous networks. In *Distributed Computing*. Springer, 2006, pp. 105–119.

[19] CHATZIGIANNAKIS, I., AND SPIRAKIS, P. G. The dynamics of probabilistic population protocols. In *Distributed Computing*. Springer, 2008, pp. 498–499.

[20] CHLAMTAC, I., AND KUTTEN, S. On broadcasting in radio networks–problem analysis and protocol design. *Communications, IEEE Transactions on 33*, 12 (1985), 1240–1246.

[21] CHLEBUS, B. S., GĄSIENIEC, L., GIBBONS, A., PELC, A., AND RYTTER, W. Deterministic broadcasting in unknown radio networks. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms* (2000), SODA '00, SIAM, pp. 861–870.

[22] CHLEBUS, B. S., KOWALSKI, D. R., AND PELC, A. Electing a leader in multi-hop radio networks. In *Principles of Distributed Systems*. Springer, 2012, pp. 106–120.

[23] CHROBAK, M., GASIENIEC, L., AND RYTTER, W. Fast broadcasting and gossiping in radio networks. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science* (2000), IEEE, pp. 575–581.

[24] CLEMENTI, A. E., MONTI, A., AND SILVESTRI, R. Distributed broadcast in radio networks of unknown topology. *Theoretical Computer Science 302*, 1 (2003), 337–364.

[25] CORNEJO, A., AND KUHN, F. Deploying wireless networks with beeps. In *Distributed Computing*. Springer, 2010, pp. 148–162.

[26] CZUMAJ, A., AND DAVIES, P. Brief announcement: Optimal leader election in multi-hop radio networks. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing* (2016), ACM, pp. 47–49.

[27] CZUMAJ, A., AND RYTTER, W. Broadcasting algorithms in radio networks with unknown topology. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science* (2003), IEEE, pp. 492–501.

[28] DE MARCO, G., AND PELC, A. Faster broadcasting in unknown radio networks. *Information Processing Letters 79*, 2 (2001), 53–56.

[29] EMEK, Y., SEIDEL, J., AND WATTENHOFER, R. Computability in anonymous networks: Revocable vs. irrecovable outputs. In *Automata, Languages, and Programming*. Springer, 2014, pp. 183–195.

[30] EMEK, Y., AND WATTENHOFER, R. Stone age distributed computing. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing* (2013), ACM, pp. 137–146.

[31] FÖRSTER, K.-T., SEIDEL, J., AND WATTENHOFER, R. Deterministic leader election in multi-hop beeping networks. In *Distributed Computing*. Springer, 2014, pp. 212–226.

[32] GHAFFARI, M., AND HAEUPLER, B. Near optimal leader election in multi-hop radio networks. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms* (2013), SIAM, pp. 748–766.

[33] GHAFFARI, M., HAEUPLER, B., AND KHABBAZIAN, M. Randomized broadcast in radio networks with collision detection. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing* (2013), ACM, pp. 325–334.

[34] GHAFFARI, M., LYNCH, N., AND NEWPORT, C. The cost of radio network broadcast for different models of unreliable links. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing* (2013), ACM, pp. 345–354.

[35] GILBERT, S., AND NEWPORT, C. The computational power of beeps. In *Distributed Computing*. Springer, 2015, pp. 31–46.

[36] GITMAN, I., VAN SLYKE, R., AND FRANK, H. Routing in packet-switching broadcast radio networks. *Communications, IEEE Transactions on 24*, 8 (1976), 926–930.

[37] HELLA, L., JÄRVISALO, M., KUUSISTO, A., LAURINHARJU, J., LEMPIÄINEN, T., LUOSTO, K., SUOMELA, J., AND VIRTEMA, J. Weak models of distributed computing, with connections to modal logic. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing* (2012), ACM, pp. 185–194.

[38] ITAI, A., AND RODEH, M. Symmetry breaking in distributive networks. In *22nd Annual Symposium on Foundations of Computer Science* (1981), IEEE, pp. 150–158.

[39] JOHANSSON, Ö. Simple distributed Δ+1-coloring of graphs. *Information Processing Letters 70*, 5 (1999), 229–232.

[40] KOWALSKI, D., AND PELC, A. Broadcasting in undirected ad hoc radio networks. In *Proceedings of the Twenty-Second Annual Symposium on Principles of Distributed Computing* (2003), ACM, pp. 73–82.

[41] KOWALSKI, D. R., AND PELC, A. Leader election in ad hoc radio networks: A keen ear helps. In *Automata, Languages and Programming*. Springer, 2009, pp. 521–533.

[42] KUHN, F., LYNCH, N., NEWPORT, C., OSHMAN, R., AND RICHA, A. Broadcasting in unreliable radio networks. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (2010), ACM, pp. 336–345.

[43] KUSHILEVITZ, E., AND MANSOUR, Y. An Ω (D log (N/D)) lower bound for broadcast in radio networks. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing* (1993), ACM, pp. 65–74.

[44] LINIAL, N. Locality in distributed graph algorithms. *SIAM Journal on Computing 21*, 1 (1992), 193–201.

[45] LUBY, M. A simple parallel algorithm for the maximal independent set problem. vol. 15, SIAM, pp. 1036–1055.

[46] MATIAS, Y., AND AFEK, Y. Simple and efficient election algorithms for anonymous networks. In *Distributed Algorithms*. Springer, 1989, pp. 183–194.

[47] MÉTIVIER, Y., ROBSON, J., SAHEB-DJAHROMI, N., AND ZEMMARI, A. About randomised distributed graph colouring and graph partition algorithms. *Information and Computation 208* (2010), 1296–1304.

[48] MÉTIVIER, Y., ROBSON, J. M., SAHEB-DJAHROMI, N., AND ZEMMARI, A. An optimal bit complexity randomized distributed mis algorithm. *Distributed Computing 23*, 5-6 (2011), 331–340.

[49] MOSCIBRODA, T., AND WATTENHOFER, R. Maximal independent sets in radio networks. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing* (2005), ACM, pp. 148–157.

[50] MOSCIBRODA, T., AND WATTENHOFER, R. Coloring unstructured radio networks. *Distributed Computing 21*, 4 (2008), 271–284.

[51] NAOR, M., AND STOCKMEYER, L. What can be computed locally? *SIAM Journal on Computing 24*, 6 (1995), 1259–1277.

[52] OSTER, G., AND PERELSON, A. Chemical reaction networks. *IEEE Transactions on Circuits and Systems 21*, 6 (1974), 709–721.

[53] SCHNEIDER, J., AND WATTENHOFER, R. A log-star distributed maximal independent set algorithm for growth-bounded graphs. In *Proceedings of the Twenty-Seventh ACM Symposium on Principles of Distributed Computing* (2008), ACM, pp. 35–44.

[54] SCHNEIDER, J., AND WATTENHOFER, R. What is the use of collision detection (in wireless networks)? In *Distributed Computing*. Springer, 2010, pp. 133–147.

[55] SEGALL, A. Distributed network protocols. Tech. rep., DTIC Document, 1980.

[56] SOLOVEICHIK, D. Robust stochastic chemical reaction networks and bounded tau-leaping. *Journal of Computational Biology 16*, 3 (2009), 501–522.

[57] SPIRAKIS, P. G. Population protocols and related models. In *Theoretical Aspects of Distributed Computing in Sensor Networks*. Springer, 2011, pp. 109–159.

[58] SUOMELA, J. A course on deterministic distributed algorithms. 2014. online textbook. https://users.ics.aalto.fi/suomela/da/. Accessed: 2015-08-01.

[59] SUOMELA, J. Survey of local algorithms. *ACM Computing Surveys 45*, 2 (2013), 24.

[60] YAMASHITA, M., AND KAMEDA, T. Computing on an anonymous network. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing* (1988), ACM, pp. 117–130.

Appendix A

Graph theory concepts

A number of algorithms, concepts, models and problems in distributed computing are closely related to graph theory. The following is a summary of the main graph theory concepts and notations. Although there are differences in literature, they are defined in a conventional manner. A more extensive compilation can be found in [58].

A graph G is a tuple $G = (V, E)$ compound of a finite set of nodes or vertices $v \in V$ and a finite set of connections or edges between pairs of nodes $\{u, v\} \in E$. A connected graph is a graph from in which there is a path for every pair of nodes. A weakly connected graph is such that replacing all of its directed edges with undirected edges produces a connected (undirected) graph. Conventionally, the size of a graph is defined as the number of nodes, denoted $n = |V|$. The degree of a vertex $v \in V$ for graph G is the number of incident edges, denoted $deg_G(v) = |\{u \in V : \{u, v\} \in E\}|$. The graph degree Δ is the maximum degree of a graph. The *eccentricity* D of a graph is the largest distance between a source node and any other node in the network. A graph family is the set of possible graphs that share a pattern or common structure. Table A.1 provides a summary of typical graph families in the referenced literature. Typical graph functions include:

- Neighboring $N(v) : \{u : \{u, v\} \in E\}$
- Labeling $f : V \mapsto \{1...k\}$ for some integer k
- d-coloring $c(v) : v \mapsto \{1...d\}$, s.t. $v \in V$ and $c(v) \neq c(u)$ for $u \in N(v)$
- Homomorphism $f(G) : G \mapsto G'$ maps two graphs preserving connections. Concretely $f(V) : V \mapsto V'$, s.t. $\{u, v\} \in E \to \{u', v'\}$

In addition to the graph family, some problems definitions make use of direction and parity modifiers. A superscript is used to denote

Notation	**Graph family**
$\mathcal{G}$	Unrestricted graphs have arbitrary topology
$\mathcal{Q}$	Complete graphs, also known as cliques, have all-pairs connected
$\mathcal{T}$	Tree graphs are graphs with no cycles
$\mathcal{T}^p$	Pseudo-trees are graphs with exactly one cycle
$\mathcal{K}_d$	d-regular graphs have all nodes with degree d
$\mathcal{S}$	Cycle graphs are a 2-regular pseudo-tree
$\mathcal{T}^C$	Cyclic graphs are the complement of tree graphs
Δ-$\mathcal{BD}$	Bounded degree graphs, s.t. $\lvert N(v)\rvert \leq \Delta$ for any $v \in V$
$\mathcal{UD}$	Unit disk graphs
$\mathcal{BG}$	Bounded growth graphs
$\mathcal{BI}$	Bounded independence graphs
$\mathcal{P}$	Path graphs are connected acyclic graphs with $\Delta = 2$
$\mathcal{B}$	Bipartite graphs do not have a cycle of odd length

Table A.1: Graph families.

direction $\{D, U\}$, corresponding to directed and undirected. A subscript is used to denote parity $\{E, O\}$, corresponding to even and odd.

- $\mathcal{S}_O$: Cycle graphs of odd size.
- $\mathcal{S}^D$: Directed cycle graphs.
- $\mathcal{G}^U$: Arbitrary undirected graphs.
- $\mathcal{S}_O^D$: Directed cycle graphs of odd size.

Appendix B

Other related PP models

This appendix describes additional PP models presented in previous literature. These include the urn automata, pairing automata, probabilistic PP, one-way PP, switching PP, community protocol, mediated protocol, chemical abstract machine and chemical reaction network.

The **urn automata** model was introduced in [6] previous to PP models. This model is based on agent interactions governed by a separate controller FSM equipped with an *urn* and an input tape. The controller is solely for the purpose of comparison to classical automata theory. The urn is an unstructured storage composed of a multiset T^* of tokens t from an alphabet T. The input tape works as a stack, and it is provided with an input set of symbols σ from a separate alphabet Σ. The tape is delimited on the left and right by special symbols ℓ and r. Similarly to touring machines, there is an input tape head with operations $\{\ell, r, -, \top, \bot\}$ corresponding to moving left, moving right, staying and halting after accepting or rejecting the input. The controller has a finite state set Q. A transition relation $\Delta \subseteq Q \times \Sigma \times T^* \times Q \times T^* \times \{\ell, r, -, \top, \bot\}$ defines the possible transitions $(q, x, i) \to (q', x', i')$. A transition $(q, \sigma, t, q', t', o)$ from state q to state q' affects both the urn and input tape. The multiset of symbols in the urn x changes to $(x \setminus t) \cup t'$, such that t and t' are the multiset of tokens withdrawn and deposited respectively. The input type current symbol σ at position i changes depending on the operation o. This base model assumed $|t| = k$, $|t| = |t'|$, as well as deterministic transitions and uniform sampling of withdrawn tokens.

The **pairing automata** model [6] is a variation of an urn automata with urn of size $k = 2$. This model represents explicitly a networks of agents interacting in pairs. In each interaction, each agents updates its state as a function of the current state of both agents. Consequently, the controller has one single state. Each token in the urn corresponds to an agent state. Interactions are modeled as transitions that sample a pair of tokens and replace it with another pair.

The **one-way PP** model introduced in [9] is a hybrid of MP and PP models. It inherits pairwise interactions from PP models and messaging from MP models. A key distinction with the standard PP model is that only receiver agents obtain information from sender agents. Two variants of this model are *transmission* and *observation*, depending on whether sender agents can detect that an interaction happened. Other orthogonal variants are defined depending on the message delivery assumption, namely *immediate*, *delayed* and *queued* delivery. This distinction is motivated by facilitating the comparison to MP models, since immediate delivery would be a strong assumption. To support this distinction a "*messages in transit*" multiset is introduced and each transition $(p, q) \mapsto (p', q')$ is split into send and receive events. The send event changes an agent state from p to p' and adds p to messages in transit. The receive event removes p from messages in transit and changes the other agent state from q to q'. The delayed variant requires agents to be ready to receive messages at any time, which may cause overflow since agents might not have time to react to incoming messages. In the queued variant agents may enter into a state in which they refuse to receive messages, which is the complement of *receive-enabled* states $Q_R \subseteq Q$. Lastly in the observation variant all states are receive-enabled.

Three more PP models were presented by Spirakis et al. in [57], namely **Mediator PP**, *PAssively mobile LOgaritmic space Machines* (**PALOMA**) and **Community Protocol**. The motivation was to expand the capabilities of PP models while keeping weak assumptions about the network. The first one uses a global mediator storage, holding a small space for each arc in the interaction graph. In the PALOMA extension agents are equipped with memory capacity logarithmic in the population size. The third extension adds unique IDs to agents.

The **switching PP** model introduced by Chatzigiannakis et al. in [19] is a generalization of probabilistic PP. This model considers a complete graph of non-terminating agents, that from time to time "review" their state and switch (transition) with some probability. It defines the density $x_q(t) = n_q/n$ of a state $q \in Q$ as the ratio between the number of nodes in that state n_q over the network size n. The population state or configuration is a vector of state densities $\vec{x}(t) = (x_1(t), \ldots, x_k(t))$ for $k = 1, \ldots, Q$. The review time of an agent depends on the population configuration $\vec{x}(t)$ and its current state. The switching probability at time t from state q_i to state q_j is a function $p_{ij}(\vec{x}(t)) = (p_{i1}(\vec{x}), \ldots, p_{ik}(\vec{x}))$. Lastly, two variants of switching PP are *Markovian PP*, in which specifications are independent of the configuration, and *Linear Viral Protocols*, in which each reviewing agents draw a paring agent at random and adopt its state.

The **Chemical Abstract Machine** model defined in [15] was motivated by modeling parallel computation carried out by interacting molecules. It defines states as chemical solution multisets, where molecules interact according to a multiset of reaction rules $m_1, \ldots, m_k \to m'_1, \ldots, m'_l$. A "magical" mechanism stirs the solution, allowing possible contacts between molecules. Reactions are asynchronous and can occur in parallel. In addition to reaction rules, heating $p|q \rightharpoonup p, q$ and cooling $p|q \leftharpoondown p, q$ rules represent molecule rearrangements between interactions. As opposed to reaction rules, heating and cooling rules are reversible and do not change the state of a solution. Furthermore, this model considers solutions encapsulation thorough "membranes", which are sub-solutions that might communicate with their environment through "pores".

The **Chemical Reaction Network** (CRN) model defined in [52] is a network of chemical species and reaction rules. In a graph $G = (V, E)$ representation, the set of species S_i for $i = \{1, \ldots, N\}$ represent the $N = |V|$ nodes, and the set of reactions R_j for $j = \{1, \ldots, M\}$ represent the $M = |E|$ connections. Reactions are directed from reactant to product species. The network is represented as a pair $\hat{v} = [v^R, v^P]$ of matrixes V_{NM}. The elements v_{ij} of each of the two matrixes specify whether specie i participates in reaction j as reactant or product respectively. An alternative representation of CRN was presented in [56] replacing the two matrices with vectors $\vec{r}_j \in \mathbb{N}^N$ and $\vec{p}_j \in \mathbb{N}^N$ of reactants and products respectively. This notation defines a state vector $\vec{x} \in \mathbb{N}^N$ with the molecular counts of each specie, which are the preconditions for a reaction R_j to have enough reactants $x_i - r_{ij} \geq 0$. The transition of reaction R_j goes from $\vec{x}$ to $\vec{x} + \vec{v}_j$, where $\vec{v}_j \in \mathbb{Z}^N$ and $\vec{v}_j = \vec{p}_j - \vec{r}_j$. The *stochastic* CRN presented in [56] extends the CRN model with a stochastic probability of reactions. In this model, a propensity function $a_j(\vec{x})dt$ defines the probability a_j of a reaction R_j occurring any time after t.

Appendix C

Algorithms appendix

Algorithm C.1: MIS algorithms for networks of processors – Las Vegas algorithm from [2].

```
Initialize: $I \leftarrow \emptyset$; $G' = (V', E') \leftarrow G = (V, E)$
while $G' \neq \emptyset$ do
    select($V', E'$); $I \leftarrow I \cup I'$; $Y \leftarrow I' \setminus N(I')$
    $G' = (V', E') \leftarrow$ induce($G', V' \setminus Y$)
end
select($V', E'$)
    $I' \leftarrow \emptyset$
    for $v \in V'$ do /*in parallel*/
        if d$_{G'}(v) = 0$ then $I' \leftarrow I' \cup \{v\}$
        w.p. $1/d_{G'}(v)$ do $I' \leftarrow I' \cup \{v\}$
    end
    for $\{u, v\} \in E'$ do /*in parallel*/
        if $u \in I'$ and $v \in I'$ then
            w.p. d$_{G'}(v)/(d_{G'}(u) + d_{G'}(v))$ do $I' \leftarrow I' \setminus \{v\}$
            otherwise $I' \leftarrow I' \setminus \{u\}$
        end
    end
end
/*For Monte Carlo algorithm presented by Luby in [45] use inclusion probability
$1/(2d_{G'}(v))$ and as validation criteria choose node $w$ with
d$_{G'}(w) = \max\{d_{G'}(u), d_{G'}(v)\}$*/
/*$\mathcal{O}(\log^2 n)$ phases [2, 45]: It requires $\mathcal{O}(\log n)$ phases since $\Omega(|E|)$ of edges are
removed from $E'$ at each phase [2, 45]. The validation step requires $\mathcal{O}(\log n)$ steps.*/
```

Algorithm C.2: MIS the PN model – algorithm 2 in [48].

```
State set: $Q : \{C, I, M, S\}$
Initialize: q $\leftarrow C$; $ActiveSet \leftarrow \{u \in N(v) : q(u) \notin M \cup S\}$
while q $\notin M \cup S$ and $ActiveSet \neq \emptyset$ do
    $b \sim \{0,1\}$; broadcast($b$); $((m)) \leftarrow$listen()
    if $b = 1$ and $0 \in ((m))$ then q $\leftarrow I$
    if $b = 0$ and $0 \notin ((m))$ then q $\leftarrow M$
    broadcast(q); listen()
    if received MIS then q $\leftarrow S$
    else $ActiveSet \leftarrow ActiveSet \setminus \{u \in N(v) : q(u) \notin M \cup S \cup I\}$
end
/*$\mathcal{O}(\log n)$ phases [48]: active edges decrease by a half at each phase*/
```

Algorithm C.3: Coloring with arbitrary number of colors in the PN [47].

```
State set: $Q : \{A, W, P\}$
State variables: $c_v$; /*chosen color*/ $colors_v$; /*color palette*/
$active_v$, $IN_v$, $OUT_v$ /*neighbor sets*/
Initialize: q $\leftarrow A$; $c_v \leftarrow \varepsilon$; /*uncolored*/
$active_v \leftarrow N_v$; /*all neighbors are active*/ $IN_v \leftarrow \emptyset$; $OUT_v \leftarrow \emptyset$;
/*Part 1*/
while $active_v \neq \emptyset$ do
    $b \sim \{0,1\}$
    for $u \in active_v$ do
        $b_u \leftarrow$ exchange($b$)
        if $b_u \neq b_v$ then
            if $b_u = 0$ then $OUT_v \leftarrow OUT_v \cup \{u\}$
            else $b_u = 1$
            $IN_v \leftarrow IN_v \cup \{u\}$
        end
    end
end
q $\leftarrow W$; $color_v \leftarrow \{1, \ldots, |IN_v| + |OUT_v|\}$ /*degree of v*/
/*Part 2*/
while $IN_v \neq \emptyset$ do
    $(c_w, w) \leftarrow$ receive(); $IN_v \leftarrow IN_v \setminus \{w\}$; $colors_v \leftarrow colors_v\{c_w\}$
end
$c_v \leftarrow \min\{colors_v\}$; q $\leftarrow C$
for $u \in OUT_v$ do send($c_v, v$)
/*$\mathcal{O}(\log n)$ rounds w.h.p. [47]: For part 1 after $4 \log n - 1$ rounds a number $\leq 2/n^2$ of
edges remain. For part 2 the longest path is bounded by a constant and the size of the
message $\mathcal{O}(\log n)$ dominates the run time. */
```

Algorithm C.4: LE in the PN – algorithm ELECT in [46].

```
**Constants**: $L$ /*lower bound on the network size*/; $r = 1/\varepsilon$ /*inverse of error*/
**State set**: $Q : \{A_l\}$, such that $l$ is the largest received ID (current leader at this node)
**State variables**: ID$_i$; /*chosen ID*/ ID$_l$; /*largest received ID*/
parent$_i$ /*pointer to the parent node*/
**Initialize**: ID$_i \leftarrow$ choose(); /*ID of length $\mathcal{O}(r \log r)$*/ ID$_l \leftarrow$ ID$_i$; parent$_i \leftarrow 0$
broadcast(ID$_i$)
while true do
    ID$_u \leftarrow$ listen()
    if ID$_u >$ ID$_l$ then parent$_i \leftarrow u$ ID$_l \leftarrow$ ID$_u$
    broadcast(ID$_l$)
end
/*$\mathcal{O}(D)$ rounds w.h.p. [46]: It takes $D$ rounds for the largest ID to reach all nodes*/
```

Algorithm C.5: BC in silent RN – decay approach [13].

```
**Constants**: $t_C = 2\lceil \log \Delta \rceil$; $t_P = \log(N/\varepsilon)$ /*timeouts*/
**State set**: $Q : \{W, A, C, P\}$
**Parameters:** $N$ /*network size upper bound*/; $\Delta$ /*maximum degree*/
**Initialize**: q $\leftarrow W$; $t \leftarrow 0$ /*time*/
repeat m $\leftarrow$ listen() until m $\neq \varepsilon$
q $\leftarrow A$
for $i \in \{1, \ldots, t_P\}$ do
    repeat $t \leftarrow t + 1$ until $t \mod t_C \equiv 0$
    q $\leftarrow C$; decay($t_C, m$); q $\leftarrow A$;
end
q $\leftarrow P$; terminate()
decay($k, m$)
    $j \leftarrow 0$
    repeat
        broadcast($m$); $coin \sim \{0, 1\}$; $j \leftarrow j + 1$
    until $j = k$ or $coin = 0$
end
/*$\mathcal{O}((D + \log n/\varepsilon) \cdot \log n) = \mathcal{O}(D \log n + \log^2 n)$ phases w.p. $1 - \varepsilon$ [13]: Each node
requires $\log(N/\varepsilon)$ phases to broadcast successfully w.p. $\geq 1 - \varepsilon$. It takes $\mathcal{O}(D)$
iterations for the message to propagate, since decay takes $k = 2 \log \Delta$.*/
```

Algorithm C.6: Randomized BC in silent RN – selective sequences [27].

Constants: $\lambda = \log(n/D)$

$$\alpha'_k = \begin{cases} \frac{1}{2\lambda} & \text{for } 1 \leqslant k \leqslant \lambda, \\ \frac{1}{2\lambda} \cdot 2^{-(k-\lambda)} & \text{for } \lambda < k \leqslant \lambda + \lceil \log\log n \rceil, k \leqslant \log n, \\ \frac{1}{2\lambda} \cdot \frac{1}{\log n} & \text{for } \lambda \lceil \log\log n \rceil < k \leqslant \log n, \\ 1 - \sum_{i=1}^{\log n} \alpha'_i & \text{for } k = 0 \end{cases} \tag{C.1}$$

```
State set: $Q : \{A, P\}$
Input: $\mathcal{J} = \langle J_1, J_2, \ldots, \rangle$ s.t. Pr$[J_r = k] = \alpha'_k$ for $r \in \mathbb{N}$, $k \in \{1, 2, \ldots, \log n\}$
Initialize: q $\leftarrow P$
for $r \in \{1, \ldots, T\}$ do /*round number*/
    if q $= A$ then
        w.p. $2^{-J_r}$ do transmit(m)
    else
        m $\leftarrow$ listen()
        if m $\neq \epsilon$ then q $\leftarrow A$
    end
end
/*$\mathcal{O}(cD \log n/D)$ rounds w.h.p. [27]: It requires $\mathcal{O}(\log n/D)$ rounds for each layer,
from up to $D$ disjoint layers.*/
```

Algorithm C.7: BC in silent RN – universal probability sequences [40].

```
Constants: $t_C = 32 \cdot N^{2/3}$; $t_P = \log N$; /*timeouts*/
State set: $Q : \{P, A_d, C_d, A_o, C_o\}$ /*$d$ is decay and $o$ is optimized BC*/
Initialize: q $\leftarrow P$
for $i \in \{1, \ldots, t_P\}$ do randomizedBroadcasting($2^i$)
q $\leftarrow P$; terminate()
randomizedBroadcasting($D$)
    if $D \leq t_C$ then decayBroadcasting($D, N$)
    else
        for $i \in \{1, \ldots, 4600 \cdot D\}$ do
            if q $= A_o$ then q $\leftarrow C_o$; stage($D, i$); q $\leftarrow A_o$
            else
                m $\leftarrow$ listen();
                if m $\neq \varepsilon$ then q $\leftarrow A_o$
            end
        end
    end
end
/*It continues in Algorithm C.8*/
```

Algorithm C.8: BC in silent RN – stage subroutine [40].

```
stage($D, i$)
    for $l \in \{0, \ldots, \log(N/D)\}$ do
        w.p. $1/2^{\ell}$ do transmit()
    end
    w.p. $p_i$ do transmit()
end
```

/*$\mathcal{O}((D \cdot \log n/D) + \log^2 n)$ rounds w.h.p. [40]: The probability of a node not receiving the message after D stages is $1/N^2$. The probability of some node not receiving the message is $r \cdot (1/N^2) = 1/N$*/

Algorithm C.9: Deterministic BC in silent RN – selective families [21].

```
Constants: $r = \lceil n/6 \rceil$; /*deterministic sample size*/;
$k_n = 2^{\lceil n/6 \rceil}$; /*selective factor*/
$f_n = |\mathcal{F}_n|$ /*size of selective family*/ $t_P = \lceil 2^n/k_n \rceil$ /*iterations*/
State set: $Q : \{A, P\}$
Input: $\ell$ /*node label or ID*/
Initialize: m $= \epsilon$ /*empty message*/
for $i \in \{1, \ldots, t_P\}$ do q $\leftarrow A$; segment($n$) q $\leftarrow P$;
segment($n$)
    for $i \in \{1, \ldots, k_n\}$ do
        for $j \in \{1, \ldots, f_n\}$ do
            if $\ell \in F_j$ and m $\neq \epsilon$ then transmit(m)
            m $\leftarrow$ listen()
        end
    end
    for $i \in \{1, \ldots, \ell\}$ do m $\leftarrow$ listen()
    transmit(m)
end
```

/*$\mathcal{O}(n^{11/6})$ rounds w.h.p. [21]: It is based on the size of (n, k_n)-selective families $\mathcal{O}(2^{5n/6})$. It iterates over $\lceil 2^n/k_n \rceil$ phases of $k_n \cdot f_n + 2^n$.*/

Algorithm C.10: MIS in the nFSM model – section 4.1 in [30].

```
State set: $Q : \{A, C, M, S, W\}$
State variables: $t \in \{0,1,2\}$ /* pulse for candidate state */; $b \in \{0,1\}$ /* coin toss*/
Initialize: q $\leftarrow A$
while q $\notin M \cup S$ do /*Double parenthesis denotes multi set of messages*/
    broadcast(q); $((\text{q})) \leftarrow$ listen()
    case q $= A$ : if $A \in ((\text{q}))$ then q $\leftarrow C_0$
    case q $= C$ :
        $b \sim \{0,1\}$
        if $b = 1$ and $C_{t-1} \in ((\text{q}))$ then $t \leftarrow t+1 \mod 3$
        else
            if $C_t \in ((\text{q}))$ or $C_{(t+1 \mod 3)} \in ((\text{q}))$ then q $\leftarrow M$ else q $\leftarrow W$
        end
    end
    case q $= W$ : if $M \in ((\text{q}))$ then q $\leftarrow A$ else q $\leftarrow S$
end
/*$\mathcal{O}(\log^2 n)$ w.h.p. [30]: Active nodes decrease by $\mathcal{O}(\log n) + NB(\mathcal{O}(\log n), 1-p)$,
since the competing rounds for a node $c_u \sim Geom(1/2)$. For all nodes is $\leq \mathcal{O}(\log n)$*/
```

Algorithm C.11: MIS in *harsh* RN – algorithm 1 in [49].

```
Constants: $\mu = 16$; /*number of small disks between $1$-hop an $2$-hop neighborhood*/
$\alpha = 6.4$; /*value that maximizes clearances (no interference in the neighborhood)*/
$\lambda = 3 \cdot 2^{\alpha+2} 4^{\frac{9/4+3\mu}{2^\alpha}}$; $\delta = \frac{8 \cdot 2^\alpha}{\tau}$; $\tau = 9000^{-1}$; /*values to achieve high probability */
$p_C = \tau/(2^\alpha \log n)$; $p_M = 2^{-\alpha}$; /*probabilities*/
$t_W = 4\mu\delta \log^2\ n$; $t_A = \lambda \log n$; $t_C = \delta \log n$; $t_M = \delta\ \log^2\ n$ /*timeouts*/
State set: $Q : \{A, C, M, S, W\}$
Initialize: $s_v \leftarrow 0$; $c_v \leftarrow 0$; q $\leftarrow W$; $p_{A_v} \leftarrow 2^{-\alpha-2}/n$
while true do
    for $s_v = 0, \dots, t_W$ do processMessage(listen())
    q $\leftarrow A$
    repeat
        $p_{A_v} \leftarrow 2 \cdot p_{A_v}$; for $s_v = 0, \dots, t_A$ do processMessage(listen())
    until $p_{A_v}$
    broadcast($m_A$); q $\leftarrow C$
    for $c_v = c_v, \dots, t_C$ do
        processMessage(listen())
        w.p. $p_C$ do $c_v \leftarrow \max\{c_v, t_C + 1\}$
        broadcast($m_C(c_v)$)
    end
    q $\leftarrow M$; w.p. $p_M$ do broadcast($m_M$)
end
processMessage($m$)
    case m $= m_A$: : q $\leftarrow A$; $s_v =\leftarrow 0$
    case m $= m_C(c_u)$: : if q $= C$ and $|c_v - c_u| \leq t_C$ then $c_v = 0$
    case m $= m_M$: : q $\leftarrow S$; terminate()
end
```

Algorithm C.12: Deterministic MIS in BE – algorithm 4.2 in [54].

```
Constants: $r = 2$ /*s.t. size of largest independent set in $r$ radious $|N_r(v)| \leq f(r)$
           for polynomial function $f$*/
State set: $Q : \{A, C, I, M, S\}$
Input: ID /*node label or ID*/
Initialize: q $\leftarrow A$
for $\ell \in \{1, \dots, f(f(2) + 2)\}$ do
    for $i \in \{0, \dots, f(2)\}$ do
        $r_v^0 \leftarrow ID_v$
        for $j \in \{1, \dots, \log^* n + 2\}$ do
            $r_j^0 \leftarrow \log^{(j)} n$
            for $k \in \{0, \dots, \log^{(j)} n\}$ do
                if q $= A$ then
                    if $r_v^{j-1}[k] = 1$ and $r_v^j = \log^{(j)} n$ then beep()
                    else
                        m $\leftarrow$ listen()
                        if m $\neq \varepsilon$ and $r_v^j = \log^{(j)} n$ then $r_v^j \leftarrow k$ end
                    end
                end
            end
            updateState(q, $j$, $r^j$)
        end
        if q $= C$ then q $\leftarrow A$
    end
    if q $= I$ then q $\leftarrow A$
end
updateState(q, $j$, $r^j$)
    if q $= A$ and $r_v^j = \log^{(j)} n$ then
        if $j = 1$ then q $\leftarrow M$; listen(); beep();
        else q $\leftarrow C$; beep(); listen();
    else
        $m_0 \leftarrow$ listen(); if $m_0$ = beep and q $= A$ then q $\leftarrow I$ end
        $m_1 \leftarrow$ listen(); if $m_1$ = beep then q $\leftarrow S$ end
    end
end
/*$\mathcal{O}(\log n)$ [54]*/
```

Algorithm C.13: MIS in BE with known N – algorithm 1 in [1].

```
Constants: $p_M = 1/2$; /*probabilities*/ $t_W = c\log^2 n$; $t_C = c\log n$; $t_M = N$
            /*timeouts*/
State set: $Q : \{C, M, W\}$
Initialize: q $\leftarrow W$
wait($t_W$); q $\leftarrow C$
for $i \in \{1, \dots, t_M\}$ do
    for $j \in \{1, \dots, t_C\}$ do
        $p_C \leftarrow 2^i/(8 \cdot N)$; w.p. $p_C$ do beep()
        listen()
    end
end
q $\leftarrow M$
forever do
    w.p. $p_M$ do beep(); listen()
    otherwise listen(); beep()
end
```

/*$\mathcal{O}(\log^2 N \log n)$ rounds [1]: At each round some neighbor beeps w.p. $\leq 1/2$, a node beeps alone every $\mathcal{O}(\log^2 N)$ rounds w.p. $1/e$, and $\mathcal{O}(\log n)$ of these events are required.*/

Algorithm C.14: MIS in BE with wake on beep and sender side CD – algorithm 2 in [1].

```
Constants: $t_W = 1$
State set: $Q : \{A, C, M, S, W\}$
Initialize: q $\leftarrow W$
upon wake up do beep()/*after woken up by adversary or beep, wake up others*/
wait($t_W$); $x \leftarrow 0$
while q $\notin M \cup S$ do
    $x \leftarrow x + 1$; q $\leftarrow A$
    for $i \in \{1, \dots, N\}$ do
        /*1st exchange*/
        m$^1$ $\leftarrow$ listen(); $p_C \leftarrow 1/2^i$; w.p. $p_C$ do q $\leftarrow C$; beep()
        m$^2$ $\leftarrow$ listen(); if m$^1$ = beep or m$^2$ = beep then q $\leftarrow A$
        /*2nd exchange*/
        m$^3$ $\leftarrow$ listen(); if q = $C$ then q $\leftarrow M$; beep()
        m$^4$ $\leftarrow$ listen(); if m$^3$ = beep or m$^4$ = beep then q $\leftarrow S$
    end
end
```

/*$\mathcal{O}(\log^2 n)$ rounds [1]: Only $\log n$ rounds are required after x reaches $\log n$. At each phase a constant number of edges $\Omega(|E|)$ is deleted.*/

Algorithm C.15: MIS in BE with wake on beep, without sender side CD – algorithm 3 in [1].

```
**Constants**: $t_W = 1$
**State set**: $Q : \{A, C, I, M, W\}$
**Initialize**: q $\leftarrow W$
**upon** *wake up /*by adversary or beep*/* **do** beep() /*wake up neighbors*/
wait($t_W$); $x \leftarrow 0$
**while** true **do**
    $x \leftarrow x + 1$; q $\leftarrow A$
    **for** $i \in \{1, \ldots, N\}$ **do**
        **if** q $= I$ **then** $p_A \leftarrow 1/2^i$ **w.p.** $p_A$ **do** q $\leftarrow A$
        **for** $j \in \{1, \ldots, c \cdot x\}$ **do** $X_j \sim \{0, 1\}$
        $k \sim \{1, \ldots, c \cdot x\}$; $X_k \leftarrow 1$ /*ensure at least one bit is 1*/
        **for** $j \in \{1, \ldots, c \cdot x\}$ **do**
            challenge()
            **if** q $\neq I$ **and** $X_j = 1$ **then** challenge() **else** beep()
            challenge()
        **end**
    **end**
**end**
challenge()
    listen(); **if** $m_{\texttt{beep()}}$ **then** q $\leftarrow I$
**end**
/*$\mathcal{O}(\log^3 n)$ rounds [1]: It is similar to wake on beep with sender side CD. It takes an additional $\mathcal{O}(\log n)$ to eliminate conflicts.*/
```

Algorithm C.16: MIS in the synchronous BE model – algorithm 4 in [1].

```
State set: $Q : \{I, C, M\}$
Initialize: q $\leftarrow I$; $next \sim \{0,1\}$
while true do
    case $t \mod 3 = 0$ :
        if $t \mod k \neq 0$ then beep()
        else
            m $\leftarrow$ listen()
            if m $=$ beep then q $\leftarrow I$; $k \leftarrow 2 \cdot k$
            else
                if q $= I$ then q $\leftarrow C$
                if q $= C$ then q $\leftarrow M$
            end
        end
    end
    case $t \mod 3 = 1$ :
        if q $= M$ then beep()
        else
            m $\leftarrow$ listen()
            if m $=$ beep then q $\leftarrow I$
        end
    end
    case $t \mod 3 = 2$ :
        if q $= I$ then m $\leftarrow$ listen()
        else if q $= C$ then
            w.p. $1/2$ do beep()
            otherwise m $\leftarrow$ listen()
            if m $=$ beep then q $\leftarrow I$
        end
        else if q $= M$ then
            if $next = 1$ then beep(); $next \sim \{0,1\}$
            else
                m $\leftarrow$ listen(); $next \leftarrow 1$
                if m $=$ beep then q $\leftarrow I$; $k \leftarrow 2 \cdot k$
            end
        end
    end
end
```

/*$\mathcal{O}(\log^2 n)$ rounds [1]: For any given k it takes $\mathcal{O}(k \log n)$ rounds for the estimate to synchronize in local $\mathcal{O}(\log n)$ neighborhoods. During this time the communication is collision free and nodes can execute Luby's algorithm. Since there is no additional overhead, the runtime of this algorithm a multiplicative factor of Luby's algorithm runtime, in total $\mathcal{O}(\log^2 n)$.*/

Algorithm C.17: Optimal LE for directed graphs in RN – algorithms 2,3 and 7 in [26].

```
Constants: $p_C = (4\log n)/n$ /*probabilities*/;
$\ell \leftarrow \log n$ /*ID length*/;
$t_E \leftarrow 2\sqrt{\log n}$ /*Elimination timeout*/
State set: $Q : \{A, C, S, W\}$ /*active,candidate, eliminated, witness */
Input: $n$ /*network size*/; $D$ /*network diameter*/;
State variables: ID; $b$ /*unique leader flag*/; $\mathsf{ID}_c$ /*Highest ID*/
while true do
    q $\leftarrow A$
    w.p. $p_C$ do q $\leftarrow C$
    if q $= C$ then ID $\sim \{0,1\}^\ell$
    m $\leftarrow$ search(q, ID, $\ell$) if m$[1,\dots,\ell] \neq$ ID$[1,\dots,\ell]$ then q $\leftarrow A$
    for $t_E$ do
        $(\mathsf{ID}_c, b) \leftarrow$ selection(q, ID)
        if $b = 1$ then Output: $\mathsf{ID}_c$
        ; terminate()
        else if $\mathsf{ID}_c >$ ID then q $\leftarrow S$
    end
end
search(q, ID, $\ell$)
    $\mathsf{ID}_c \leftarrow 0^\ell$
    for $i \in \{1,\dots,\ell\}$ do
        if ID$[j] = \mathsf{ID}_c[j]$ for $j < i$ and $\mathsf{ID}_i = 1$ then m $\leftarrow$ multiBC(q, 1, 1)
        else m $\leftarrow$ multiBC($\varepsilon$, 1, 1)
        if m $= \varepsilon$ then $\mathsf{ID}_c[i] \leftarrow 0$
        else $\mathsf{ID}_c[i] \leftarrow 1$
    end
    return($\mathsf{ID}_c$)
end
selection(q, ID, $\ell$)
    m $\leftarrow$ multiBC(q, ID, $16\log n$)
    if m $\neq \varepsilon$ then
        for $i \in \{1,\dots,16\log n\}$ do
            if m$[i] = 1$ then for $j \in \{1,\dots,4\}$ do $\mathsf{ID}_c \leftarrow$ decay(ID)
            if m$[i] = 0$ and $\mathsf{ID}_c[i] = 1$ then q $\leftarrow W$;m $\leftarrow \max$(m, $\mathsf{ID}_c$)/*Witness*/
        end
        $\mathsf{ID}_c \leftarrow$ multiBC($W$, m, $16\log n$)
        if $\mathsf{ID}_c = \varepsilon$ then return(m, 1)
        else return($\mathsf{ID}_c$, 0)
    end
    return(m, 0)
end
/*$\mathcal{O}(D\log n/D + \log^2 n)$ [26]: Search takes $T_{BC} \cdot \sqrt{\log n}$, and performing $2 \cdot \sqrt{\log n}$
iterations of slection takes $T_{BC} \cdot \sqrt{\log n}$.*/
```

Algorithm C.18: Optimal LE for undirected graphs in synchronous clocks BE – algorithms 4 and 8 in [26].

```
Constants: $p_C = 1/n$ /*probabilities*/; $\ell \leftarrow 4\log n$ /*ID length*/;
State set: $Q : \{A, C, S, W\}$ /*active,candidate, eliminated, witness */
Input: $n$ /*network size*/; $D$ /*network diameter*/;
State variables: ID; m; /*Received message from beepWave()*/ ID$_c$ /*Highest ID
                 substring*/
while true do
    q $\leftarrow A$
    w.p. $p_C$ do q $\leftarrow C$
    if q $= C$ then ID $\sim \{0,1\}^{4\ell}$, s.t. ID has exactly $\log n$ 1s
    m $\leftarrow$ beepWave(q, ID, $\ell$)
    if m $\neq \varepsilon$ then
        if m *contains more than* $\log n$ *1s* then
            q $\leftarrow W$; ID$_c \leftarrow$ beepWave(q, $1, 1$)
            if $ID_c = \varepsilon$ then Output: m
            ; terminate()
        end
    end
end
beepWave(q, ID, $\ell$)
    ID$_c \leftarrow 0^\ell$
    if q $= C$ then
        beep() /*time 0*/
        for $i \in \{1, \ldots, \ell\}$ do
            listen(); listen() /*Listen 2 times*/
            if $ID[i] = 1$ then beep(); ID$_c[i] = 1$
            else listen(); ID$_c[i] = 0$
        end
    end
    else
        $j \leftarrow 0$
        repeat
            m$_1 \leftarrow$ listen();$j \leftarrow j + 1$
        until m$_1$ = beep *or* $j = D \cdot \ell$
        beep()
        for $i \in \{1, \ldots, \ell\}$ do
            m$_2 \leftarrow$ listen(); m$_3 \leftarrow$ listen();
            /*Received beep at step $j + 3i$*/
            if $Message_3$ = beep then beep(); ID$_c[i] = 1$
            else m$_1 \leftarrow$ listen(); ID$_c[i] = 0$
        end
    end
end
/*$\mathcal{O}(D \log n/D + \log^2 n) \cdot \sqrt{\log n}$ for BE, assuming known $n$ [26] */
```

Algorithm C.19: Clustering based LE in RN – template of [32].

```
Constants: $p_C = (10 \log n)/n$; /*probabilities*/
State set: $Q : \{A, C, L, U, B, S, P\}$ /*active, candidate, clustered, unclustered,
           boundary, eliminated, passive*/
Input: $n$ /*network size*/; $D$ /*network diameter*/; L /*codeword length*/
State variables: ID /*generated node ID*/; $d$ /*distance to closest candidate*/; $t$
                 /*time*/; $c$ /*Cluster ID*/
Initialize: ID $\sim \Theta(\log n)$ /*draw ID u.a.r.*/
w.p. $p_C$ do q $\leftarrow C$
otherwise q $\leftarrow S$
for $i \in \theta(\log \log n)$ do
    if debate(ID, $D$) then break()
end
if q $= C$ then ID$_\ell \leftarrow$ ID; broadcast(ID)
else ID$_\ell \leftarrow$ listen()
Output: ID$_\ell$
q $\leftarrow P$
debate(ID, $D$)
    $H \leftarrow$ cluster(); /*Build overlay graph*/
    exchange(ID, $D$)
    $\delta =$ calculateDegree(); exchange(($\delta$, ID), $D$)
    if q $= C$ and $(\delta, \text{ID}) > \max\{(\delta_v, \text{ID}_v) : v \in N_H(u)\}$ then q $\leftarrow S$
end
exchange(m, $D$)
    /*Uplink and downlink use network diameter*/
    uplink(m, $D$); intercommunicate(m); downlink(m, $D$)
end
/*$\mathcal{O}(D \log n/D + \log^3 n) \cdot \min\{\log \log n, \log n/D\}$ for RN without CD and
$\mathcal{O}(D + \log n \log \log n) \cdot \{\log \log n, \log n/D\}$ for BE [32]*/
```

Algorithm C.20: Clustering an numbering in BE – algorithm 4 and 5 from [32].

```
numbering($D$)
m $\leftarrow$ Encode(superimposedCode(1), bitXor(0,ID)) /*SI(1)-code*/
uplink(m)
m$'$ $\leftarrow$ listen()
if m $\in$ $valid$ then $c$ $\leftarrow$ decode(m'); q $\leftarrow$ $L$
else $c \leftarrow 0$; q $\leftarrow$ $U$
for $t \in \{0,\dots,L-1\}$ do
    if m$'[t] = 1$ then beep(); m $\leftarrow$ listen()
    else m $\leftarrow$ listen(); beep()
    if m = beep then q $\leftarrow$ $B$
end
numbering($D$)
    if q = $C$ then $d \leftarrow 0$
    for $t \in \{1,\dots,D\}$ do
        if q $\in \{A, C\}$ then beep
        else
            m $\leftarrow$ listen();
            if m = beep and q = $L$ then q $\leftarrow$ $A$; $d \leftarrow t$
        end
    end
end
```

Algorithm C.21: Uplink BE – algorithm 6 from [32].

```
uplink(m, $D$)
    if q $\neq C$ then q $\leftarrow P$
    for $t \in \{0, \ldots, D + 3L - 3\}$ do
        switch $t - d \mod 3$ do
            case $0$ : if q $= C$ and m$[t/3] = 1$ then beep()
            m $\leftarrow$ listen()
            case $1$ : m $\leftarrow$ listen()
            case $2$ :
                m $\leftarrow$ listen();
                if m $=$ beep then
                    m$'[\lfloor (t - d + 1)/3 \rfloor] \leftarrow 1$;
                    if q $\neq C$ then q $\leftarrow A$
                end
                else
                    m$'[\lfloor (t - d + 1)/3 \rfloor] \leftarrow 1$;
                    if q $\neq C$ then q $\leftarrow P$
                end
            end
        end
    end
end
```

Algorithm C.22: Intercommunication in BE – algorithm 7 from [32].

```
intercommunicate(m)
    for $t \in \{0, \dots, L-1\}$ do
        if $\mathsf{q} = B$ then
            if $\mathsf{m}[t] = 1$ then beep(); beep(); $\mathsf{m}'''[t] \leftarrow 1$
            else
                $\mathsf{m}_0 \leftarrow$ listen();
                $\mathsf{m}_1 \leftarrow$ listen()
                $\mathsf{m}'''[t] \leftarrow (\mathsf{m}_0 = \mathsf{beep}) or (\mathsf{m}_1 = \mathsf{beep})$
            end
        end
        else
            $\mathsf{m} \leftarrow$ listen();
            if $\mathsf{m} = \mathsf{beep}$ then beep()
            listen()
        end
    end
end
```

Algorithm C.23: Downlink in BE – algorithm 8 from [32].

```
downlink(m, $D$)
    if q $\neq C$ then q $\leftarrow P$
    for $t \in \{D + 3L - 3, \dots, 0\}$ do
        switch $t - d \mod 3$ do
            case $0$ :
                if q $= B$ and $t - d \in [0, 3(L-1)]$ and m$[\lfloor (t-d)/3 \rfloor] = 1$ then
                beep()
                else m $\leftarrow$ listen()
            end
            case $1$ : m $\leftarrow$ listen()
            case $2$ :
                m $\leftarrow$ listen();
                if q $\neq C$ then
                    if $t \in [1, 3(L-1)+1]$ then
                        if m = beep then m$'[(t-d-1)/3] \leftarrow 1$
                        else m$'[(t-d-1)/3] \leftarrow 0$
                    end
                end
                else
                    if m = beep then q $\leftarrow A$
                    else q $\leftarrow P$
                end
            end
        end
    end
end
```

Algorithm C.24: Max detection in BE – algorithm 9 from [32].

```
maxDetection(m)
    if q $= B$ then
        for $t \in \{0, \ldots, L-1\}$ do
            if q $\neq S$ then /*Not marked*/
                if m$[t] = 1$ then beep(); beep();
                else
                    m$_0 \leftarrow$ listen()
                    m$_1 \leftarrow$ listen()
                    if m$_0 =$ beep or m$_1 =$ beep then q $\leftarrow S$
                end
            end
            else
                m $\leftarrow$ listen()
                if m $=$ beep then beep()
                listen()
            end
        end
end
```

Algorithm C.25: $(\Delta + 1)$-coloring undirected trees in the nFSM model [30].

```
State set: $Q : \{W, A, C\}$
Input: $b = 3$ /*model bounding parameter*/
Initialize: q $\leftarrow A$; c$_v \leftarrow \varepsilon$; /*uncolored */;
$C(v) \leftarrow \{1, \ldots, b\}$ /*palette of available colors*/
while q $\neq C$ do
    broadcast(q); ((q)) $\leftarrow$ listen() /*Discover*/
    d$_v \leftarrow$ calculateDegree(((q)));
    broadcast(d$_v$); ((d)) $\leftarrow$ listen() /*Exchange degrees*/
    if $\max$ ((d)) $\leq$ d$_v < b$ then
        randColor()
    else
        listen(); /*Just listen while others do randColor()*/
        if d$_v = 1$ then q $\leftarrow W$
    end
    ((c)) $\leftarrow$ listen(); $C_v \leftarrow C_v \setminus$ ((c)) /*Update palette*/
    if q $= W$ and ((c)) $\neq \varepsilon$ then q $\leftarrow A$
end
randColor()
    c$_v \sim C(v)$; broadcast(c$_v$); /*Propose color*/
    ((c)) $\leftarrow$ listen()
    if c$_v \notin$ ((c)) then q $\leftarrow C$; broadcast(c$_v$) /*Reserve color*/
end
/*$\mathcal{O}(\log n)$ rounds w.h.p. [30]: */
```

Algorithm C.26: $\mathcal{O}(\Delta)$-coloring $\mathcal{GB}$ graphs in the harsh RN [50].

```
Constants: $\sigma = 10e^2\kappa_2/((1-1/\kappa_2)(1-1/(\kappa_2\Delta)))$;
$\gamma = 5\kappa_2/((1/e(1-1/\kappa_2))^{\kappa_1/\kappa_2}(1/e(1-(\kappa_2\Delta)))^{1/\kappa_2})$; /*bound multipliers*/
$\alpha \geq 2\gamma\kappa_2 + \sigma + 1$; $\beta \geq \gamma$ /*timeouts*/
State set: $Q : \{R, A_i, C_i\}$, where $i$ is the stage number
Input: $\Delta$ /*network degree*/
State variables: $P_v$; /*competitor list*/ $d_v$; /*local copy of neighbor's
counters*/ $i$ /*phase number*/; $L_v$ /*leader ID*/
$A_{succ}$ /*next state*/; $\zeta$ /*degree*/; $\mathcal{Q}$ /*queue of requesters*/
Initialize: $P_v \leftarrow \emptyset$; $d_v(w) \leftarrow 0$ s.t. $w \in N(v)$; $i \leftarrow 0$; q $\leftarrow A_0$; $\zeta \leftarrow 1$; $A_{succ} \leftarrow R$
while q $\neq C$ do
    compete() if q $= C$ then playLeader()
    $\zeta \leftarrow \Delta$; $A_{succ} \leftarrow A_{i+1}$
end
compete()
    for $j \in \lceil \alpha\Delta \log n \rceil$ do
        for $w \in P_v$ do $d_v(w) \leftarrow d_v(w) + 1$
        if $m^i_A(w, c_w)$ received then $P_v \leftarrow P_v \cup \{w\}$; $d_v(w) \leftarrow c_w$
        if $m^i_C(w)$ received then
            q $\leftarrow A_{succ}$; $L_v \leftarrow w$;
            if q $= R$ then requestColor()
        end
    end
    $c_v \leftarrow max\chi(P_v)$ s.t. $\chi(P_v) \notin [d_v(w) - \lceil \gamma\zeta \log n \rceil, \ldots, d_v(w) + \lceil \gamma\zeta \log n \rceil]$ s.t.
    $w \in P_v$ and $\chi(P_v) \leq 0$
    while q $= A$ do
        $c_v \leftarrow c_v + 1$
        for $w \in P_v$ do $d_v(w) \leftarrow d_v(w) + 1$
        if $c_v \geq \lceil \sigma\Delta \log n \rceil$ then q $\leftarrow C_i$
        else
            w.p. $1/(\kappa_2\Delta)$ do broadcast($m^i_A(v, c_v)$)
            if $m^i_C(w)$ received then q $\leftarrow A_{succ}$; $L_v \leftarrow w$
            if $m^i_A(w, c_w)$ received then
                $P_v \leftarrow P_v \cup \{w\}$; $d_v(w) \leftarrow c_w$
                if $|c_v - c_w| \leq \lceil \gamma\zeta \log n \rceil$ then $c_v \leftarrow \chi(P_v)$
            end
        end
    end
end
/*$\mathcal{O}(\Delta \log n)$ rounds w.h.p. [50]: */
```

Algorithm C.27: requestColor subroutine – algorithm 2 in [50].

```
requestColor()
    repeat
        w.p. $1/(\kappa_2\Delta)$ do broadcast($m_R(v, L_v)$)
    until $m_C^0(L_v, v, tc_v)$
    q ← $A_{tc_v \cdot (\kappa_2+1)}$
end
```

Algorithm C.28: playLeader subroutine – algorithm 3 in [50].

```
playLeader()
    $c_v \leftarrow i$
    if $i > 0$ then
        repeat w.p. $1/(\kappa_2\Delta)$ do broadcast($m_C^i(v)$)
        until protocol stopped
    end
    else
        $t_c \leftarrow 0$; $\mathcal{Q} \leftarrow \emptyset$
        repeat
            if $m_R(w, v)$ received and $w \notin \mathcal{Q}$ then $\mathcal{Q} \leftarrow \mathcal{Q}\{w\}$
            if $\mathcal{Q} = \emptyset$ then
                w.p. $1/\kappa_2$ do broadcast($m_C^0(v)$)
            end
            else
                $t_c \leftarrow t_c + 1$; $w \leftarrow pop(\mathcal{Q})$
                for $j \in \lceil \beta \log n \rceil$ do
                    w.p. $1/\kappa_2$ do broadcast($m_C^0(v, w, t_c)$)
                end
            end
        until protocol stopped
    end
end
```

Algorithm C.29: Interval coloring in BE [25].

```
Constants: $\eta = 1/16$ /*safety boundary factor*/
State set: $Q : \{A, C\}$
Initialize: q $\leftarrow A$; c$_v \leftarrow \varepsilon$; /*uncolored */; $(\tilde{d_v}, b_v) \leftarrow$ Estimate($S$)
forever do
    if q $= A$ then $p_v \sim$ getFreeSlots($b_v$)
    $jitter_v \sim [0, \ldots, 1]$
    $S \leftarrow$ listen($p_v + jitter_v$) $\cup$ beep $\cup$ listen($Q - p_v - jitter_v$)
    c$_v \leftarrow$ calculateInterval($S$)
    $(\tilde{d_v}, b_v) \leftarrow$ Estimate($S$)
    if beepsHeardInRange($S[p_v - b_v, p_v + b_v]$) $= \varnothing$ then q $\leftarrow C$
    else if beepsHeardInRange($S[p_v - 1, p_v + 2]$) $\neq \varnothing$ then q $\leftarrow A$
end
getFreeSlots($b_v$) /*Leave safety boundaries*/
calculateInterval($S$) /*Span free time before beep*/
Estimate($S$)
    $\tilde{d_v} \leftarrow \max(|S|, 1)$
    $b_v \leftarrow \eta \cdot Q/\tilde{d_v}$
end
/*$\mathcal{O}(\log n)$ periods w.h.p. [25]: */
```

Algorithm C.30: Self-stabilizing LE in odd cycles – protocol 6 in [11].

```
State variables: $bullet_u$; $leader_u$; $probe_u$; $phase_u$ /*switch between generating a
                 probe or moving itself forward*/
if $label_u = label_v$ then
    if $probe_u = 1$ then $leader_u \leftarrow 1; probe_u \leftarrow 0$
    if $phase_u = 0$ then
        $phase_u \leftarrow 1$
        if $leader_v = 0$ then $probe_v \leftarrow 1$
    end
    else if $probe_v = 0$ then
        $label_v \leftarrow \overline{label_v}; phase_v \leftarrow 0; bullet_v \leftarrow 0$
    end
else if $leader_v = 1$ then
    if $bullet_v = 1$ then $leader_v \leftarrow 0$
    else $bullet_u \leftarrow 1; probe_u \leftarrow 0$
else
    if $bullet_v = 1$ then $bullet_v \leftarrow 0; bullet_u \leftarrow 1$
    if $probe_u = 1$ then $probe_u \leftarrow 0; probe_v \leftarrow 1$
end
```