Alexander Gogolev, MSc, BSc

Randomized Binary Majority Consensus: Efficiency and Robustness

DISSERTATION

to gain the Joint Doctoral Degree

Doctor of Philosophy (PhD)

Alpen-Adria-Universität Klagenfurt

Fakultät für Technische Wissenschaften

in accordance with
The Erasmus Mundus Joint Doctorate
in Interactive and Cognitive Environments
Alpen-Adria-Universität Klagenfurt | Universitá degli Studi di Genova

Supervisors:

Univ.-Prof. Dr.-Ing. Christian Bettstetter
Institut für Vernetzte und Eingebettete Systeme
Alpen-Adria-Universität Klagenfurt

Prof. Dr. Lucio Marcenaro
Dipartamento di Ingegneria, Navale, Elettrica,
Elettronica e Delle Comunicazioni
Universitá degli studi di Genova

April 2014

Declaration of honor

I hereby confirm on my honor that I personally prepared the present academic work and carried out myself the activities directly involved with it. I also confirm that I have used no resources other than those declared. All formulations and concepts adopted literally or in their essential content from printed, unprinted or Internet sources have been cited according to the rules for academic work and identified by means of footnotes or other precise indications of source.

The support provided during the work, including significant assistance from my supervisors has been indicated in full.

The academic work has not been submitted to any other examination authority. The work is submitted in printed and electronic form. I confirm that the content of the digital version is completely identical to that of the printed version.

I am aware that a false declaration will have legal consequences.

(Signature) (Place, date)

First Reviewer:

Univ.-Prof. Dr.-Ing. Christian Bettstetter

Institut für Vernetzte und Eingebettete Systeme
Alpen-Adria-Universität Klagenfurt

Second Reviewer:

Prof. Dr. Salima Hassas

Laboratoire d'Informatique en Image et Systémes d'Information
Université Claude Bernard Lyon 1

Acknowledgments

This PhD Thesis has been developed in the framework of, and according to, the rules of the Erasmus Mundus Joint Doctorate on Interactive and Cognitive Environments EMJD ICE [FPA n° 2010-0012] with the cooperation of the following Universities:

Alpen-Adria-Universität Klagenfurt – AAU

Queen Mary, University of London – QMUL

Technische Universiteit Eindhoven – TU/e

Università degli Studi di Genova – UNIGE

Universitat Politècnica Catalunya – UPC

According to ICE regulations, the Italian PhD title has also been awarded by the Università degli Studi di Genova.

This work was supported by Lakeside Labs GmbH, Klagenfurt, Austria with funding from the European Regional Development Fund and the Carinthian Economic Promotion Fund (KWF) under grant 20214/21530/32602.

Lakeside Labs GmbH – Klagenfurt, Austria

Acknowledgements

I joined the Erasmus Mundus Joint Doctorate program on Interactive and Cognitive Environments (EMJD ICE) in April of 2011. Since then I have worked in Mobile Systems Group at University of Klagenfurt and in Information and Signal Processing Group at University of Genoa. This thesis is a result of my research conducted under the supervision of Prof. Christian Bettstetter, Prof. Lucio Marcenaro, and Prof. Carlo Regazzoni. I am grateful for the supervision, motivation, and help that they have provided me with.

I would like to thank Prof. Salima Hassas from Université Claude Bernard Lyon 1 for reviewing this thesis and Prof. Hermann De Meer from Universität Passau for being the second examiner. Some ideas behind this work have been inspired by their proficient research.

During my work in Klagenfurt and Genoa I found new friends and colleagues, from whom I learned a lot. I would like to thank Nikolaj Marchenko who has spent endless hours straightening my writing. I would also like to thank Johannes Klinglmayr, Helmut Adam, and Evşen Yanmaz, who found time to help me with whatever I could think of. Special thanks goes to Kornelia Lienbacher, whose intense care saved me a huge effort in many areas.

During my time at NES I was surrounded by a great team of colleagues. I kindly thank my fellow researchers Torsten Andre, Wilfried Elmenreich, Sérgio Crisóstomo, István Fehérvári, Pasquale Grippa, and many others.

At University of Genoa I was lucky to meet people with whom I had fruitful discussions both on scientific and non-scientific topics. I would like to thank Krešimir Dabčević, Pietro Moreiro, Simone Chiappino, Jorge Reyes Ortiz, Maira Carhvalho and many, many others.

I would like to thank my sister Anastasiya, and my parents for believing in me and helping me on the way to accomplish this. My very special thanks is to my wife Marina who filled my last six years with joy and love.

Klagenfurt, April 2014 Alexander Gogolev

Abstract

Decision making by means of distributed consensus algorithms can be used in systems where centralized control is difficult or impossible. Algorithms that perform such a coordination in a fixed period of time — wait-free algorithms — can be beneficial in real-life distributed systems. These algorithms should be efficient and robust towards various disturbances, since the distributed consensus problem becomes non-trivial in real-life networked systems, restricted in connectivity and time.

Solutions that can solve consensus problem in such systems often decrease efficiency with stochastic disturbances. At the same time, some other algorithms can increase performance with disturbances. Employment of stochastic disturbances to promote efficiency and robustness of consensus is called *randomization*. Randomization has limited application to wait-free algorithms, due to the stochasticity it employs.

In this thesis we focus on wait-free algorithms for binary majority consensus with stochastic elements. First, in Chapter 4 we show that two standard algorithms, Gacs-Kurdyumov-Levin (GKL) and Simple Majority (SM) consensus, converge more often if randomized by noise and message loss in ordered and topologically random networks.

Next, in Chapter 5 we propose a Random Neighbor Majority (RNM) consensus with embedded randomization by neighbor selection. We show that with additional randomization by noise and errors RNM can outperform the GKL and SM in asynchronous environments.

Chapter 6 investigates the impact of the faulty nodes on consensus. We investigate faulty nodes with random, full, and persistent failure with different layout over the network. We show that faulty nodes with persistent failure are more adverse for binary majority consensus than faulty nodes with random and full failure. We show that randomization can promote robustness towards faulty nodes, and that RNM is more robust towards faulty nodes than GKL and SM.

Chapter 7 concludes the thesis with summary of results. We explain open issues and discuss further research directions.

Contents

List of Figures

CHAPTER 1
Introduction

The term *consensus* means “agreement” and is widely used in the literature and everyday life. Its etymology roots down to two Latin words: *con* and *sentio* which mean “together” and “feel”. The *consensus problem* therefore is an *agreement problem.* It emerges in systems, where a set of communicating entities is expected to agree on some opinion, based on their initial suggestions. Every entity in such a system can make decisions on what the negotiated value is and exchange this value with other entities. The phenomenon of emerging agreement among multiple entities can be widely seen in the everyday life of human society. This is why consensus problems were initially studied in a framework of social studies [TF83, RGH77, FD88, KFN92].

However, with growth of information networks consensus problems came into focus of computer science. From a perspective of computer science, a consensus problem is generally considered as a problem of distributed network management. An algorithm that can steer all agents to a required state can be beneficial in a distributed network of communicating agents. Such distributed *consensus algorithms* can be used in systems where centralized decision-making is difficult or impossible. Such tasks can arise, for example, in sensor fusion [OSS05], object tracking [GC96], localization [SKB10], or distributed mission planning [AH06]. Technical studies generally consider consensus problems in weakly connected distributed networks with additional restrictions, such as limited time and, sometimes, random disturbances. These limitations can yield non-linear dynamics similar to that of natural complex systems.

Simple consensus algorithms operating in such distributed systems with limited connectivity can produce global coordination [MMDA04]. The complex behavior, emerging out of simple rules in weakly connected networked systems is often referred to as one of the features of *self-organization* [HMB08, Gac01]. Self-organization is a property often attributed to various real-world complex systems [HMSKC06]. Research on self-organization can provide insightful results and lead to development of efficient bio-inspired networked algorithms such as, e.g., firefly synchronization [MS90, HMSKC06, KKBT12, FH02]. It is a subject of a broad interest to physicists, complex systems researchers and computer scientists.

In a similar way, studies on consensus problems can provide insightful results in the area of complex systems. If a distributed consensus problem is solved, it can result in a class of distributed agreement protocols for information networks.

Depending on a research perspective, a general consensus problem can be subdivided into several sub-areas, each of them having their own limitations and a rather diverse behavior. Scholars distinguish average and binary, deterministic and randomized, waiting and wait-free consensus.

Real-life networked systems, such as wireless sensor networks or networks of unmanned flying vehicles seldom operate in a disturbance-free manner. Studies show that stochastic disturbances can promote simple consensus algorithms [MMDA04]. In this thesis we investigate whether a randomization by stochastic disturbances can promote wait-free binary majority consensus.

1.1 Binary Majority Consensus

Studies on a *general consensus* problem consider systems that are expected to agree on a certain value. *Binary consensus* is a subset of the general consensus problem, where a network of communicating agents with initial binary states is expected to agree on one of those states. In a centralized system, or if each agent can communicate to all other agents, this problem can degrade to trivial. Binary consensus becomes non-trivial in distributed, weakly connected networks with restricted time frames and possible disturbances. Studies on binary consensus problem can reveal certain aspects of complex system dynamics but have limited applicability to the real-life networks. However, the binary *majority consensus* can be applied in real-life networks for, e.g., the tasks of distributed estimation or decision making. A binary *majority consensus* problem emerges when a network of communicating nodes is expected to agree on a single binary value that corresponds to the initial majority of the given values. Algorithms that tackle such problems are called *binary majority consensus algorithms*. In real-life networked systems algorithms with deterministic execution time can be beneficial. Such algorithms, providing distributed agreement in a fixed period of time, are called *wait-free* consensus algorithms. Therefore, a *wait-free binary majority consensus system* can be defined as a network of communicating nodes that are expected to agree on a single binary value that corresponds to the initial majority of the given values, in a given time. There are algorithms that solve a general binary consensus problem, even with presence of faulty-behaving nodes [BO83, CT96]. However, there is no full wait-free solution for binary majority consensus problem with aforementioned restrictions.

1.2 Efficiency and Robustness of Consensus Algorithms

A combination of all initial states of the networked nodes forms an initial system configuration. Due to the restrictions in time and connectivity it is difficult to guarantee an agreement for all possible initial configurations. Therefore, the efficiency of algorithms for binary majority consensus is generally evaluated with their *convergence rate*, R — a fraction of initial network configurations that results in a successful agreement. Clearly, an ideal algorithm that guarantees convergence should provide a convergence rate of 100%.

Binary majority consensus has been intensively studied since 1978, when Gacs *et al.* [GKL78] introduced a Gacs-Kurdyumov-Levin (GKL) algorithm that converges on $\simeq$ 82% of initial configurations in a 6-connected ring of 149 nodes. In the following decades scholars proposed several algorithms with R up to 86% in the same conditions. Land and Belew [LB95] showed that deterministic algorithms cannot reach convergence rate of 100%. However, this restriction does not apply to randomized algorithms. Randomized algorithms are algorithms that employ stochastic elements to increase both convergence rate and fault tolerance.

Robustness towards various disturbances is another vital issue for consensus algorithms that operate in real-life networked systems. Studies on robustness of consensus consider various disturbances, e.g., noise [KM07, Men11] or topological changes [SZ93, KB06] as threats that inhibit consensus. Other scholars consider stochastic disturbances as a beneficial intrusion that can promote consensus, e.g., in terms of fault-tolerance [Asp03]. The first approach appears in research for both simple and complex consensus algorithms, while the second one can mainly be traced in fault-tolerance studies of complex algorithms.

Simple binary majority consensus, however, can exhibit both positive and negative response to randomization. Some algorithms, such as GKL, show high efficiency in noiseless and error-free ordered networks but can significantly degrade if disturbed by topology or communication errors [MMDA04]. At the same time, other algorithms, e.g., Simple Majority (SM), can increase the convergence rate in random networks with errors [MMDA04] (which can be explained by the aforementioned randomization [SSW91, Fat13]). A challenging task, therefore, is to design an algorithm that is efficient and robust both in disturbed and disturbance-free environments. Several decades of research on distributed consensus show that for resource-constrained systems this task is difficult. However, it was recently shown that a relatively simple self-organizing algorithm, accompanied by stochastic disturbances can guarantee synchronization in a wide range of distributed systems [KKBT12]. These latest advances motivated our research.

In this thesis we focus on a wait-free binary majority consensus with stochastic disturbances. First, we study standard consensus algorithms in random networks with noise and random message loss, showing that these stochastic disturbances can promote consensus and hinder impact of faulty nodes in the system. Next, we propose a Random Neighbor Majority (RNM) — a simple randomized consensus algorithm that is more efficient and robust in both disturbed and undisturbed environments. Finally, we test the aforementioned algorithms for the robustness towards faulty node behavior.

1.3 Outline and Contributions

The remainder of this thesis is composed of six chapters. Chapter 2 defines the binary majority consensus problem in the scope of the general consensus problem. Chapter 3 motivates and explains system modeling. Original contributions are presented in Chapters 4-6. Conclusions are drawn in Chapter 7.

Chapter 2: Binary Majority Consensus

This chapter defines and explains the important differences between *general consensus* and *wait-free binary majority consensus*. It describes more strict termination conditions of the latter and explains their implications. An overview of the related work with different approaches and solutions to the binary majority consensus problem is given. Then we provide an outlook on the best solutions for the wait-free binary majority consensus problem and discuss studies that investigate consensus with disturbances and faulty nodes. We explain different approaches to randomizing disturbances, discuss origins of important modeling limitations, and introduce main concepts of efficiency and robustness.

Chapter 3: System Modeling

This chapter gives an overview of the models used for simulations of consensus algorithms. It explains the motivation behind the simulation engine and describes its features. Detailed explanations of network models, graph and node attributes, and updating functions are provided. Generation of initial configurations is explained and computation of the efficiency metrics is described. Then we provide characteristics of the network simulator with different settings and explain mechanisms of randomizing disturbances. Finally, we provide and explain pseudo code for selected simulator features.

Chapter 4: Performance of Standard Algorithms

This chapter investigates SM and GKL consensus in ring lattices, Watts-Strogatz and Waxman networks. It studies algorithms in networks of different size, with different node degree and scope of randomization. We compare the performance of algorithms under different randomizing disturbances, showing that an optimal combination of additive noise, message loss and topology randomization can increase their efficiency. We show that aforementioned randomizing disturbances do not critically decrease the convergence time of the algorithms. We explain the low-level mechanisms behind this and suggest how to embed them in a consensus algorithm without significant increase in complexity.

Chapter 5: Randomized Consensus

In this chapter we introduce the Random Neighbor Majority (RNM) consensus inspired by results presented in Chapter 4. We investigate two random neighbor selection schemes and two additional types of randomization — by noise and errors, showing that such a randomization can increase the efficiency of RNM up to 100%. RNM is studied in ring lattices, Watts-Strogatz and Waxman networks with different disturbances, to show that gains in efficiency are robust towards noise and topology randomization. The mechanisms behind these gains are explained and an optimal randomization scheme depending on system conditions is suggested.

Chapter 6: Consensus with Faulty Nodes

This chapter addresses the impact of the faulty nodes on SM, GKL, and RNM in ring lattices and random networks. We show that randomization can promote robustness towards faulty node behavior, and that RNM is more robust towards faulty nodes than SM and GKL. We show that faulty nodes with persistent failure always inhibit consensus, while commonly used Byzantine-like nodes with random failure and faulty nodes with full or "crash" failure have lower impact. Finally, we show that in some cases faulty nodes with random failure can promote binary majority consensus.

Chapter 7: Conclusions

This chapter concludes the thesis. We provide the motivation for our research and give the summary of results, explaining our contribution and drawing common conclusions. Finally, we show further research directions and describe remaining challenging issues.

this thesis present result of the research, conducted in cooperation with Nikolaj Marchenko, Christian Bettstetter and Lucio Marcenaro. Main contributions have been published in [1, 2, 3, 4] and some are still under review in [5].

CHAPTER 2

Distributed Consensus Algorithms

2.1 Introduction and Motivation

Binary majority consensus can be used in systems where centralized decision making is difficult or impossible. Such conditions may arise in various systems, ranging from purely technical [BG84] to social and natural networks [TInY$^+$13, BTV11]. In this thesis we focus on consensus algorithms from a perspective of networked and distributed technical systems. In such systems consensus algorithms and, in particular, majority consensus found first applications in distributed database management [Tho79, BG84, Kum91, MM81]. Later, consensus methods were developed in pattern recognition and detection [LS97, LEWV10], image analysis [FB81], bootstrapping [Wil96], and object tracking [GC96].

The rapid development of the computing technologies has offered new areas for the distributed decision making, such as DNA and gene analysis [GWR$^+$06, YWW07], data clustering [GF08], neural network classification [LM05], medical analysis [NBD$^+$09] and vehicular networks [Ren07].

In our work we investigate consensus algorithms from a perspective of distributed and wireless sensor networking. This particular interest is related to the author's background in distributed wireless networks and interest in phenomena of emerging complex behavior, such as self-organization [GKPS11]. In wireless sensor and mobile and cognitive networks consensus algorithms found broad use [OSS05, BDG12, SKB10] for various tasks, ranging from spectrum sensing [AMM11] to mission planning [AH06].

2.2 General Consensus and Binary Majority Consensus

Consensus algorithms are a class of algorithms for decision making. Such algorithms, deployed in a networked system, aim to steer a system to a common state for all networked nodes. Distributed consensus systems are often restricted by network connectivity, so that each node can only access information from a limited number of its neighboring nodes. This assures distributed execution similar to that of real-life systems and leads to emerging complex dynamics. Additional restrictions, such as communication noise, lack of synchrony, or random link failure, can further exaggerate the execution dynamics. Such restrictions make it difficult to predict, whether the system will reach an agreement or not.

All consensus algorithms, whether it is average or binary consensus, share certain concepts and have to satisfy the following conditions [Asp03]:

1. *Agreement.* All nodes choose the same value.
2. *Termination.* All non-faulty nodes eventually decide.
3. *Validity.* The common output value is an input value of some node.

One can see that these conditions do not necessarily require that the agreed value corresponds to the initial majority of values, nor it requires a time boundary for the agreement process.

2.2.1 Wait-Free Consensus Algorithms

Consensus algorithms with bounded execution time are called *wait-free* consensus algorithms. Wait-free consensus algorithms are a sub-class of consensus algorithms that have specific *termination* condition. These algorithms are terminated after a predefined time T, whether an agreement was reached or not. Wait-free consensus algorithms can be beneficial in real-life networked systems where termination time is important. Bounded execution time, however, can lead to lower performance and higher sensitivity to disturbances [GKL78, MMDA04, SZ93].

2.2.2 Binary Consensus Algorithms

Binary consensus is yet another sub-class of consensus algorithms with a specific *agreement* condition. Binary consensus algorithms are used to steer a system to an agreement on a state or a value, selected out of limited set of binary states, generally defined as $\{0, 1\}$ or $\{-1, 1\}$.

2.2.3 Wait-Free Binary Majority Consensus

Binary majority consensus is also known as density classification or majority sorting consensus. This is a further specification of the *agreement* condition of the *wait-free binary consensus*. Such a consensus assumes that a network converges to a binary state, corresponding to the initial majority of all states in the network.

Therefore, in a *wait-free binary majority consensus* system of N nodes, with initial binary states $\sigma_i[0] \in \{-1, 1\}$, agreement is reached if there exists a time $t_c \in \{0, T\}$, so that $\sum_i \sigma_i[t_c] = -N$ for $\sum_i \sigma_i[0] < 0$ or $\sum_i \sigma_i[t_c] = N$ for $\sum_i \sigma_i[0] > 0$. Here, a time boundary T is equal to $2N$ [GKL78]. At a time $t = T$ an algorithm is terminated whether an agreement was reached or not.

2.3 Basic Terminology and Performance Metrics

Let us briefly specify some of the basic concepts and terms we use in this work. We study undirected networks consisting of N nodes, each having $2K$ neighbors. Nodes $j, i \in \{0, \ldots, N\}$ are called neighbors if they are connected with a link. State of the node i at the time t is denoted as $\sigma_i[t]$. Initial state $\sigma_i[0]$, or σ_0 is a state, randomly assigned to each node at the time $t = 0$. The set of N states $\sigma_i[0]$ is called initial configuration, and denoted as I_0. The sum of all states in the initial configuration $\sum_{i=0}^{i=N} \sigma_i[0]$ is called initial density and denoted as $\rho[0]$ or ρ_0. Similarly, the density at each time step $t \in \{0, \ldots, T\}$ of the system evolution is denoted as $\rho[t]$. We refer to *connectivity* as an average number of neighbors per node in the system. Therefore, a network with “lower” or “weaker” connectivity has on average less neighbors per node than the one it is compared with. The term *interconnectivity* refers to an average link span in the system. Therefore, a “system with higher interconnectivity” means that such a system has more “long” links than the one it is compared with. The term *link span* which is also referred to as a *link length*, depends on network type: (a) in Watts-Strogatz networks link length is an absolute difference between indices $|i - j|$, and (b) in Waxman networks it is as an actual Euclidean distance between nodes i and j. We explain and motivate these metrics in more detail in Section 3.2.

Here and in the following chapters we refer to a *consensus* as a process of negotiation. The term *agreement* is referred to as a final stage of *consensus* process, when all networked nodes have reached the common state. We employ the term *convergence rate* denoted as R, as a fraction of initial configurations that result in a successful agreement. We refer to the *efficiency* or *performance* of consensus algorithms as convergence rate. *Convergence speed* is denoted as F and defined as a fraction $F = \frac{t_c}{T}$, where t_c is the time step when the agreement was reached, and T is a consensus cycle duration, $T = 2N$ [GKL78].

We employ the term *fault tolerance* as defined by Laprie in [Lap95] and later specified by Avizienis *et al.* [ALRL04]: "...fault tolerance is a means to maintain system function in the presence of faults ...". We use the term *robustness* as defined in [DMS09]: "...robustness ...is the ability of the system to sustain a satisfactory level of the function under system perturbations including partial failure ...". We distinguish these terms here because in our work they often overlap, e.g., some types of faulty node behavior include full or "crash" failure, which can be interpreted as a partial system failure. Hence, we relate"fault tolerance" to the ability of the system to tolerate the faulty node behavior and the term "robustness" to the ability of the system to overcome a wider range of disturbances.

General consensus can be also characterized with other metrics, such as, e.g., *convergence time* or *convergence assurance*. First one is defined as the time that an algorithm takes to reach an agreement, and is not suitable to wait-free consensus as it generally requires the relaxed time limitations. The second metric is defined as a probability of successful consensus termination, and is not directly applicable to binary majority consensus as due to the imposed restrictions the 100% convergence cannot be guaranteed [LB95].

2.4 State of the Art

2.4.1 Consensus with Disturbances

In this thesis we study the performance of binary majority consensus randomized by topology, noise, message loss, and faulty nodes. During last several decades some of these aspects were addressed by other scholars.

It was shown that noise and errors can decrease the convergence rate of consensus algorithms in general, and wait-free consensus in particular [GKL78, MMDA04]. Similar effects registered in systems with disruption of synchrony [DDS87], presence of faulty nodes [PSL80, FLP85], or randomized topology [Bea05, KB06]. Let us briefly describe the studies on the impact of various disturbances on consensus.

2.4.2 Impact of System Synchrony

System-wide synchrony is a substantial factor for the convergence rate of consensus algorithms. A synchrony level in a consensus system is generally defined through a difference in nodes mutual decision moments. Such a scheme, however, assumes that all networked nodes have equal decision periods (or decision frequency), i.e., asynchrony is imposed via the "phase-shift", and not the "frequency drift". Generally, the following approaches to modeling of synchrony levels can be distinguished in a consensus system:

- full synchrony; when all networked nodes decide simultaneously [GKL78, MMDA04, MHC93],
- full asynchrony; when nodes take decisions independently from each other and previous states of the system [BO83],
- conditional asynchrony or partial synchrony; when nodes decide independently from each other, but the update order (sequence) is predefined to a certain extent [MMDA04].

The first approach is recognized as a rather artificial one [MMDA04] for two reasons: fully synchronous distributed real-life systems are rare, and, in systems that can reach full synchrony, binary consensus problem can become trivial [LDCH10, RBA05].

The second approach is close to real life, but the consensus problem becomes significantly more difficult to tackle. To make consensus possible with full asynchrony, scholars relax other restrictions, e.g., allowing full, or nearly full, network connectivity or guaranteed message delivery [BO83].

The third approach presents a trade-off between the two previous ones. It was shown [KKBT12] that partial synchrony can be achieved in a distributed system. This makes the partial synchrony a more realistic restriction than, e.g., full connectivity. In this thesis we consider all three approaches.

Influence of synchrony on consensus algorithms has been previously studied in multiple works, including contributions by Bracha and Tueg [BT85], Dolev *et al.* [DDS87] and Dwork *et al.* [DLS88]. Latter works studied the boarder cases, investigating the minimum required synchrony and influence of partial synchrony on consensus, respectively.

Studies generally show that lack of system-wide synchrony strongly inhibits consensus. However, real-life distributed systems seldom work in a synchronous manner [FGEM11] and *asynchronous* consensus problem has been further tackled with *randomized* consensus algorithms. Comprehensive overview by Aspnes [Asp03] gives detailed explanation and description of existing approaches; papers [CIL94, BHKPS06, Asp02] offer new solutions, while the paper by Attiya and Censor [AC07] sets bounds for asynchronous and randomized consensus algorithms. A randomized consensus algorithm that guarantees convergence in asynchronous networks was proposed by Ben-Or [BO83].

2.4.3 Impact of Noise and Errors

Impact of noise and errors on a consensus is a subject of a broad interest among scientists for obvious reasons — real-life systems rarely operate in an undisturbed manner. Two general (and often overlapping) paradigms can be distinguished in consensus studies: the first one considers noise and errors as a threatening disturbance, while the second one counts them as beneficial ones. This can be rooted to the level of refinement of the investigated algorithms. If an algorithm is refined to perform in certain conditions, stochastic disturbances can decrease its efficiency [GKL78, MMDA04, RM08]. On the other hand, if the algorithm is simple and does not indicate competitive performance in refined environments, disturbances can be beneficial [Asp03, MMDA04]. Influence of noise on consensus algorithms has been previously widely covered in [Asp00, Asp02, CFF$^+$07, FS08, HM07b, HM07a, KM07, MMDA04, RW11, TN09, WE10]. These works generally consider noise as a negative disturbance that inhibits performance. However, some studies on noisy consensus, e.g., [MMDA04, BT85], show that noise can have positive influence. Latter works served as main motivation for research presented in this thesis.

2.4.4 Impact of Topology Randomness

Studies on topology influence on consensus generally consider two aspects: impact of decreasing or increasing connectivity [TK12a, TK12b], and the impact of randomly switching topologies [Bea05, KB06]. While a decrease in connectivity generally inhibits consensus, randomly switching topologies can produce a twofold influence. In some cases topological randomization decreases the performance (e.g., if an algorithm is specifically designed to perform in certain topologies) [GKL78, MMDA04]. Other cases show the increase in performance under topology randomization. This can be sometimes contributed to the increase of the efficient interconnectivity of the network, as, e.g., in Watts-Strogatz randomization [MMDA04, 1]. Topology randomization in weakly connected WS networks explicitly increases the average link length in the graph, thus increasing the interconnectivity. It also changes the network by adding random links that can counter the network clustering process [BR99]. A different case of topology randomization is illustrated in Figures 2.1, 2.2 and 2.3 where the network topology is left intact but nodes can randomly access state information from available neighbors. They represent a situation of, e.g., random link failure or message loss, when a node can access C out of $2K$ available neighbors. Thus, Figure 2.1 presents a node (n_3) connected to all of its $2K = 6$ neighbors. Figure 2.2 presents a node connected to $C = 4$ neighbors randomly selected at each time step. Figure 2.3 presents a node, connected to $C = 2$ neighbors randomly selected at each time step only from the left side. Such a randomization yields a twofold impact: it can decrease the information exchange (if $C < K$), but at the same time it counters topological clustering [BR99]. This inspired the random neighbor selection schemes for Random Neighbor Majority (RNM) consensus in Chapter 5.

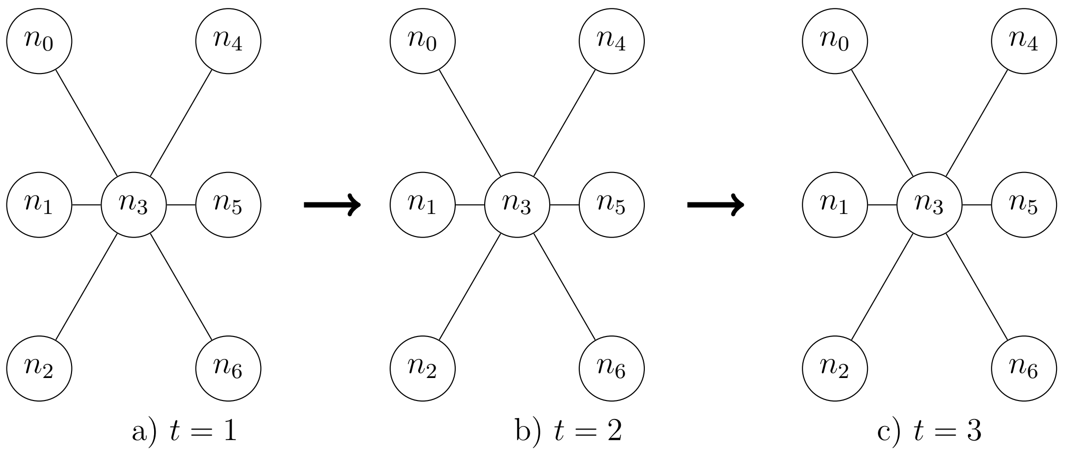


Figure 2.1: A networked node n_3 connected to its $2K$ neighbors. $K = 3$.

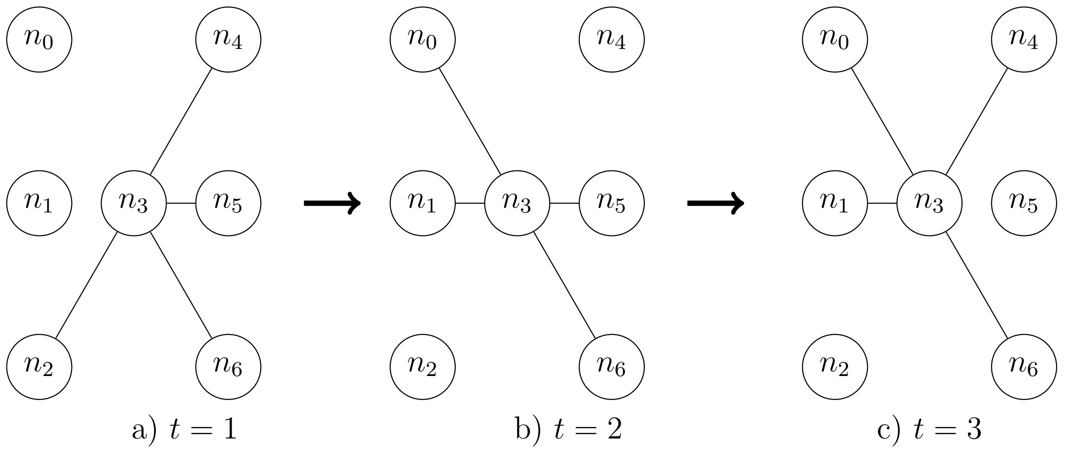


Figure 2.2: A node n_3 connected to C randomly selected neighbors. $K = 3, C = 4$.

2.4.5 Impact of Faulty Node Behavior

The faulty node behavior is often considered as one of the main impediments to consensus. A faulty node is generally defined as a Byzantine faulty node [PSL80, FLP85], i.e., a node with any kind of failure except full failure. It actively takes part in the consensus process but can supply its neighbors with wrong information. Initial studies by Pease *et al.* [PSL80] show that in a *synchronous* system of N nodes, M of them being faulty, agreement is possible if $M \leq \frac{N-1}{3}$. This condition was later strengthened by Fischer, Lynch, and Patterson [FLP85, FLM85], who showed that in *asynchronous* systems even a single faulty node can prevent consensus. Latter restriction became known as *FLP-impossibility*, named after the scholars names. FLP-impossibility was tackled multiple times (e.g., [GY89]) but was not disproved.

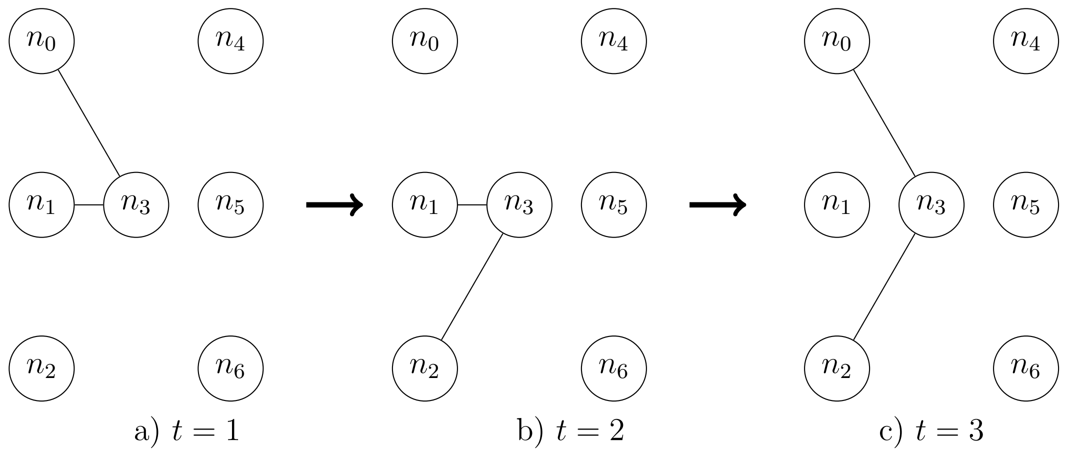


Figure 2.3: A node n_3 connected to its C random neighbors from the left. $K = 3, C = 2$.

Later studies compare the impact of Byzantine faulty nodes with dormant faulty nodes [MP91] and crash-failing faulty nodes [AFJ06], showing that consensus can stabilize and overcome the impact of such non-Byzantine nodes.

The issue of fault tolerance is generally addressed with increasing synchrony, failure detection [CT96, CHT96, ACT98] and randomization [Asp03]. Randomized consensus algorithm show higher robustness towards faulty node behavior [MNC10] than deterministic solutions. Some randomized algorithms can guarantee the agreement in systems with $M \leq N/2$ faulty nodes in the system [BO83]. We address the impact of faulty nodes on consensus in Chapter 6 of this thesis.

2.4.6 Overview of Algorithms

Binary majority consensus has more strict termination conditions than a general consensus. Together with low connectivity and time limitations it leads to a weaker convergence assurance. It has, however, raised significant interest since Gacs *et al.* [GKL78] proposed an algorithm that converges on $\simeq 82\%$ of initial configurations. Since then scholars offered several more efficient solutions to the problem using the same setup — a synchronized and disturbance-free 6-connected ring of 149 nodes. We summarize these solutions in Table 2.1.

Let us briefly describe some of the main achievements in the area. The best (to that moment) solution for density classification was obtained with the genetic programming by Andre *et al.* [ABK96]. Right after this, Fukś [Fuk97] offered a combination of two deterministic rules that converges in networks with ρ_0, close to 0.5 with $R \simeq 100\%$. This solution, however, does not scale into lower or higher densities. Indeed, just two years prior to that, Land and Belew [LB95] show that

Table 2.1: Efficiency of wait-free algorithms for binary majority consensus. $N = 149$, $K = 3$.

Rule	Year	Design	Ref.	R,%	ρ_0	Number of test samples
GKL	1978	human-designed	[GKL78]	81.6	Gaussian	10^4
Davis	1995	human-designed	[ABK96]	81.8	Gaussian	10^4
Das	1995	human-designed	[DCMH95]	82.2	Gaussian	10^4
Koza	1996	genetic programming	[ABK96]	82.3	Gaussian	10^4
Juille and Pollack	1998	co-evolution	[JP98]	86.0	Gaussian	10^4
Fukś	1997	combination of rules	[Fuk97]	100.0	0.5	10^4
Simple Majority	2004	human-designed, randomized	[MMDA04]	85.0	Gaussian	10^3
Traffic Majority	2013	human-designed, randomized	[Fat13]	90.0	Biased	10^4

no deterministic rule can guarantee 100% convergence rate in a reference setup. However, this restriction does not apply to the randomized algorithms. Moreira *et al.* [MMDA04] show that a Simple Majority rule can reach a convergence rate of 85% if randomized by topology and errors (although authors themselves do not mention the term “randomization”, the described effects match that of the randomized consensus).

Later, Fates [Fat13] offered the randomized Traffic Majority rule that solves the task in a reference setup with $R = 90\%$. This efficiency is achieved over test sets with biased initial density which can contribute to higher efficiency, but this does not decrease the significance of the result.

Note that the convergence rate of the algorithm can significantly vary depending on the simulation parameters, such as N, K, and the distribution of density in the initial configurations. Due to this scholars register the efficiency of algorithms in a “reference setup”, introduced in [GKL78]: an 6-connected ring of 149 nodes, synchronized and disturbance-free.

Simple Majority Consensus

In this work we study two of the aforementioned standard algorithms for binary majority consensus. Let us briefly define and explain them. First one is the

Simple Majority SM [MM81, MMDA04] consensus. SM consensus is arguably the simplest algorithm for solving the binary majority consensus. In a networked system of N nodes, each node $i \in \{0, \dots, N\}$ driven by a SM consensus calculates its new state on the basis of information about its current state and states of its $2K$ closest neighbors. I.e.:

$$\sigma_i[t+1] = G\left(\sum_{j=i-K}^{j=i+K} \sigma_{i,j}[t]\right) . \tag{2.1}$$

Here, $\sigma_{i,j}[t]$ denotes the state of node j, received by the node i at the time t. The update function $G(x)$ is defined as [MMDA04]:

$$G(x) = \begin{cases} -1 & \text{for } x < 0, \\ +1 & \text{for } x > 0. \end{cases} \tag{2.2}$$

Simplicity of the SM consensus can result in low convergence rate. In a reference setup (for more details, see Section 4.3.1) with a flip-coin generated sets of initial configurations (see Section 3.7) it shows $R \simeq 1\%$. However, Moreira *et al.* [MMDA04] show that with randomization by errors and topology in Watts-Strogatz networks SM can increase R and outperform GKL.

Gacs-Kurdyumov-Levin Consensus

The Gacs-Kurdyumov-Levin (GKL) consensus, proposed in [GKL78], is essentially a modification of the SM with a built-in state-dependent direction bias. According to GKL, each node calculates its new state based on the information of its own current state and states of its closest neighbors.

$$\sigma_i[t+1] = \begin{cases} G\Big(\sigma_{i,i}[t] + \sigma_{i,i+1}[t] + \sigma_{i,i+3}[t]\Big) & \text{for } \sigma_{i,i}[t] < 0, \\ G\Big(\sigma_{i,i}[t] + \sigma_{i,i-1}[t] + \sigma_{i,i-3}[t]\Big) & \text{for } \sigma_{i,i}[t] > 0. \end{cases} \tag{2.3}$$

In GKL, each node can select from which side of the lattice it will receive messages. Selection is based on its own current state: if $\sigma_{i,i} < 0$, it receives messages from the nodes on its “left” side, otherwise it receives messages from the noses on the “right” side.

This state-direction bias enables GKL to wash out clusters of nodes that have the same states. GKL is known among best algorithms for binary majority consensus and is often used as a benchmark for the newly developed algorithms. As we mention above, the network model from [GKL78] (a $2K = 6$-connected ring of $N = 149$ nodes) has been also adopted as a “reference setup” to measure the efficiency of algorithms.

Here we define consensus algorithms for ordered networks on the example of ring lattices. Such algorithms have to be adjusted to perform in random networks. These adjustments are described in detail in Section 4.2.1.

Impact of Initial Configuration

Initial system configuration I_0 is defined as a set of random binary states, initially assigned to all nodes in the system. Recall that a system is expected to agree on a state that corresponds to the initial majority of the states in the initial configuration.

Ideally, binary majority consensus algorithm should converge on all possible combinations of initial configurations, but a simulation over 2^N combinations is not feasible for the large N. Nevertheless, simulations should be performed over the sufficient number of initial configurations to ensure stable R. Our simulations show that if the simulations are performed over tests set with fewer than 10^4 initial configurations the resulting convergence rate may vary within the range of several percent. The most intuitive and simple way to obtain initial states for each node in the network is flip-coin trial, returning 1 and -1 with equal probabilities.

Our simulations also show that test sets obtained by a flip-coin procedure generally result in lower R for both GKL and SM compared to test sets with uniform ρ distribution, used by some scholars [MHC93]. This can be due to the fact that test sets, generated with a flip coin-trial, result in ρ_0 normally distributed around $N/2$, and initial configurations with densities close to $N/2$ are the most difficult for wait-free binary majority consensus [MHC93, Fuk97]. In our simulations we mainly use test sets generated with a flip-coin approach as it is the simplest and the most intuitive one. We compare the convergence rate obtained from test sets generated with both flip-coin and uniform ρ_0 distribution in Section 4.3.1.

Solutions Obtained by Evolutionary and Genetic Programming

The binary majority consensus problem has been in the focus of complex systems scholars for several decades. Apart from studies offering deterministic and randomized human-designed solutions, we would like to distinguish studies that approach the problem with genetic and evolutionary programming, for they have achieved some of the best results of their time. Evolutionary and genetic programming use a “black box” approach when a system is evolved by means of state transitions and permutations to satisfy a given fitness function. It case of consensus, a system is expected to evolve rules that can provide a distributed agreement. In such a case the scholar defines the fitness function, transitions and permutations, but does not know the output of the algorithm from the beginning of the simulation.

Using this method, in the early 90's Das and Mitchell [DMC94, DCMH95, MHC93, MCH94] have tackled the distributed consensus, offering some of the most comprehensive papers on the problem. Later, Juille and Pollack [JP98] approached the density classification with slightly more complex tools, offering yet again the best solution to a date. However, later development of randomized algorithms resulted in algorithms with even higher efficiency.

2.5 Randomized Consensus Algorithms

Randomization is a technique that utilizes random disturbances to increase the convergence rate and the fault tolerance of consensus algorithms [Asp03]. The controlled randomizing intrusions were initially offered as a tool to overcome the impact of Byzantine faulty nodes in relatively complex multi-step consensus algorithms. Randomization can be beneficial due to its relative simplicity and because it does not require relaxed system restrictions. Additionally, stochastic disturbances are intrinsic for various real-life distributed systems and can be easily utilized. Intensive studies on randomized consensus can be traced back for more than thirty years to an algorithm proposed by Michael Ben-Or [BO83] that solves consensus with faulty nodes in a fully asynchronous system. Since then several randomized algorithms were proposed [Asp03, BR92, Cha96, FZ08, CUBS02, FB81, SSW91, MNC10]. These works study consensus with various types of randomization and restrictions, showing increase in convergence rate and fault tolerance, but most of the proposed algorithms are complex or require relaxed system restrictions.

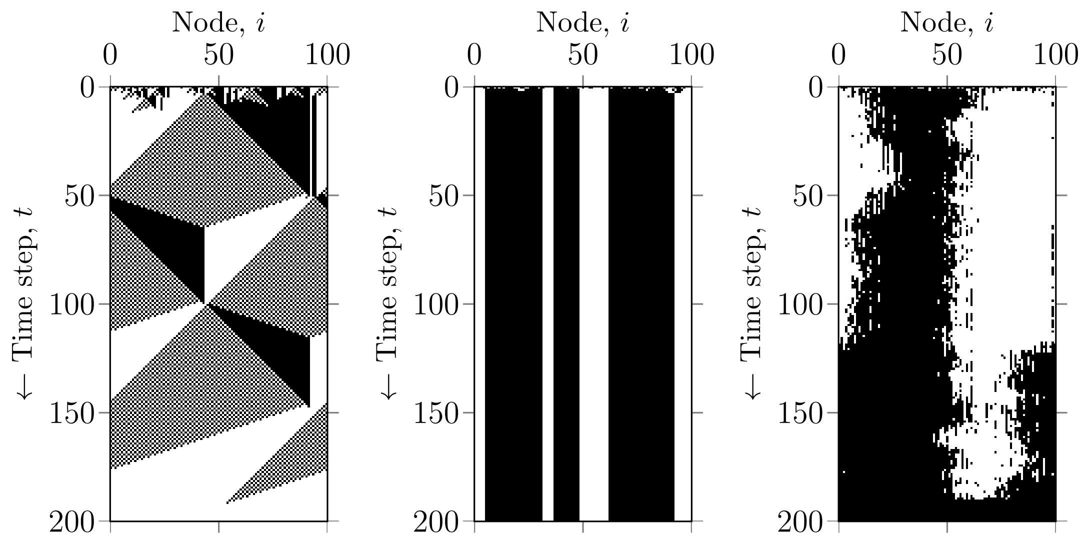
Simple algorithms have low fault-tolerance, do not guarantee the convergence, and the best ones can strongly deteriorate with disturbances [GKL78, MMDA04]. This is why the impact of randomization on simple consensus algorithms and, in particular, on wait-free binary majority consensus was not extensively studied: Moreira *et al.* studied the impact of random errors and topology on GKL and SM, and Fatés [Fat13] considered Traffic Majority consensus with random switching. In this work we focus on simple algorithms that can exhibit efficient system-wide coordination in restricted distributed systems with noise, message loss, errors, and random topology.

Principles

There are two common approaches to introduce randomization in the system. First approach assumes randomization of the environment, e.g., by noise [HM07a], random delays [BHKPS06] or randomly switching topologies [KB06]. Second approach assumes randomization within the algorithm, e.g., noisy scheduling, initially offered by [BT85] and later developed by [Asp02].

Benefits

Randomized consensus can overcome the “phase-lock” or “clustered” states of the system due to random disturbances that can destabilize such states. In terms of complex systems theory, a “phase-lock” or a “clustered state” is a stable attractor of the dynamic system. Such a state is undesirable but stable and is often reached by the system. In a binary majority consensus system such state can be represented by a network of nodes where there exist clusters of nodes that have the same values. In such clusters nodes do not change their states, except for the nodes at the borders of the cluster that may switch back and forth between states of the bordering clusters. Figure 2.4 shows examples of such systems. Figure 2.4a



a) Synchronous GKL, system converges to the majority state

b) Synchronous SM, network evolves to stable clusters

c) Asynchronous SM, clusters disturbed by noise, system converges

Figure 2.4: State evolution in ring lattices. $N = 99, K = 3$.

presents a synchronous GKL that successfully solves the consensus by converging to the state that corresponds to initial majority. Figure 2.4b shows how SM consensus clusterizes the system into stable regions (clusters) of nodes having the same state, so that the system does not agree. Figure 2.4c shows SM consensus randomized by noise: noise dithers the clusters and promotes consensus.

Limitations

Randomization adds stochastic intrusions to the system and affects the convergence process. In particular, agreement itself becomes probabilistic [Asp03]. In a randomized consensus system, agreement can be guaranteed, even with presence of faulty nodes [BO83, AT12], but the exact moment of agreement cannot be predicted. This condition can complicate the applicability of randomization to the wait-free consensus algorithms. There have been proposed randomized wait-free consensus algorithms [Cha96, BR92] but these solutions are complex and only a few address the binary majority consensus problem. In this thesis we do not relax the connectivity and time restrictions of the wait-free systems. We focus on the simple algorithms for binary majority consensus because, despite the simplicity, they were shown to perform global coordination in restricted distributed systems.

2.6 Contributions and State of the Art

Let us briefly summarize how the contributions of our work complement preceding research. We have already mentioned that the binary majority consensus is a subclass of general consensus problem which is often investigated with restrictions in connectivity and execution time. Tables 2.2 and 2.3 show how our work complements the previous studies on binary majority consensus with disturbances.

Table 2.2: Binary majority consensus in various conditions.

Rule	System Size	Connectivity	Synchrony	ρ_0	Ref.
GKL	$N \geq 149$	$K = 3$	full	Gaussian, Uniform	[MHC93]
SM	$N \geq 149$	$K \geq 3$	partial	Gaussian	[MMDA04]
GKL	$N = 149$	$K = 3$	partial	Gaussian	[MMDA04]
TM	$N = 149$	$K = 3$	full	0.9	[Fat13]
Fukś	$N = 149$	$K = 3$	full	0.5	[Fuk97]
GKL, SM	$N \leq 149$	$K \leq 3$	none, full, partial,	Gaussian, Uniform	Chapter 4

Table 2.2 summarizes the studies on standard algorithms for binary majority consensus in different conditions, including varied system size N, number of neighbors K, synchrony level and distribution of initial density ρ_0. It shows that in our investigations we address the influence of conditions that were not covered previously, such as the influence of weak connectivity, small system size and noise. This

is motivated by a distributed decision making in noisy wireless sensor networks. Such systems are weakly connected and rarely exceed a few hundred nodes in size. The impact of additive noise on simple consensus algorithms was not widely studied due to the their initially low efficiency and robustness that can further deteriorate with noise.

Table 2.3: Binary majority consensus with disturbances.

	Disturbed by					
Rule	Noise	Errors	Mess. Loss	Topology	Faulty nodes	Ref.
GKL, SM	—	✓	—	Global, WS	—	[MMDA04]
GKL	✓	—	—	—	—	[GKL78]
GKL, SM	✓	—	✓	Global, WS, Waxman	Persistent, Byzantine	Chapters 4 and 6
RNM	✓	✓	✓	WS, Waxman, localized	Persistent, Byzantine	Chapters 5 and 6

Table 2.3 classifies research on randomizing impact of various disturbances, such as noise, errors, random topology and faulty nodes. It indicates that in our research we investigate the previously not covered impact of noise, errors and topology randomization. For the first time we extend this by studying the impact of random message loss and faulty nodes of different type, with a focus on their positive impact. Let us describe the disturbances considered in this work in more detail.

2.7 Randomization Schemes Considered in This Work

In this thesis we investigate the impact of randomization on binary majority consensus. We focus on randomization by topology and random neighbor selection, and disturbance by errors, noise, message loss. Let us briefly describe these techniques.

Randomization by Topology

Previous works [MMDA04, Bea05, KB06] mainly consider global topology randomization by the underlying network topology. In our work we also consider this type of randomization, where the implementation of the algorithm itself is adjusted to perform in both ordered and randomized networks, without "actual knowledge" of the underlying topology. Along with this scheme we implement a new local-scope topology randomization scheme by random neighbor selection, described in Section 2.4.4. In this scheme each node can randomly select C out of $2K$ neighbors provided by the network, without altering the underlying network topology and keeping the connectivity within the boundaries of the reference setup.

Randomization by Neighbor Selection

Let us briefly elaborate on topology randomization by neighbor selection. Randomization by neighbor selection is implemented in a way that leaves the underlying network topology intact and allows for logical topology changes at each time step that can inhibit network clustering. The network is generated once in the beginning of the simulation, and each node on the network is connected to its neighboring nodes with bidirectional links; these nodes form a list of its neighbors S. At each time step $t \in \{0, \ldots, T\}$ nodes can randomly select neighbors from the list $S, ||S|| = 2K$ to receive messages from. We introduce two random neighbor selection procedures:

- update-biased neighbor selection;
- uniform or "balanced" neighbor selection.

The motivation behind these schemes grounds in changing network topology that can influence the synchrony level. In the networks where a partial synchrony or an update sequence can be established, the first scheme can be used. The second one can be beneficial in random topologies, where the synchrony or the update sequence are disrupted. We explain these schemes in detail in Chapters 5 and 6. We then investigate the impact of random neighbor selection as a sole source of randomization and in combination with additional randomization by noise and errors.

Randomization by Noise

In this work we consider additive noise of two types, Additive White Gaussian Noise and Additive White Uniform Noise. We study influence of additive noise on SM, GKL, and RNM in Chapters 4, 5 and 6.

Randomization by Errors

Positive impact of randomization by errors was previously reported by Moreira *et al.* [MMDA04]. We investigate the impact of low levels of errors in addition to random neighbor selection in Chapter 5. This allows

Randomization by Message Loss

For the first time we simulate systems with stochastic message loss in Chapters 4, 5 and 6. We apply message loss independently and jointly with noise and errors to compare their sole impacts with the combined influence.

Noise, errors and message loss are present in most of the real-life networks, but are generally considered as a negative influence. Our studies can exhibit whether networked nodes in a consensus system can utilize the beneficial impact of these disturbances rather than allocate the scarce resources to eliminate it.

Randomization by Faulty nodes

A Byzantine faulty node with arbitrary failure can act as a noise generator. Moreira *et al.* [MMDA04] show that binary errors can promote consensus. For the first time we implement faulty nodes with a reduced state space which act as error generators, switching between possible states $\sigma \in \{-1, 1\}$ and $\sigma \in \{-1, 0, 1\}$. The positive and negative impact of faulty nodes is investigated in Chapter 6.

CHAPTER 3 System Modeling

3.1 Introduction and Motivation

Due to the distributed execution manner and restrictions in connectivity and time, wait-free consensus algorithms are difficult for analytic investigations. For this reason binary majority consensus is widely studied with use of computer simulations. Simulations are used to model a wide range of networked interactions but often suffer from incomplete modeling or missing assumptions [KCC05, PJJJS02].

There exist simulation engines and software libraries that are proven to be correct, and many can be found in open access. However, consensus simulations require numerous runs and the use of third-party non-optimized simulator engines can be time exhausting. Our simulation experience shows that an optimized self-made simulation engine, built in *C++*, can significantly reduce the simulation run time, compared with commonly used (e.g., Python) graph libraries. Apart from the low simulation performance third-party libraries still require significant development effort for particular experiments. Such a development has to account for predefined simulator features so that new features often are implemented in a non-optimal way. These restrictions motivated us to implement our own simulation engine.

In this chapter we explain modeling assumptions and the choice of software. We describe the networking model, messaging system, and updating functions. We elaborate on the system adjustments, implemented to accommodate both the system-wide and localized randomizing disturbances, such as noise, message loss, and asynchronous update. We illustrate the selected procedures and structures with pseudo-code listings. Our simulation engine is built using boost libraries (`www.boost.org`) after respective features of Python Networkx graph library (`networkx.lanl.gov`).

3.2 Modeling Assumptions

To exhibit various aspects of the behavior of consensus algorithms we simulate them in various conditions. In our simulations we reflect networks that can represent abstractions of real-life systems. The use of consensus algorithms is reported in multiple diverse areas, such as distributed database management [BG84], mission planning [AH06] and other computational problems [Bea05], which are described by different networking architectures. In addition to this, the development of the algorithms requires comparison to the “reference” network topology, initially introduced in [GKL78]. To study consensus algorithms in different networks we classify them into three subclasses:

- Ordered or regular networks, such grids and lattices;
- Random networks of social and natural origin, such as fully random or Small-World networks;
- Human-designed random networks, such as internet or wireless sensor networks.

We focus on simple self-organized consensus algorithms. Such algorithms account for communication between directly neighboring networked nodes, i.e., one-hop communications. This makes it possible to neglect the long-range network order in favor of a close-range and reduce the first class of networks to simplest lattice models.

Random networks of natural origin can have globally spanning links and exhibit scale-free properties. Such networks are often modeled with Small-World graphs that incorporate both of these features. To model such networks we use the Watts-Strogatz (WS) [WS98] network model that can produce networks, ranging from ring lattices to Small-World networks and fully random graphs. To describe the scale-free phenomenon and clustering levels, latter models are often characterized with a “path length” between two nodes that is combined of multi-hop connections. As mentioned above, simple consensus algorithms only account for single-hop communications, so we employ the term “link length/span”, described in Section 2.3, to characterize the network structure.

Human-designed random networks such as internet or wireless sensor networks can also have random links, but the probability of such a link strongly depends on a distance between nodes. Intuitively it is clear that if the links in, e.g., a wireless sensor network can span between most distant nodes, the network can easily become fully connected. Fully connected distributed networks can pose their own challenges that are out of scope of our research. In our paradigm fully connected networks are seldom due to the power and communication range constraints. To model not fully connected human-designed random networks we use Waxman [Wax88] networks which we define and describe in the following sections.

Randomizing disturbances that can be utlized as a source of beneficial randomization can be introduced in two ways [Asp03]. First way assumes that randomization is provided by a networked environment (e.g., communication noise), the second way assumes that randomization is implemented as a part of an algorithm itself (e.g., embedded random state switching). In our simulations we implement both approaches with independent superposition of noise, errors, and message loss. We describe this implementation in detail in Section 3.4.

3.3 Network Modeling

Let us start with network modeling. In our simulations we implement three types of networks:

1. $2K$-connected regular ring lattice;
2. $2K$-connected Watts-Strogatz graph;
3. $2K$-connected Waxman graph.

We use ring lattice network as an example of ordered networks and for comparison with reference performance figures.

To study the efficiency of consensus algorithms in abstractions of the real-world networks, such as natural [TInY+13] and technical systems [MM09], we employ two commonly used network topologies: Watts-Strogatz graphs [WS98] and Waxman graphs [Wax88], respectively. We model undirected networks that consist of N networked nodes. In such networks nodes are connected with bi-directional links. Nodes that share a link are called “neighbors”.

Nodes’ Neighbors and Degree

Let us elaborate on definition of nodes’ attributes and neighborhood. We define the link (or a connection) length of the node i to the node j as an absolute part of the difference between indices i and j. Considering, e.g., that a WS network is initially modeled as a ring lattice we refer to a maximum link span as $N/2$. For Waxman networks, however, link span is an actual Euclidean distance $d \in (0, 1]$, randomly assigned in the beginning of simulation. We model $2K$-connected ring lattices, WS, and Waxman networks to ensure that the average degree of the node in each of the networks is close to that of initial reference setup (see Section 3.3.1). Here and in the following we refer to a “node degree” as an average number of neighbors per node in the system. After initial network configuration neighbors of the each node compose three lists of neighbors. Let us describe them for ring lattices. All neighbors of the node i form the list of neighbors $S, ||S|| = 2K, S = \{i-K, \ldots, i-1, i+1, \ldots, i+K\}$. Neighbors $j \in \{S\}, j < K$ form the list of left-side neighbors $S_l, ||S_l|| = K$, while neighbors $j \in \{S\}, j > K$ form the list of right-side neighbors $S_r, ||S_r|| = K$. These lists are further used by algorithms to access the state information of node's neighbors.

Node Attributes

Each node in the network shares a set of properties with assigned values. Let us illustrate it with a reduced set of attributes of the non-faulty node $i = 1$ at the time $t = 0$, in a ring lattice:

- $node\ number = 1$
- $is\ faulty = false$
- $has\ random\ failure = false$
- $has\ full\ random\ failure = false$
- $current\ state = -1$
- $new\ state = 0$
- $neighbors\ list = \{98, 99, 0, 2, 3, 4\}$
- $left\ neighbors\ list = \{98, 99, 0\}$
- $right\ neighbors\ list = \{2, 3, 4\}$
- $messages\ received\ from = \{\}$
- $received\ state\ messages = \{\}$.

These attributes are used to implement the node's behavior (e.g., a faulty node with a random failure) in accordance with a given network model. Let us describe the network modeling and implementation in full detail.

3.3.1 Ring Lattice

To study the efficiency of the algorithms in a reference setup we implement a $2K$-connected ring lattice illustrated by Figure 3.1. Ring lattice of $N = 149$ nodes, each with $2K = 6$ neighbors represents a reference setup for performance comparison.

Such a lattice is initially modeled as a one-dimensional cellular automaton of N nodes, where each node is connected to its K following nodes. This lattice is then closed in a ring to avoid boundary effects. Such a setup was initially offered by Gacs *et al.* [GKL78], and later was adopted by multiple scholars for performance comparison of binary majority consensus algorithms.

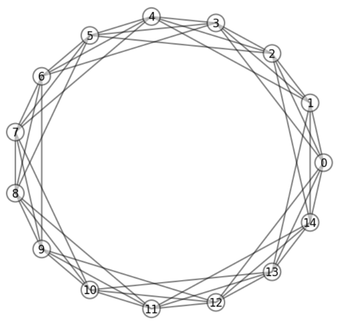


Figure 3.1: Ring lattice of $N = 15$ nodes with $K = 3$ neighbors on each side.

3.3.2 Watts-Strogatz Graph

A Watts-Strogatz (WS) graph can produce networks that incorporate the small-world phenomenon and the properties of scale-free topologies [BR99, WS98, Wol02]. It can be employed to represent social and natural networks [WS98, Yan01, MN04, NM05]. Investigation in such networks can exhibit the behavior of consensus algorithms in abstraction of natural random networks. It can also show whether stochastic topologies, intrinsic to such networks can be beneficial for distributed decision making.

Initially, the WS network is created as a $2K$-connected ring lattice, and at this point the network matches the reference setup. Next, every link of the node i to the node j is *rewired* with rewiring probability P, i.e., deleted and reassigned to a random node $z \notin \{i - K, \dots, i + K\}$. Varying P from 0 to 1 results in the transition from a regular ring lattice to a random graph: at $P = 0$, the network is a regular $2K$-connected ring lattice. At $P = 0.5$, it can be represented as a Small-World network, where $\simeq 50\,\%$ of links are random. At $P = 1$, all links are random and the network is a pure random graph.

Rewiring

We implement pairwise rewiring procedure as follows. For a node n_1 a regular link to a node n_2 is rewired to random nodes on the network $j, z \notin \{i - K, \dots, i + K\}$ in a following manner:

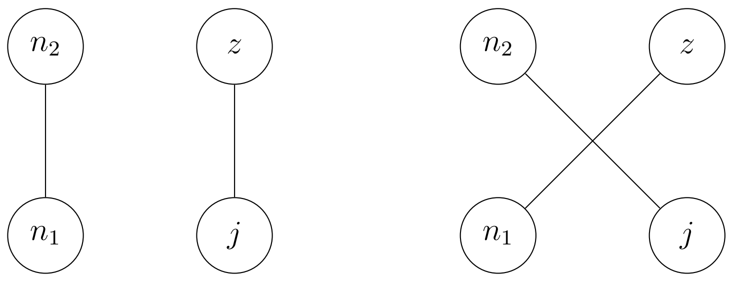
1. node's n_1 entry n_2 is substituted with j;
2. node's n_2 entry n_1 is substituted with z.

This pairwise rewiring procedure is illustrated by figure 3.2 Such a rewiring procedure increases the number of links that span for more than K indices, but preserves the average network connectivity. Dynamics of the link span in WS network with increasing rewiring probability P is shown in Figure 3.3.

Figure 3.3 shows that with growth of P the average link span (link length) increases from $\leq K$ to $\simeq \frac{N}{3}$. Thus, growth of P leads to increase of system interconnectivity, when more nodes have links spanning through the whole system. This transition is illustrated in Figure 3.4. Interconnectivity of the network is an important property for the distributed consensus, as globally spanning links can significantly increase the efficiency of consensus algorithms.

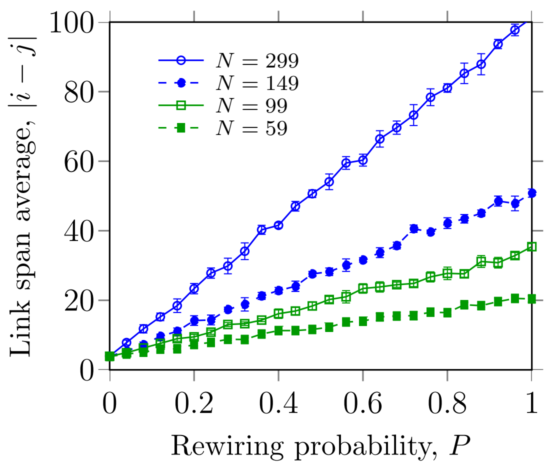
WS graph generation in our simulator is provided by the procedure *GenerateConnectedWS*(), illustrated by the following code. This procedure is implemented after the [WS98] and the respective graph generator from Networkx libraries.

```
void GenerateConnectedWS
(int nodes_number, int neighbors, float rewiring, int tries){
  int tries_counter=0;
  bool graphDone=false;
  cout<<"WS started"<<"\n";
  while(tries_counter<tries && !graphDone){
    nodes_list.Clear();
//Fill in the links, as a ring lattice
    for (int i=0;i<nodes_number;i++){
      for (int k=1;k<=neighbors;k++){
        PutEdge(i,(i+k)%nodes_list.size());
        }
      }
//Rewiring
    for(int i=0;i<nodes_list.size();i++){
      for(int k=0;k<nodes_list[i].neighbors_list.size();k++){
        if (CheckEdge(i,nodes_list[i].neighbors_list[k])){
          RewireEdges(i,nodes_list[i].neighbors_list[k],rewiring);
          }
        }
      }
//Neighbors' lists fill in
    for(int i=0;i<nodes_list.size();i++){
      SortNeighbors(i, neighbors);
      }
    tries_counter++;
    graphDone=CheckNeighbors(neighbors);
  }
  if (graphDone){
    cout<<"WS done"<<endl;
    }
  else {
    cout<<"WS NOT done, tries exceeded"<<endl;
    }
  AverageDegree();
}
```

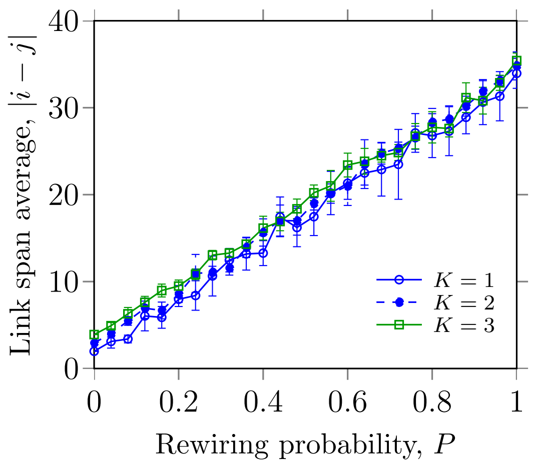


a) Regular links   b) Rewired links

Figure 3.2: Pairwise rewiring of the nodes n_1 and n_2.

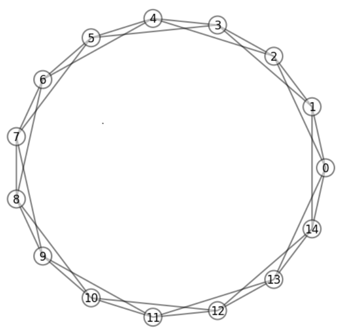


a) Networks with $K = 3$

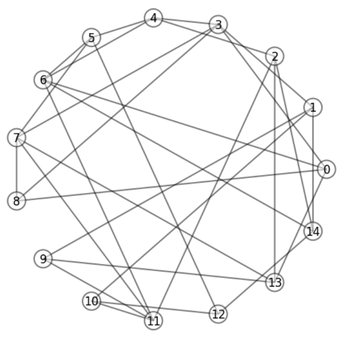


b) Networks of $N = 99$

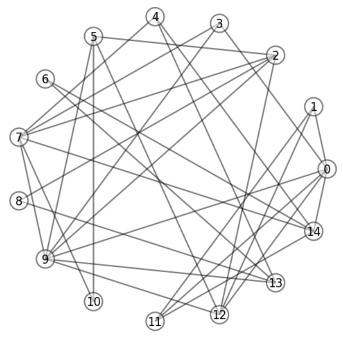
Figure 3.3: Average link span in WS networks with growing P. Networks of different size with different number of neighbors.



a) $P = 0$, ring lattice



b) $P = 0.5$, a Small-World network



c) $P = 1$, random graph

Figure 3.4: WS network with growing P, $N = 15, K = 2$.

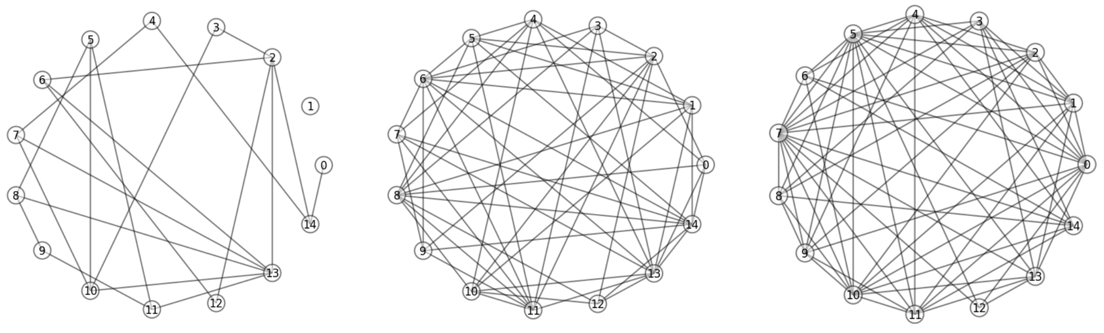
3.3.3 Waxman Graph

Graphs proposed by Bernard Waxman [Wax88] can be used to model human-designed random networks, such as the Internet [Van01]. Simulations in such networks can exhibit the differences in behavior of consensus algorithms between natural random networks and human-designed random networks and provide insights on design of distributed decision-making algorithms.

Unlike the WS model that can produce networks, ranging from a ring lattice to a fully random graph, Waxman graph is an initially random network where the probability of a link depends on a distance between nodes. A graph is built as follows. First, for each pair of nodes $i, j \in \{1, 2, \ldots, N\}, i \neq j$, the distance d is randomly uniformly chosen from the interval $(0, 1]$. Next, the nodes are linked with probability

$$\alpha \exp\left(-\frac{d}{\beta}\right) , \tag{3.1}$$

with parameters $\alpha, \beta \in (0, 1]$. Parameters α and β influence the network as follows. An increasing α yields an increasing link probability, thus increasing the average node degree. An increasing β increases the number of the long links compared to the short links, thus increasing the average link length in the network. For comparison purposes we model sparsely connected Waxman graphs with fixed $\alpha = 0.05$ and $\beta \in [0.01, 0.4]$.

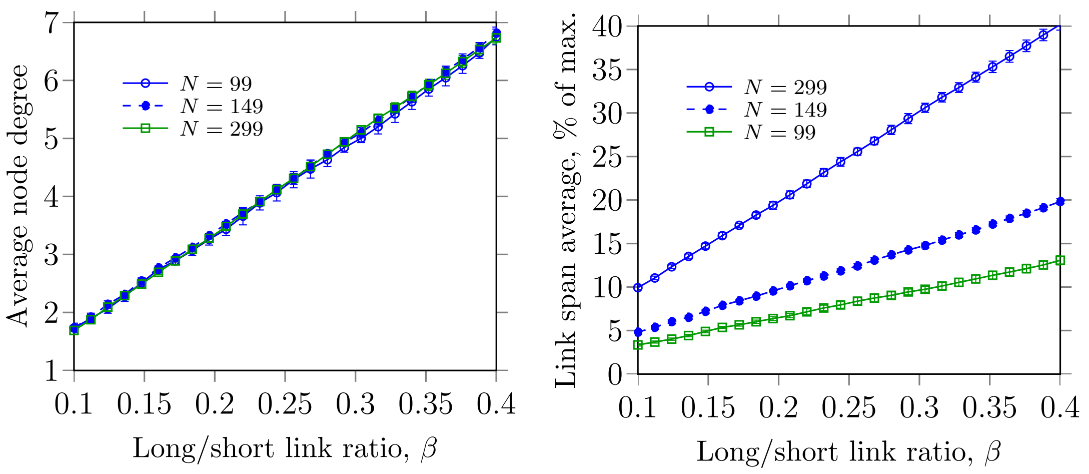


a) $\alpha = 0.5$, $\beta = 0.5$ b) $\alpha = 1$, $\beta = 0.5$ c) $\alpha = 1$, $\beta = 1$

Figure 3.5: Waxman graph of $N = 15$ nodes with $\alpha, \beta \in \{0.5, 1\}$

Within the given parameter range of α and β, we artificially limit the average node degree and average link length to match that of the WS model (see Figure 3.3). Dynamics of the node degree in a Waxman network of $N = 99$ nodes, depending on β with fixed $\alpha = 0.05$, is shown in the Figure 3.6

The Waxman network is generated by procedure *GenerateConnectedWaxman()*, which is illustrated by the following pseudo code.



a) Average node degree

b) Average link length

Figure 3.6: Nodes connections in Waxman networks of different size with growing β, $\alpha = 0.1$

```
bool graph:: GenerateConnectedWaxman
(int nodes_list, float alpha_val, float beta_val, int tries){
  int tries_counter=0;
  bool IsConnected=false;
  while(!IsConnected && tries_counter<tries){
    nodes_list.Clear();
    distance_matrix=FillDistances();
  for (int i=0;i<nodes_list.size();i++){
    for (int k=0;k<nodes_list.size();k++){
      if(i!=k){
        ConnectNodes(i,k, alpha_val, beta_val);
        }
      }
    }
  IsConnected=LimitNodeDegreeToWS();
  tries_counter++;
  }
  if(!IsConnected){cout<<"WM_Not_Done"<<endl;}
  else{cout<<"WM_Done,_"<<" Tries="<<tries<<endl;}
  WaxNbSort();
  WaxAvDegree();
  return IsConnected;
}
```

This procedure returns a Waxman graph of the second type, where the distance between each pair of nodes $\{i, j\}$ is randomly chosen between 0 and 1. It is implemented after the [Wax88] and the respective graph generator from Networkx libraries.

Our simulations show that with starting conditions $\alpha = 0.05$ and $\beta = 0.04$, Waxman networks of, e.g., $N = 99$ nodes are not guaranteed to be connected. In this case, node's i sets of neighbors S_l and S_r can be empty. These conditions make Waxman graphs difficult for consensus algorithms like GKL. State-direction bias of GKL requires that a network structure is known, and "left" and "right" directions are predefined. If this condition is not met, GKL efficiency can degrade from $\simeq 82\%$ to $\simeq 1\%$.

3.3.4 Synchrony and Update Modes

As we explain in Section 2.4.2, the synchrony level in consensus studies generally accounts for "phase shifts" between nodes, but assumes that the decision frequencies (or the period between consequent decisions for each node) are equal for all nodes. Under this assumption, in our simulations we implement three types of update modes:

- synchronous;
- asynchronous sequential;
- fully asynchronous.

In the synchronous mode all nodes calculate the *newState* and then simultaneously update their *currentState*. In the sequential asynchronous update mode nodes are updated according to their indices, e.g., $0 \to N$ or $N \to 0$. Update sequence is generated once in the beginning of the simulation run. In the fully asynchronous mode nodes are updated randomly, independent from each other and the update sequence is also generated once in the beginning of simulation. Updates are implemented at a node level via the *Update*() function illustrated by the following pseudo-code:

```
void node:: Update (){
        if(!faulty){
//If the node is not faulty
                if(newState<0)          {currentState=-1;}
                else if(newState>0)     {currentState=1;}
        }
        else{
                if(randomFailure && !randomExtended){
//Two-state faulty node changes its Stateue randomly in{-1,1}
                currentState= 2*(rand()%2) - 1;
                }
                else if(randomFailure && randomExtended){
//Three-state faulty node
                currentState= (rand()%3) - 1;
                }
        }
//If faulty persistently - do not update, keep faulty state
```

```
        receivedOld=receivedStates;
        receivedFromOld=receivedFrom;
        receivedStates.clear();
}
```

One can see how the nodes attributes alter behavior of the update function. These alterations implement faulty node behavior, in accordance to the respective failure model (for further details see Section 3.5). In this thesis we present simulation results for *fully synchronous* and *asynchronous sequential* updates, unless mentioned otherwise.

3.4 Message Exchange and Randomization

In our model a node i can access state information of node j via state information message $\sigma_{i,j}$. Message exchange is organized via procedure *GetState()*. This procedure takes the current state of the *senderNode* (node j), changes it according to the level of system noise, errors, or probability of message loss and puts it into a receiving buffer *received* of the *receiverNode* (node i). Procedure is illustrated by the following pseudo-code:

```
void graph:: GetState(node receverNode, node senderNode){
 float ReceivedState;
 float MesslossSample=(float)rand()/RAND_MAX;
 float ErrorSample=(float)rand()/RAND_MAX;
 float NoiseSample=(float)rand()/RAND_MAX;

//Message is lost?
  if(MesslossSample<MessageLossProbability){return;}
  else{
//If message is not lost
  ReceivedState=SenderNode.currentState;

//Erroneous State Message
  if(ErrorSample<ErrorProbaility){
    ReceivedState=-SenderNode.currentState;
    }
  else{
    ReceivedState=  SenderNode.currentState;
    }

  if (NoiseSample<NoiseProbability){
//Additive White Uniform Noise
    if(UniformNoise){
      ReceivedState = SenderNode.currentState +
      UniformFloat(NoiseAmplitude);
      }
//Additive White Gaussian Noise
```

```
    if(GaussianNoise){
      ReceivedState = SenderNode.currentState +
      NormalFloat(NoiseAmplitude);
      }
   }
//Zero noise - receive correct state
   ReceiverNode.receivedState.push_back(ReceivedState);
   ReceiverNode.receivedFrom.push_back(SenderNode);
   }
}
```

This listing shows that with randomization by noise, message loss and errors, a received state message $\sigma_{i,j}$ can be corrupted. In a disturbance-free case, correct message is received. In a similar way the node i own state $\sigma_{i,i}$ can be corrupted. Such a scheme allows us to investigate whether explicit stochastic disturbances, intrinsic to natural networked systems can positively influence the simple consensus algorithms. In our simulations we focus on noise, errors, and message loss to investigate whether such disturbances could trigger an increase in convergence.

3.4.1 Randomization by Additive Noise

In the reference system [GKL78, MMDA04] node i receives state information from the node j via the state information message $\sigma_{i,j}[t]$. Procedure *GetState*() implements noise added to a received state information by the following transformation:

$$\sigma_{i,j}[t] \to \sigma_{i,j}[t] + \phi_{i,j} \; . \tag{3.2}$$

Here, the random value $\phi_{i,j}$ is a sample of noise. We implement two types of noise: Additive White Gaussian Noise (AWGN) where $\phi_{i,j} \sim \mathcal{N}(0, (\frac{A}{3})^2)$, and Additive White Uniform Noise (AWUN) where $\phi_{i,j} \sim \mathcal{U}(-A, A)$, with the magnitude $A \in [0, 4]$. The range for the noise amplitude is selected empirically to account for both positive and negative influences.

Previous studies mostly consider AWGN as the most common noise type in real networks [KM07, CFF^{+}07], and AWUN is generally used to model response of filters and amplifiers [VD02].

3.4.2 Randomization by Message Loss

To model message loss we employ a simple probabilistic model where a state information is lost with the probability $\mathcal{E} \in [0, 1)$, i.e.:

$$\sigma_{i,j}[t] \to \begin{cases} \sigma_{i,j}[t], \text{ with probability } (1-\mathcal{E}) \\ 0, \text{ with probability } \mathcal{E} \end{cases} \; . \tag{3.3}$$

3.4.3 Randomization by Errors

Positive impact of randomization by errors was reported by Moreira *et al.* [MMDA04]. We consider errors as additional source of randomization in Chapter 5. Errors are modeled as a probability of receiving the wrong state information:

$$\sigma_{i,j}[t] \to \begin{cases} \sigma_{i,j}[t] \text{ with probability } (1-\eta) \\ -\sigma_{i,j}[t] \text{ with probability } \eta \end{cases} . \tag{3.4}$$

In Chapter 5, we empirically select a narrow interval $\eta \in [0.05, 0.08]$ that provides an optimal level of randomization for the investigated randomized consensus.

3.4.4 Scope of the Randomization

We apply randomizing disturbances in two schemes where the node's own state information is: (a) disturbed by noise and message loss or (b) free of randomizing disturbances. In the first scheme σ_i is influenced by noise, errors, and message loss, i.e., $\sigma_i \neq \sigma_{i,i}$. This case corresponds to the scenario of distributed detection where the network is expected to agree on a binary state (e.g., whether the detected event took place or not), while each node has noisy inputs from unreliable sensors. In such a scenario, the sensing subsystem is "decoupled" from decision-making subsystem of the networked node.

In the second scheme, $\sigma_{i,i}$ is free of randomizing disturbances. In this scenario noise and message loss present node-to-node communication disturbances that do not affect the node's i own state.

To allow for multiple combinations of influences we apply noise, errors and message loss independently from each other. In such a model, a node can loose a meaningful state message, but still receive a noise sample, or receive an erroneous message additionally affected by noise. Independent influence is a crucial requirement for investigating the combined influence of randomizing disturbances.

3.5 Faulty Nodes

To investigate the robustness and fault tolerance of the consensus algorithms we implement faulty nodes as a special type of nodes that counter the consensus. At a starting time $t = 0$, M nodes are added to N non-faulty nodes, to avoid bias of the initial configuration. Network topology is then created for all $N + M$ nodes. After adding M faulty nodes to the system, they are labeled as faulty. These nodes counter consensus according to their failure model. Let us describe the faulty node attributes for, e.g., a faulty node with persistent failure model in a network with $\rho_0 > 0$:

- $is\ faulty = true$ (updates independently from received state messages);
- $randomly\ failing = false$ (no random failure, $\sigma_M \notin \{-1, 1\}$);
- $extended\ random\ failure = false$ (no random and no full failure, $\sigma_M \notin \{-1, 0, 1\}$).
- $current\ state = -1$ (opposite to the ρ_0)

These attributes alter the node's $Update()$ function behavior, as we describe in the following sections.

Research on the fault tolerance generally considers faulty nodes with Byzantine failure model. Byzantine faulty node is a node that can have arbitrary failures, except full or crash failures [Asp03]. We consider two failure models for faulty nodes: (a) faulty nodes with random failure and (b) faulty nodes with persistent failure. In the two following sections we describe their modeling and corresponding adjustments to the system.

3.5.1 Faulty Nodes with Random Failure

Faulty nodes with random failure ("non-persistent faulty nodes") are implemented after Byzantine random failure with a reduced state space. Such nodes change their broadcasted state randomly, independently from the state information received from their neighbors.

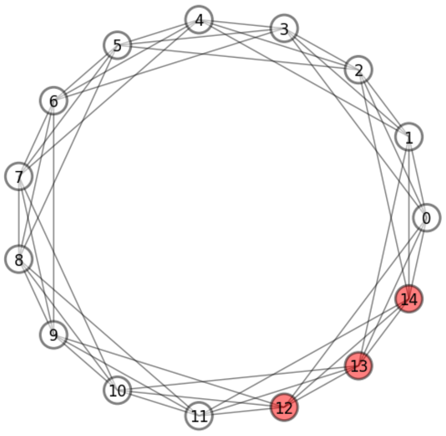
In this thesis we investigate two types of randomly failing faulty nodes:

- two-state faulty nodes with random failure states $\sigma_M \in \{-1, 1\}$;
- three-state faulty nodes with random failure states $\sigma_M \in \{-1, 0, 1\}$.

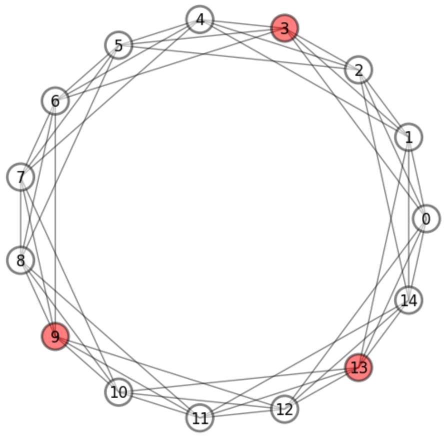
First case presents a faulty node that broadcasts correct and erroneous state information with equal probabilities. Second case additionally implements a state of sending no information, i.e., full or crash failure.

3.5.2 Faulty Nodes with Persistent Failure

Faulty nodes with persistent failure ("persistent faulty nodes") are modeled as follows. After M nodes are added to the system and labeled as faulty, they are assigned with a faulty value σ_M, opposite to the initial majority. I.e., if $\sum_{i=0}^{i=N} \sigma_i[0] < 0, \sigma_M = 1$ and if $\sum_{i=0}^{i=N} \sigma_i[0] > 0, \sigma_M = -1$. During consensus process such faulty nodes broadcast their state but do not update it. Unlike faulty nodes with random failure, faulty nodes with persistent failure provide enduring inhibition for consensus.



a) Clustered faulty nodes



b) Distributed faulty nodes

Figure 3.7: WS network with clustered and distributed faulty nodes.

3.5.3 Layout of Faulty Nodes Over the Network

We implement two schemes of faulty node layout over the network, illustrated in Figure 3.7: clustered and distributed. With the clustered layout all faulty nodes are located next to each other, and the location of the cluster is randomly chosen at each simulation run. With a distributed layout all faulty nodes are randomly placed over the network independently from each other. We investigate consensus excluding faulty nodes, where only N non-faulty nodes are expected to reach an agreement.

3.6 Consensus Termination

We consider a network to reach an agreement if all N non-faulty nodes have agreed on the same state within a certain time period T, and this final state corresponds to the initial majority of values [GKL78]. In other words, the system converges if (a) $\sum_i \sigma_i[0] < 0$ and there exists $t_c \leq T$ so that $\sigma_i[t_c] = -1, \forall\ i$, or if (b) $\sum_i \sigma_i[0] > 0$ and there exists $t_c \leq T$ so that $\sigma_i[t_c] = 1, \forall\ i$. If this condition is met, the algorithm is terminated. Algorithms are always terminated after time T whether agreement has been reached or not ("wait-free" consensus). Time is divided into steps of equal size Δ. Without loss of generality, we assume that the duration of one time step Δ equals to one unit of time. The time steps are indexed by $t = \{0, 1, 2, \ldots, T\}$ for simplicity of notation. We use $T = 2N$ as initially proposed by Gacs *et al.* [GKL78]. The termination condition is verified with a function *CheckConvergence()* illustrated by the following pseudo-code:

```
bool CheckConvergence(){
  bool Converged=false;
  step_density=0;

  for (int i=0;i<nodes_list.size();i++){
    if( !nodes_list[i].faulty){
      step_density += nodes_list[i].currentState;
      active_nodes_list++;
      }
    }
  }
  if((step_density ==  active_nodes) && (Initial_density >0)){
    Converged=true;
    }
  else if ((step_density == -active_nodes) && (Initial_density <0)){
    Converged=true;
    }
  else if ((active_nodes - step_density)!=0){
    Converged=false;
    }
  else{
    Converged=false;
    }
return Converged;
}
```

3.7 Initial Configurations

At the beginning of each simulation run, every node $i \in \{1, \ldots, N\}$ is assigned with binary state $\sigma_i[0] \in \{-1, 1\}$. A set of $\sigma_i[0]$ is called *initial configuration*, and denoted as I. The sum of all initial states $\sum_{i=0}^{i=N} \sigma_i[0]$ is called the *initial density* and is denoted as ρ_0. Each state σ_0 in the initial configuration is acquired by a coin-flip trial, returning -1 and 1 with equal probability.

Some algorithms can be sensitive to initial configurations with the same density but different permutations. Due to this fact, some scholars argue that sufficient diversity of initial configurations is more important than the ρ_0 distribution among them. They simulate test sets combined of initial configurations with uniform density distribution [MHC93]. We implement such test sets as well, for comparison purposes.

Test sets are generated preliminary to the simulation and stored in separate files, each containing 10.000 initial configurations. This simple division enables to run parallel threads to reduce the simulation time. Test sets of I with uniform distribution of ρ_0 are obtained with the Python Numpy function *numpy.random.uniform()*. Test sets generated by a coin-flip procedure are obtained with Python function *random.randint()*. In the following chapters we present performance figures registered over test sets obtained by a flip-coin trial, unless mentioned otherwise.

3.8 Summary

Due to the distributed execution manner and restrictions in connectivity, synchrony, and time consensus algorithms are difficult for analytic investigations and often are studied be means of computer simulations. Computer simulations of consensus algorithms can be time-exhausting. The use of third-party simulation libraries and engines can save the development effort, but later it can result in a longer simulation time. Moreover, such third-party simulation engines often have limited available features and require a significant effort to extend the functionality. For these reasons we develop our own simulation engine. It is built in *C++* using Boost libraries (`www.boost.org`) after the original models presented in [GKL78, Wax88, WS98, Wat99]. The simulation engine is tested with the respective features and graph generators of Networkx library (`networkx.lanl.gov`).

In this chapter we explain motivation for the network modeling and the choice of software. We explain the important network features, attributes of the networked nodes, and messaging system. We describe and explain the character of randomizing disturbances and faulty nodes. Finally we illustrate the implementation process for the selected simulator features.

CHAPTER 4 Standard Algorithms with Randomization

4.1 Introduction and Motivation

Consensus algorithms can be used in distributed systems where centralized decision making is difficult or impossible. Such conditions can arise in distributed detection and tracking [GC96], database management [BG84, Kum91, Tho79], or mission planning [AH06].

Algorithms that perform such coordination in real-life networked systems should be efficient and robust towards different types of faults. In the last decades several new algorithms have expanded boundaries of consensus performance. We describe these algorithms and the approaches that can increase the performance of the consensus in detail in Section 2.4.6. *Randomization* is one of the approaches that utilizes random disturbances to improve both performance and robustness of consensus algorithms. Randomization is mainly used to increase the robustness towards faulty node behavior. It is generally embedded in relatively complex multi-step algorithms [Asp03]. The beneficial impact of randomization on simple consensus algorithms has only been studied recently [MMDA04, Fat13].

Binary majority consensus, described in Section 2.2, is a subclass of simple consensus algorithms that steer a system with initial random binary states to a common state that corresponds to the initial majority of state distribution. It has more strict termination conditions than general binary consensus and thus can be more sensitive to disturbances.

In Section 2.4 we describe recent studies on binary majority consensus with random disturbances that increase the efficiency of algorithms. In particular, Moreira *et al.* [MMDA04] study binary majority consensus in Watts-Strogatz (WS) networks with errors. They show that with randomization by errors in WS networks Simple Majority (SM) consensus achieves efficiency of $\geq 85\%$, while Gacks-Kurdyumov-Levin (GKL) consensus (known among the best algorithms) generally degrades. A recent paper by Fates [Fat13] shows that another binary majority consensus algorithm, namely Traffic Majority (TM) can achieve an efficiency of 90% with randomization.

SM, TM and GKL are wait-free consensus algorithms, i.e., they are terminated after a certain time T, whether agreement was reached or not. Such algorithms, capable of steering a distributed system to a required state in a fixed period of time, can be beneficial in networked control systems.

These results motivated us to investigate binary majority consensus with a wider set of disturbances to determine how randomization affects their efficiency. In this chapter we study SM and GKL in WS and Waxman networks with randomization by noise, message loss, and topology. Main contributions of this chapter can be summarized as follows:

- Randomizing disturbances by noise, message loss and topology can provide different types of cluster-breaking influence, beneficial for simple consensus algorithms;
- These disturbances indicate a cumulative beneficial effect if applied jointly;
- An optimal combination of such disturbances can significantly promote SM consensus, even in asynchronous networks.

We show that these new effects are evident in ring lattices and random and asynchronous Watts-Strogatz and Waxman networks. Next, we illustrate them with examples of system evolution and explain the underlying mechanisms. Some of the results presented in this chapter were published in [1, 5].

4.2 System Model

In this chapter we study SM and GKL in ordered and topologically randomized networks with additive noise and message loss. The efficiency of an algorithm for the binary majority consensus is generally measured in a "reference" setup that we describe in Chapter 3.

To measure the efficiency of the algorithms in randomized environments with disturbances, we adjust the reference networking model. In the following we briefly describe the implemented adjustments.

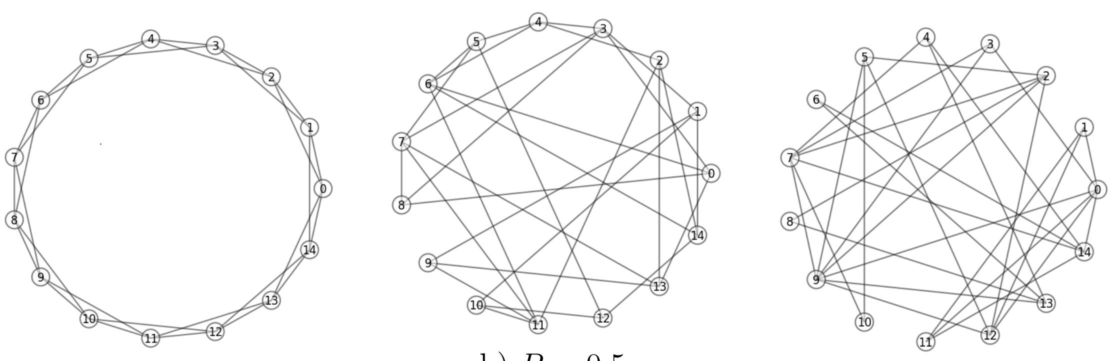
We study consensus algorithms in the following types of networks:

- Ring lattices as an example of ordered networks;
- Watts-Strogatz random networks;
- Waxman random networks.

Ordered networks are implemented as a one-dimensional ring lattice (reference setup), described in Section 3.3.2. Watts-Strogatz ([WS98]) and Waxman ([Wax88]) graphs are described in detail in Sections 3.3.2 and 3.3.3, respectively.

4.2.1 Algorithm Modification for Random Networks

The transition from a ring lattice to a random network should not affect the algorithm implementation. Let us consider a transition from a ring lattice to, e.g., a WS random network. Recall that the WS model can produce networks ranging from a ring lattice to a random graph with rewiring probability $P \in [0, 1]$. Figure 4.1 illustrates a network of $N = 15$ nodes, each with $2K = 4$ neighbors, with different rewiring probabilities P.



a) $P = 0$, ring lattice   b) $P = 0.5$, Small-World   c) $P = 1$, random graph

Figure 4.1: WS network of $N = 15$ nodes with $K = 2$ neighbors on each side.

We implement the transition from a ring lattice to the random network by the following transformation. Instead of referencing the closest neighbors of a node i by their indices (i.e., $i-1$ or $i+1$ as allowed in ring lattices) we reference neighbors by lists of the node's neighbors: list of all available neighbors S, list of left-side neighbors S_l, and list of right-side neighbors S_r. In the ring lattice such an adjustment is fully reversible: a neighbor index is directly correlated to a neighboring node's actual number, e.g. a third neighbor to the right r_3 is the actual third node to the right, $i + 3$. In random networks, however, the situation is different. Let us describe it for Watts-Strogatz network. In a Watts-Strogatz network lists of neighbors are defined at the initial stage of the network configuration in accordance to the neighbors' indices: $S = \{i - K, \dots, i + K\}$, $S_l = \{i - K, \dots, i - 1\}$, $S_r = \{i + 1, \dots, i + K\}$. Next, links to these neighbors are randomized according to the WS topology (see Section 3.3.2), but the referencing remains the same, e.g., the third neighbor to the right from the node i (in a ring lattice, node $i - 3$) becomes the third element of the list of right-side neighbors $S_r - r_3$, which in WS network can be a random node, i.e., not always $r_3 = i + 3$. These lists are further used by consensus algorithms to access the respective neighbors. A similar transformation is implemented for Waxman networks.

Simple Majority Consensus

To study SM consensus in random networks with message loss and noise, the algorithm should be adjusted to match the new system model. We modify SM, defined in Section 2.1, using the state information messages $\sigma_{i,j}$ randomized by noise and message loss, and a list of node i's neighbors S:

$$\sigma_i[t+1] = G\left(\sigma_{i,i}[t] + \sum_{j \in S} \sigma_{i,j}[t]\right) . \tag{4.1}$$

The update function $G(x)$ remains as defined in Section 2.2:

$$G(x) = \begin{cases} -1 & \text{for } x < 0, \\ +1 & \text{for } x > 0. \end{cases} \tag{4.2}$$

SM consensus is arguably the simplest algorithm for binary majority sorting, and has a balanced design: in ring lattices each node i receives an equal number of messages from both sides of the lattice.

Gacs-Kurdyumov-Levin Consensus

GKL consensus is adjusted to perform in randomized networks as follows. We substitute the neighbors $i-1$ and $i-3$ with first and third neighbors l_1 and l_3 from the list of left-side neighbors of the node i, S_l. A similar transformation is implemented for right-side neighbors:

$$\sigma_i[t+1] = \begin{cases} G\Big(\sigma_{i,i}[t] + \sigma_{i,l_1}[t] + \sigma_{i,l_3}[t]\Big) & \text{for } \sigma_{i,i}[t] < 0, \\ G\Big(\sigma_{i,i}[t] + \sigma_{i,r_1}[t] + \sigma_{i,r_3}[t]\Big) & \text{for } \sigma_{i,i}[t] > 0. \end{cases} \tag{4.3}$$

In GKL, each node chooses from which side to receive messages on the basis of its own current state. It enables GKL to "wash out" clusters of nodes having the same states. This bias provides for high efficiency of GKL in ring lattices but it can lead to low efficiency if the network structure or an update sequence is disturbed, as we will show in Section 4.3.

The transition from a ring lattice to WS and Waxman networks preserves the algorithm function and allows to randomize the underlying network topology. Such a topological randomization can affect the update sequence of both algorithms and the built-in direction bias of GKL.

4.2.2 Randomization by Noise and Message Loss

Additive Noise

To introduce noise, we modify the reference system as follows. In the reference system the node i receives state information from the node j at time t via the state information message $\sigma_{i,j}[t]$. We implement noise added to a received state information as described in Section 3.4.1.

We implement two types of noise: Additive White Gaussian Noise (AWGN) where $\phi_{i,j} \sim \mathcal{N}(0, (\frac{A}{3})^2)$, and Additive White Uniform Noise (AWUN) where $\phi_{i,j} \sim \mathcal{U}(-A, A)$, with the magnitude $A \in [0, 4]$.

Previous studies mostly consider AWGN as the most common noise type in real networks [KM07, CFF$^+$07], and AWUN is generally used to model the response of filters and amplifiers [VD02].

Message Loss

Message loss can severely decrease the performance of the studied algorithms, since a node decision is based on state information received from other nodes. If a message is lost, a node can come to a state when the sum of all received state messages $\sum_S \sigma_{i,j} = 0$ and the state of the node stays unchanged. In our simulations we implement message loss as described in Section 3.4.2.

4.2.3 Scope of Randomization

We study randomizing disturbances in two schemes. In the first scheme the own state of the node σ_i is influenced by noise and message loss, i.e., $\sigma_i \neq \sigma_{i,i}$. This case corresponds to the scenario of distributed detection where the nodes are expected to agree whether the detected event took place or not, based on noisy inputs from unreliable sensors. In such a scenario, the sensing subsystem is "decoupled" from the decision-making subsystem of the networked node.

In the second scheme, σ_i is free of randomizing disturbances. In this scenario noise and message loss represent node-to-node communication disturbances that do not affect the node i's own state.

4.2.4 Initial Configurations and Update Modes

Recall that a system is expected to converge to a binary state given a set of N initial random binary states called *initial configuration*. We use initial configurations obtained by a coin-flip trial. In addition to this, for comparison purposes we also use initial configurations with $\rho \sim \mathcal{U}(-N, N)$ (for further details on initial configurations refer to Section 2.4.6).

Update Modes

System-wide synchrony is important for a consensus process [DDS87]. We simulate systems with synchronous and asynchronous updates. In the synchronous mode, all nodes update their states simultaneously. In the asynchronous sequential mode, nodes are updated sequentially, one after another, according to their indices. In the fully asynchronous mode, nodes are updated randomly, independent from their indices. Random update sequence is generated once in the beginning of simulation. To update its state, a node uses the latest available states of its neighbors. Such distributed state updates can provide network randomization in time (for more details on system synchrony refer to Sections 3.3.4 and 2.4.2).

4.3 Performance Analysis

Performance Metrics

We use performance metrics as described in Section 2.3. For each set of parameters we generate three random networks. Each network is then simulated over 30 different sets of initial configurations (each set combined of 10.000 initial configurations). The resulting 90 values of R and F are then averaged and plotted with 95% confidence intervals.

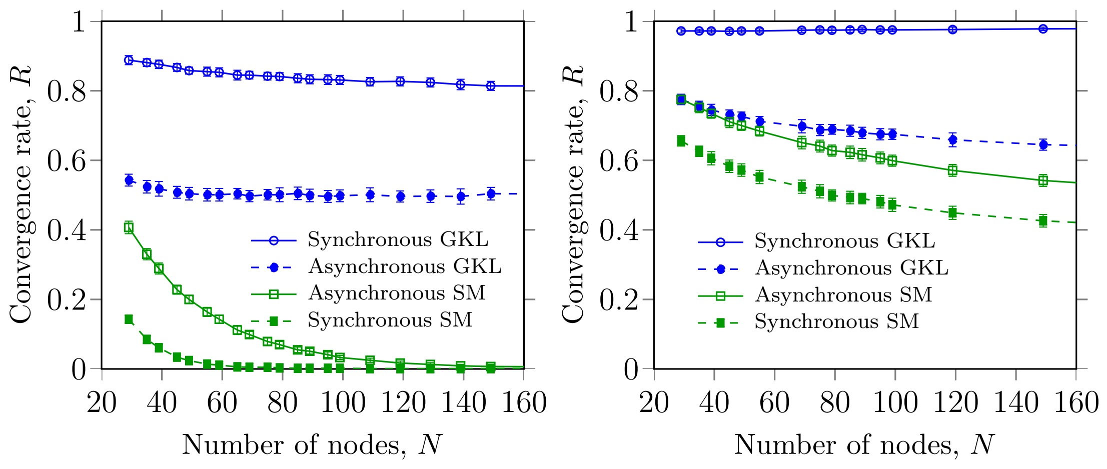
In the following sections we analyze the impact of:

- Different initial configurations (obtained by a flip-coin trial and with $\rho \sim \mathcal{U}(-N, N)$);
- System size (growing N with constant K);
- Different number of neighbors (growing K with constant N);
- Additive noise (increasing A);
- Message loss (increasing E);
- Topology randomization (increasing P and β);
- Combined impact of randomization by topology, noise and message loss.

In the next section we address the influence of initial configurations and system size on convergence rate of SM and GKL. In Sections 4.3.1, 4.3.3, 4.3.4 we investigate effects of sole randomization by additive noise, message loss and topology, respectively. Then, we study the combined randomizing impact of these disturbances in Sections 4.3.5 and 4.3.6. The efficiency in random Waxman networks with noise and message loss is addressed in Section 4.3.7. Section 4.3.8 investigates how noise and message loss affect the convergence rate of GKL and SM with different scope of randomization. The influence of randomization on convergence speed is studied in Section 4.3.9. Finally, Section 4.4 concludes the chapter.

4.3.1 Impact of Initial Configurations and System Size

Certain algorithms can converge with different R on two initial configurations with the same density but different permutations. Due to this, some scholars study binary consensus with different initial configurations that can result in different convergence rate [MCH94].



a) Test sets obtained by a coin-flip trial b) Test sets with $\rho \sim \mathcal{U}(-N, N)$

Figure 4.2: Efficiency of SM and GKL in ring lattices of different size.

Binary majority consensus has been previously studied in relatively large systems. Thus, Moreira *et al.* [MMDA04] and Mitchell *et al.* [MHC93] study binary majority consensus in networks of $N \geq 149$ nodes, showing that its efficiency changes with growth of the system. In this thesis we approach the problem of distributed decision making from a perspective of wireless sensor networks that often consist of a hundred, or even fewer, nodes. This motivated us to study consensus in networks of $N \leq 149$ nodes. Figure 4.2 shows that consensus algorithms can significantly change efficiency in such relatively small systems. It presents R of SM and GKL in networks of $N \in \{29, \ldots, 160\}$ nodes and synchronous and asynchronous updates. Figure 4.2a presents convergence rate with initial configurations obtained by a coin-flip trial, and Figure 4.2b presents convergence rates for test sets with $\rho \sim \mathcal{U}(-N, N)$. Their comparison shows that test sets obtained by a flip-coin trial result in lower convergence rate for both algorithms. It also shows that all algorithms, except synchronous GKL, degrade in efficiency with larger N and that SM degrades stronger than GKL.

The decrease in efficiency with growing N can be related to the fact that systems with larger ratio $\frac{N}{K}$ can be more prone to clustering [BR99]. The built-in direction bias of GKL is designed to dither such clusters [GKL78], which explains the higher R of GKL that is independent from the system size.

Figure 4.2 indicates that in networks of $N = 149$ nodes with test sets obtained by a coin-flip procedure, synchronous GKL shows $R \simeq 82\%$, while asynchronous GKL reaches $R \simeq 50\%$. It also shows that in networks of $N = 29$ nodes synchronous and asynchronous SM show $R \simeq 15\%$ and $R \simeq 40\%$ respectively, which decreases to $R \simeq 0.5\%$ at $N = 149$. These observations can be generalized as follows:

- the convergence rate of GKL weakly depends on the system size, while the R of SM degrades in larger systems;
- lack of synchrony inhibits GKL but increases the R of SM.

The different response to asynchronous updates can be also explained by the state-direction bias of GKL: asynchronous state updates can hinder this bias and decrease the R of GKL. At the same time, asynchronous updates can provide randomization for SM in time, and thus promote consensus. We analyze these new observations in further detail in the following section.

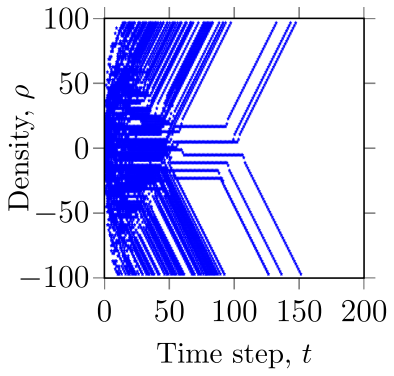
Analysis of the System Evolution

Let us analyze the convergence dynamics of GKL and SM with synchronous and asynchronous updates in ring lattices. We plot the density of the system $\rho[t]$ at every time step $t \in \{0, \ldots, T\}$. Figure 4.3 presents examples of density evolution for successfully and falsely converged networks.

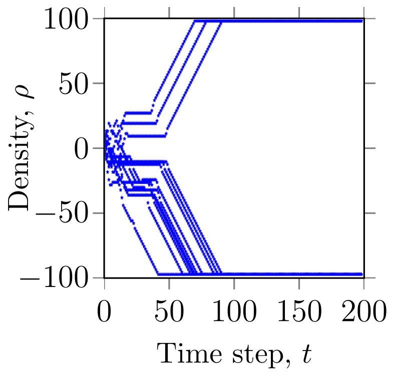
Figure 4.3a presents the evolution of ρ over time for GKL with synchronous updates. It shows that the "state-direction bias" of GKL can steer a system to both successful (Figure 4.3a) and unsuccessful (Figure 4.3b) agreement. Figure 4.3b indicates that unsuccessful agreement is generally a result of initial configurations with ρ_0 close to 0.

It also shows that successful and unsuccessful agreements emerge from the initial configurations with initial densities close to 0, and that density evolution has similar dynamics. Latter effects can be explained by the sensitivity of GKL to different permutations in initial configurations (our simulations show that GKL converges to the wrong majority over "mixed" initial configurations with multiple small clusters).

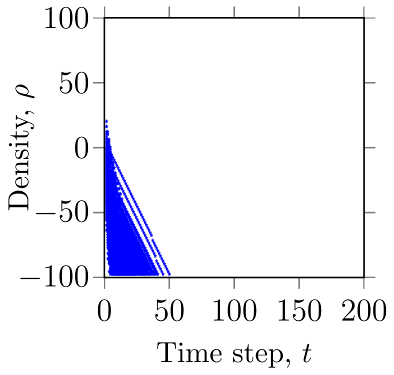
Figure 4.3c shows examples of density evolution for asynchronous GKL, indicating that GKL with sequential asynchronous updates tends to converge to a single



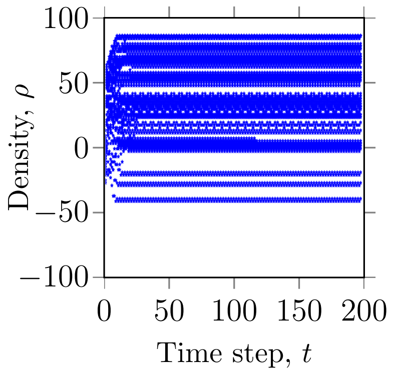
a) Synchronous GKL, about 82% of networks converge to the correct state



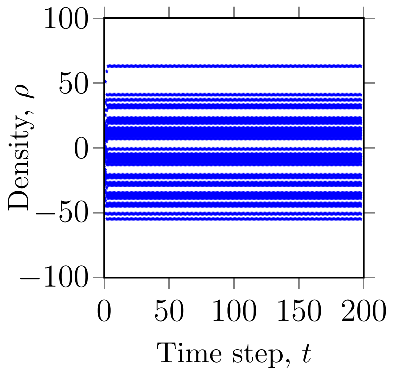
b) Synchronous GKL, about 18% of networks converge to the wrong state



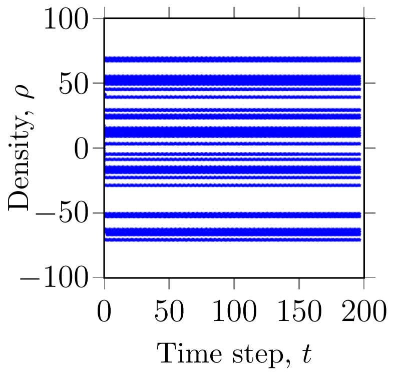
c) Asynchronous GKL, about 50% of networks converge



d) Fully asynchronous GKL, 100% of networks do not converge



e) Synchronous SM, almost 100% of networks do not converge



f) Asynchronous SM, nearly 100% of networks do not converge

Figure 4.3: Examples of density evolution of GKL and SM in ring lattices. $N = 99, K = 3$.

stable but incorrect majority. This happens due to the sequential asynchronous updates. Such updates can steer the direction bias of GKL: networks converge to $-N$ if the update sequence is defined as $0 \to N$, and to N if the updates defined as $N \to 0$. Figure 4.3d shows that GKL does not converge in fully asynchronous systems.

Figures 4.3e and 4.3f shows that both synchronous and asynchronous SM tend to cluster around stable ρ and mostly do not converge. It also shows that asynchronous SM Figure 4.3f shows evolutions to a wider spread set of stable clusters, which can explain its higher R in smaller networks.

In the further analysis we will mainly focus on asynchronous networks that can be more difficult for consensus [DDS87]. In the following we investigate whether stochastic intrusions such as noise and message loss can promote consensus in such networks.

4.3.2 Impact of Additive Noise

Figure 4.5 shows examples of system evolution with GKL and SM consensus in ring lattices with noise.

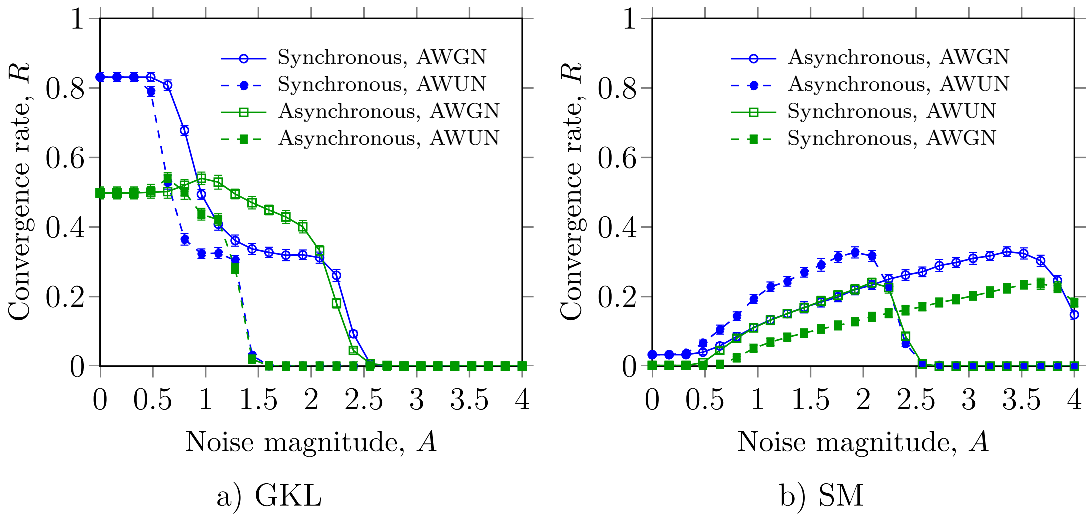
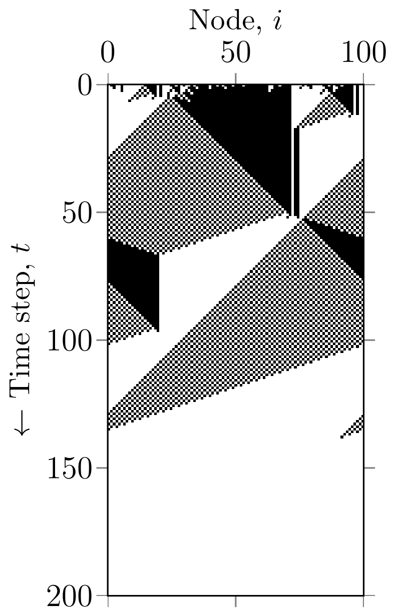
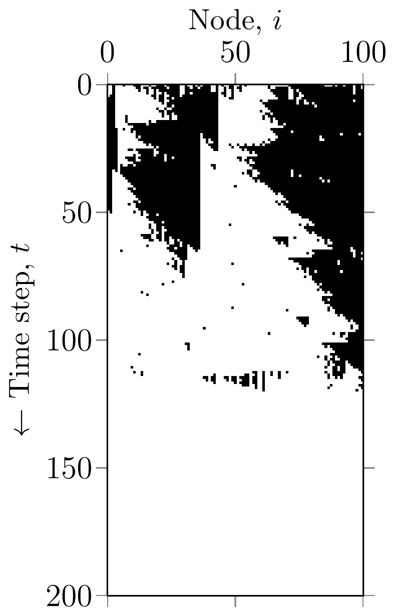


Figure 4.4: Convergence rate of GKL and SM in ring lattices with noise. $N = 99, K = 3$.

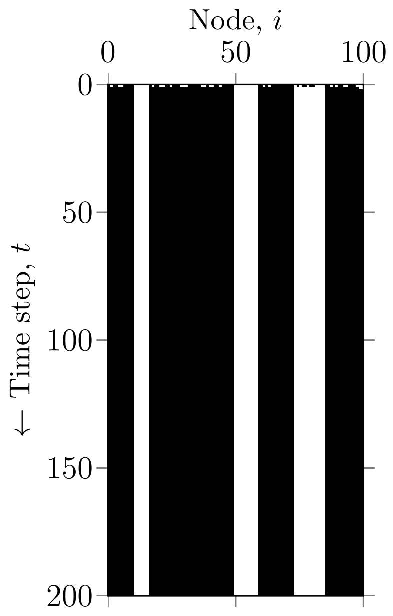
Figure 4.5a presents an example of the synchronous GKL evolution in a network that converges in noiseless conditions (gray areas on a diagram indicate regular switching between available states). Figure 4.5b shows an asynchronous GKL that reaches agreement with noise. Both evolutions illustrate that GKL washes out clusters of nodes with the same state due to its state-direction bias.



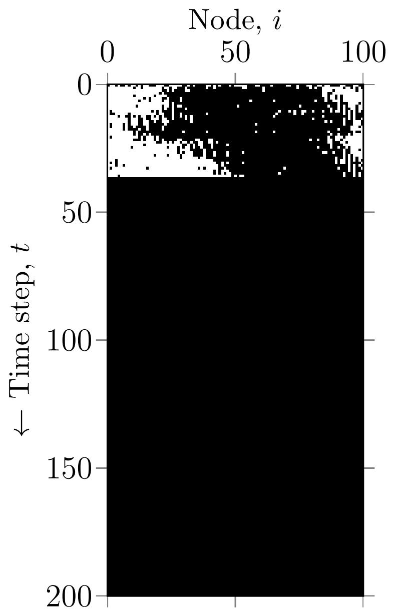
a) Synchronous GKL, state-direction bias dithers clusters, $A = 0$



b) Synchronous GKL, noise inhibits the state-direction bias, $A = 2$



c) Asynchronous SM, clusters, no convergence, $A = 0$



d) Asynchronous SM, noise dithers clusters, convergence, $A = 2$

Figure 4.5: Impact of noise on state evolution in ring lattices. $N = 99, K = 3$.

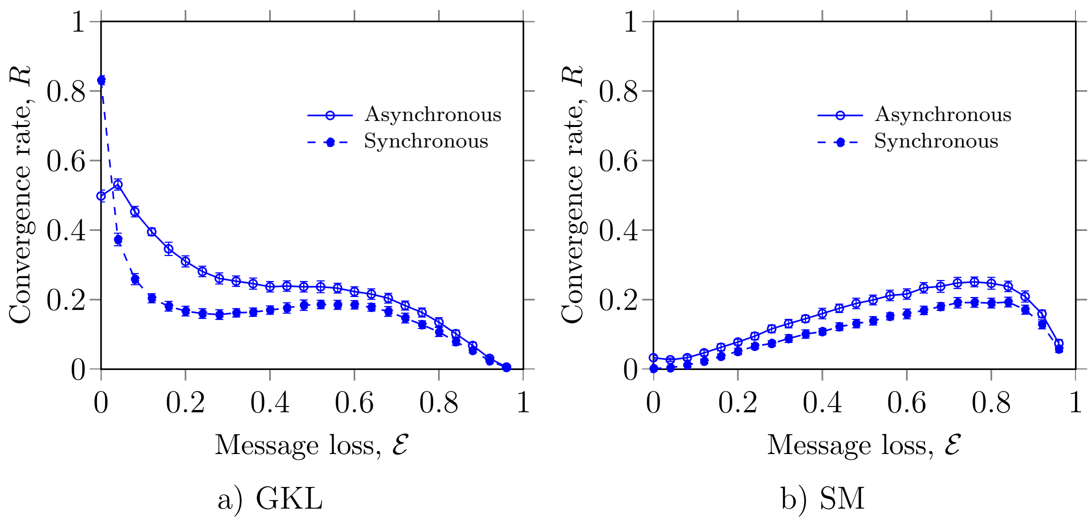


Figure 4.6: SM and GKL in ring lattices with message loss. $N = 99, K = 3$.

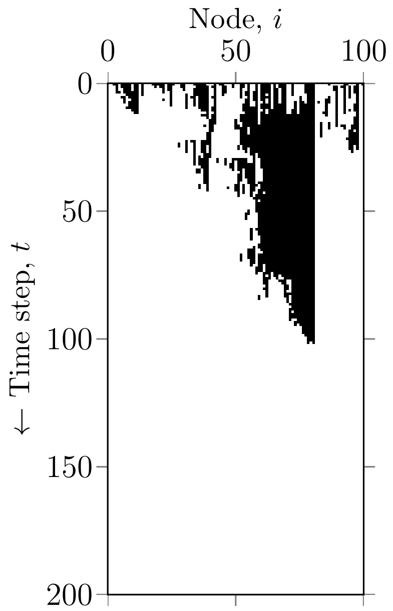
Figure 4.5b shows that asynchronous updates and noise can destabilize this bias. Further growth of the noise magnitude can fully inhibit GKL. Figures 4.5c and 4.5d illustrate how noise can promote SM consensus by destabilizing the clusters of nodes with the same states.

The influence of noise on the convergence rate of GKL and SM in ring lattices is illustrated in Figure 4.4. The new observations can be described as follows. Figure 4.4a indicates that noise inhibits both synchronous and asynchronous GKL to $R \simeq 0$ in ring lattices, and that synchronous GKL degrades faster than asynchronous. It also indicates a weak positive effect on asynchronous GKL with $A \in \{0.75, 1\}$ for AWUN and AWGN. Figure 4.4b shows that noise can significantly increase the convergence rate of SM. AWUN and AWGN influence the consensus differently: with AWUN SM reaches the maximum R at $A \simeq 2$, while with AWGN SM achieves the same R at $A \simeq 3.5$.

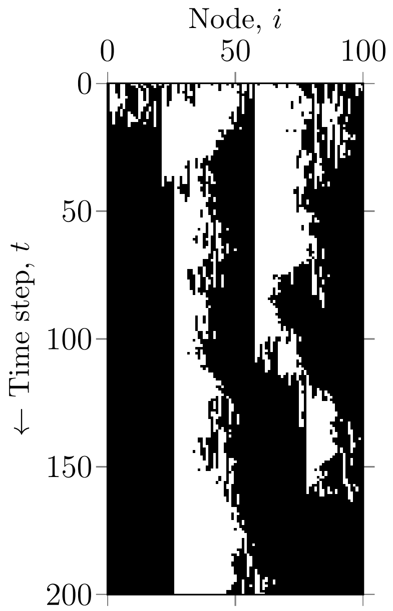
These newly observed effects can be explained by randomization provided by noise. Noise randomizes the information exchange and destabilizes the clusters of nodes that have the same values. GKL is designed to wash out such stable clusters [GKL78], and unstable clusters can inhibit its performance. SM has no built-in cluster-dithering features, and thus can benefit from clusters destabilized by noise.

4.3.3 Impact of Message Loss

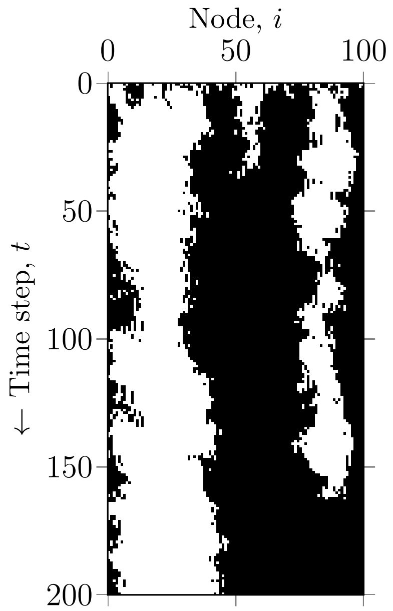
Let us investigate the influence of random message loss on GKL and SM in ring lattices. We vary the message loss probability $\mathcal{E} \in [0, 1)$ with steps of 0.04.



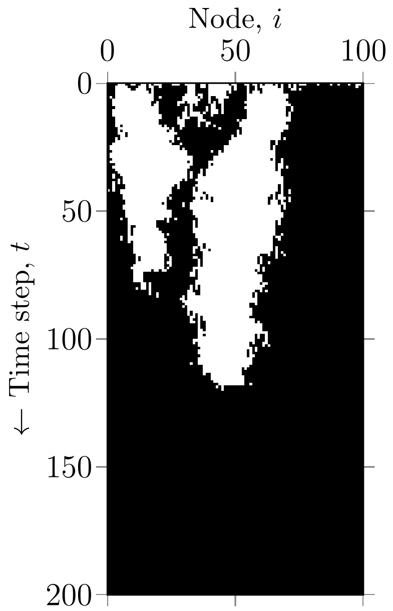
a) Message loss inhibits asynchronous GKL, network converges, $\mathcal{E} = 0.1$



b) Message loss inhibits asynchronous GKL, no convergence, $\mathcal{E} = 0.8$



c) Asynchronous SM, message loss dithers clusters, no convergence, $\mathcal{E} = 0.1$



d) Asynchronous SM, message loss dithers clusters, convergence, $\mathcal{E} = 0.8$

Figure 4.7: State evolution in ring lattices with message loss. $N = 99, K = 3$.

Figure 4.6 shows the impact of random message loss on both algorithms in noiseless ring lattices. It illustrates that message loss can destabilize the state-direction bias of GKL (Figure 4.7a), and even prevent agreement (Figure 4.7b) for high message loss rate. Figure 4.7 shows an example of GKL and SM evolutions for such cases.

On the other hand, SM responds to message loss with increasing convergence rate. This happens due to the fact that random message loss destabilizes the clusters, and thus promotes consensus. Figure 4.7c shows how clusters become unstable, and Figure 4.7d shows that further growth of $\mathcal{E}$ leads to agreement.

Figure 4.6a shows that asynchronous GKL can benefit from message loss below 4%. Figure 4.6b indicates that SM is promoted with up to 80% of message loss.

These observations show that consensus can be promoted by a message loss until a certain threshold when its positive randomizing effect is outweighed by the reduction of the information exchange. This threshold may vary depending on the algorithm, number of neighbors and update scheme. The randomizing effect of message loss is similar to that of noise. With message loss, nodes at the borders of the cluster can randomly change their state, and thus destabilize the cluster.

4.3.4 Impact of Topology Randomization

Figure 4.8 shows the impact of topology randomization on both algorithms in noiseless WS networks.

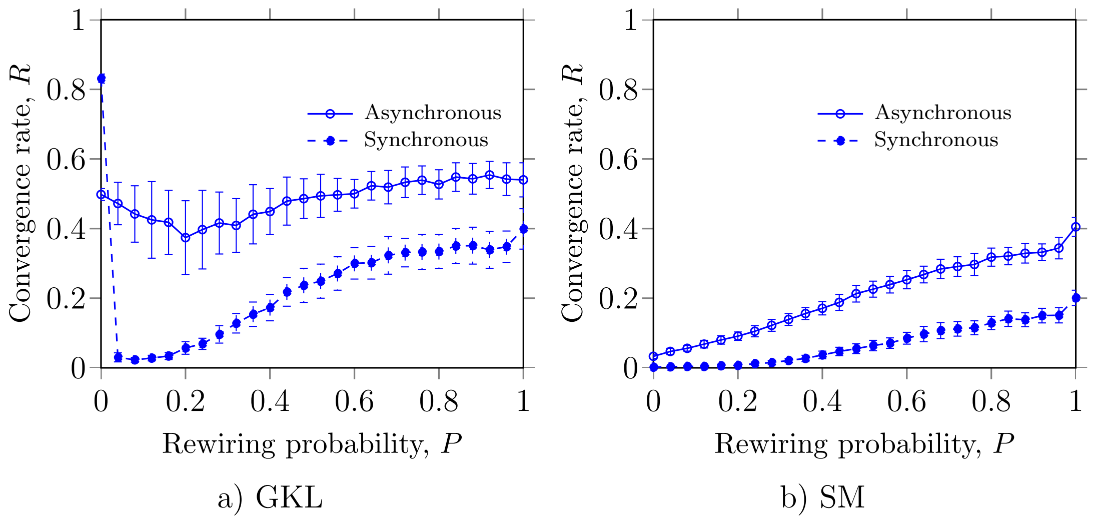


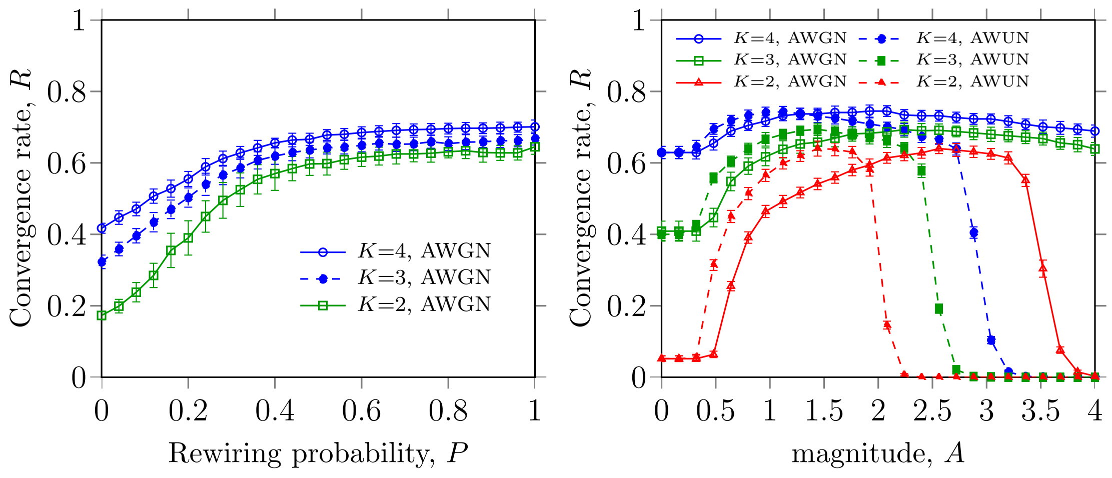
Figure 4.8: Impact of topology randomization on SM and GKL in networks. $N = 99, K = 3$.

It shows that GKL is inhibited by topology randomization in WS networks (Figure 4.8a). The decrease is more evident for synchronous GKL: it degrades from $R \simeq 82\%$ in ring lattices ($P = 0$) to $R \simeq 2\%$ at $P = 0.04$. This happens due to the fact that topology randomization changes the network structure and inhibits the state-direction bias of GKL. However, with further topology randomization ($P \geq 0.1$ for synchronous and $P \geq 0.2$ for asynchronous) GKL converges more often. This happens because topology randomization in WS networks also increases the average link length, which promotes consensus (see Fig. 3.3). Topology randomization in WS networks disrupts the updating sequence, and therefore stronger influences the synchronous consensus. This can explain why synchronous GKL degrades much stronger than the asynchronously updated version.

Figure 4.8b indicates that topology randomization promotes SM consensus, showing that SM can benefit from both randomized network structure and increased link length.

4.3.5 Combined Impact of Noise and Topology Randomization

Figure 4.9 shows the combined influence of topology randomization and noise on SM. Figure 4.9a indicates that in noisy networks ($A = 3.5$) SM with $K \in \{2, 3, 4\}$



a) Response to topology randomization, noise magnitude $A = 3.5$

b) Response to additive noise, WS network, $P = 0.8$

Figure 4.9: Asynchronous SM, response to noise and topology randomization in WS networks with K neighbors, $N = 99$.

neighbors, randomized by topology, reaches $R \geq 60\%$. Figure 4.9b shows that in random WS networks ($P = 0.8$) SM is promoted by noise in a manner similar to that of ring lattices. It also shows that noise promotes SM until a certain threshold, while the randomizing effect outweighs the information exchange. This threshold grows with higher information exchange, e.g. SM with AWUN and $K = 4$ neighbors is promoted by noise with magnitudes below $A = 3$, while SM with $K = 2$ degrades already with noise magnitude $A \geq 1.8$.

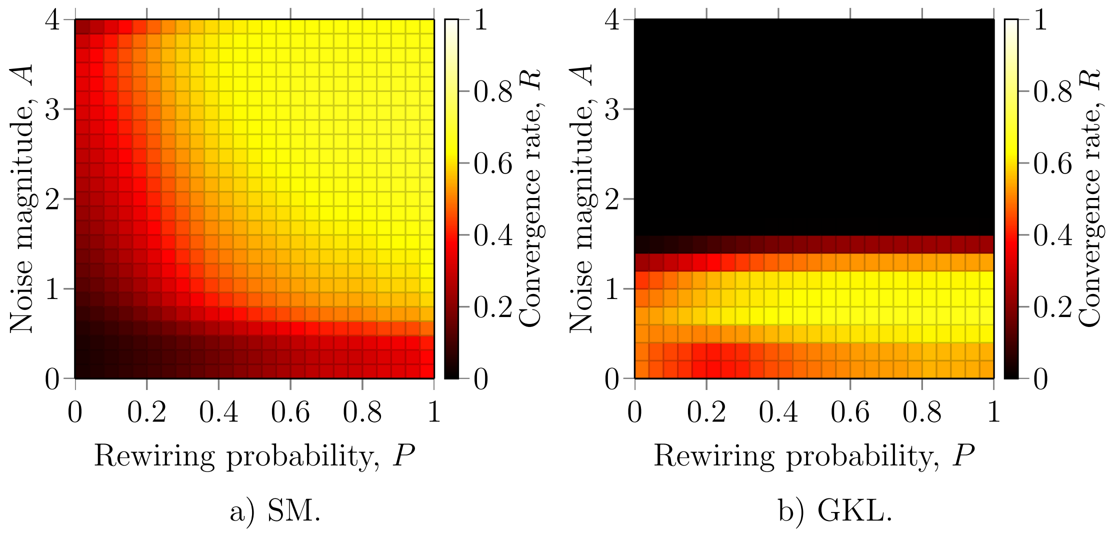


Figure 4.10: Asynchronous SM and GKL, response to noise and topology randomization in WS networks. $K = 3$, $N = 99$.

Figures of convergence rate indicate that a combination of topology randomization and noise produces a cumulative positive effect on SM. This combined influence for asynchronous GKL and SM is further shown in Figure 4.10 . It illustrates that randomization by topology and noise produce a cumulative effect when combined. This cumulative effect is possible due to different types of randomization that complement each other. The character of the impact, however, varies for GKL and SM: the area of optimal disturbances for GKL is smaller and lies within lower values of noise and rewiring probability. This can be explained by previously observed response of GKL to sole randomization by noise and topology, determined by its direction bias.

These new results show that an optimal combination of disturbances can be selected to promote binary majority consensus, depending on given system conditions.

4.3.6 Combined Impact of Message Loss, Noise, and Topology

In the previous sections we show that random topology, noise, and message loss can promote consensus due to their anti-clustering influence. This motivates us to investigate whether their combined influence can increase the positive impact.

Figures 4.11 and 4.12 present the combined influence of topology randomization and message loss in noiseless and noisy networks, respectively.

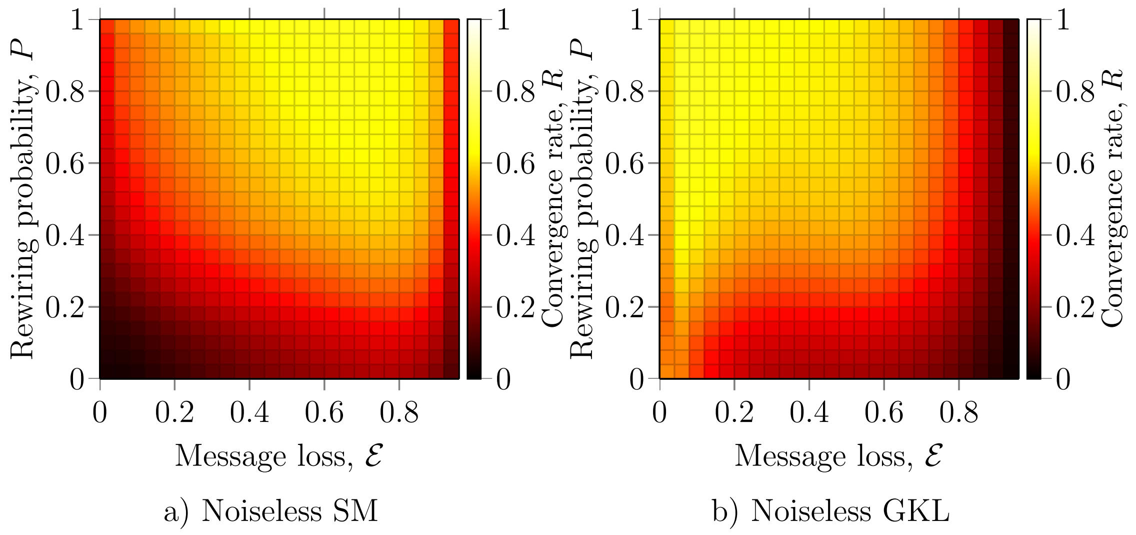


Figure 4.11: Asynchronous SM and GKL with combined randomization by message loss and topology in noiseless WS networks. $K = 3$, $N = 99$.

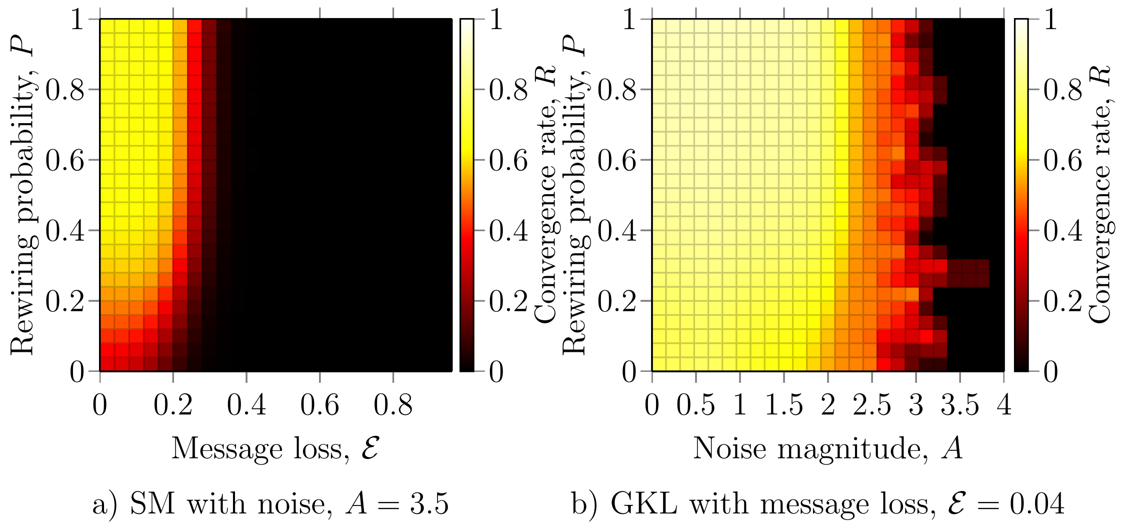


Figure 4.12: Asynchronous SM and GKL with message loss and topology randomization in noisy WS networks. $K = 3$, $N = 99$.

A comparison of Figures 4.11a and 4.12a shows that randomization with a high level of noise increases the convergence rate of SM, but significantly lowers robustness towards message loss. This happens due to the excessive randomization that outweighs the information exchange which is additionally reduced by message loss.

Figure 4.12b compared to Figure 4.10b shows that in networks with low message loss GKL randomized by noise and topologically converges more often. This can be explained by positive randomizing effect of low message loss for GKL strengthened by noise and increased link length (as a result of randomized topology).

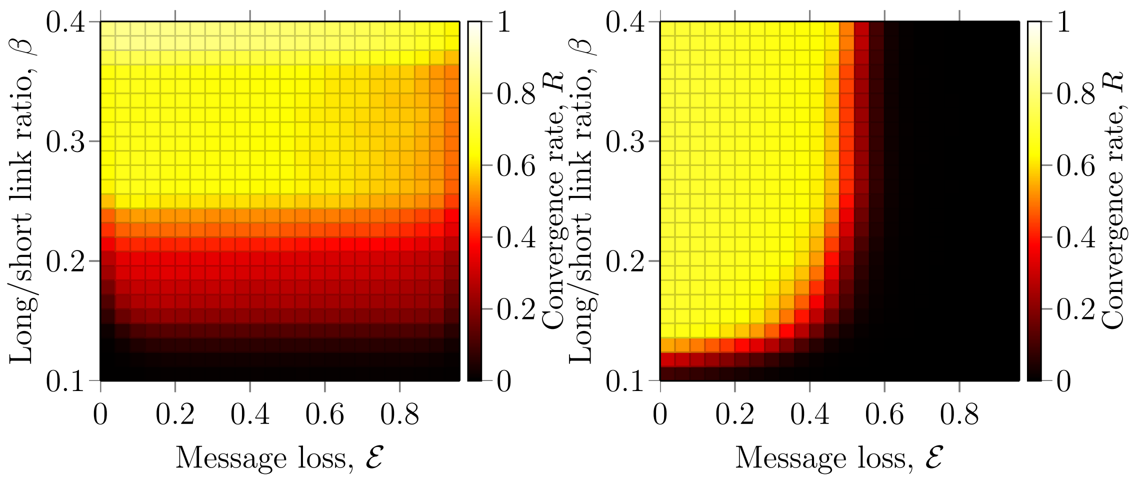
We can generalize these new observations as follows. Solely applied additive noise, topology randomization and message loss can promote consensus. The positive impact of such randomization has a cumulative effect if these disturbances are applied jointly. Randomization can inhibit consensus as well as promote it, and both effects are stronger expressed with combined randomization: with combined randomization algorithms can achieve higher R, but with further growth of disturbances R quickly decreases.

In other words, results shows that although with multiple sources of randomization a positive impact is expressed stronger, the area of optimal combination of disturbances becomes more narrow. Consequently, consensus promotion with an optimal combination of disturbances is sensitive to an exceeding level of any of these disturbances. This brings up a question for the future research: can a combination of randomizing disturbances be described by a single metric that accounts for the types and the levels of stochastic intrusions? This ambitious challenge can be approached with an extended analysis of de-clustering impact of different disturbances and their fusion at an algorithm level. Interested scholar is advised to look into the network semantic extraction and the data fusion techniques.

4.3.7 Impact of Noise and Message Loss in Waxman Networks

In this section we study GKL and SM in Waxman networks. Unlike WS networks that are used to model natural random networks [WS98], Waxman networks are used to model human-designed random networks [Wax88]. The behavior of the binary consensus in such networks can provide insights on dynamics of distributed decision making algorithms in human-designed random networks, e.g., wireless sensor networks.

The network is modeled as described in Sections 3.3.3 and 4.2. Recall that the parameter β here has an impact similar to that of P in WS networks, with an important difference: growth of β increases both the average link length and the node degree. We vary β between 0.1 and 0.4 with fixed $\alpha = 0.05$. We artificially limit the maximum node degree to $2K = 6$ to match that of WS networks. Our simulations show that with initial settings of $\beta = 0.1$ and $\alpha = 0.05$, Waxman networks are not guaranteed to be connected. This can explain low figures of R in networks with $\beta \leq 0.25$.



a) Noiseless conditions, $A = 0$. b) Network with noise, $A = 2.0$.

Figure 4.13: Asynchronous SM promoted with message loss and topology randomization in Waxman networks. $K = 3$, $N = 99$.

Figure 4.13 shows the convergence rate of asynchronous SM in Waxman networks with noise and message loss. Figure 4.13a indicates effects similar to that of Figures 4.12a and 4.10: additive noise promotes consensus in random networks but also lowers the robustness to message loss (recall that Waxman networks are initially random networks). It also shows that with sufficient connectivity ($\beta \geq 0.25$) SM reaches high R and tolerates high level of message loss ($\mathcal{E} \leq 80\%$ with $R \geq 80\%$). Finally, it shows that in Waxman networks noise can promote consensus in networks with low connectivity ($\beta \leq 0.25$).

This can be generalized as follows. Although WS and Waxman networks describe different random networked models and are modeled differently, consensus algorithms exhibit similar response to separately and jointly applied randomizing disturbances.

We omit results for GKL since due to non-guaranteed connectivity and random state updates in Waxman networks it achieves R below 10%. The latter effect can be mitigated if the network structure is made “known” to the algorithm and a direction bias can be established. This can be done via embedding the location parameters in the nodes decision-making system, but it is out of scope of our research and of moderate scientific importance.

4.3.8 Impact of the Scope of Randomization

In this section we analyze the difference between randomization that includes the node’s own state, i.e., where $\sigma_{i,i} \neq \sigma_i$ and the one that does not affect it, i.e., where $\sigma_{i,i} = \sigma_i$.

Figure 4.14 shows how noise and message loss influence asynchronous GKL and SM in ring lattices.

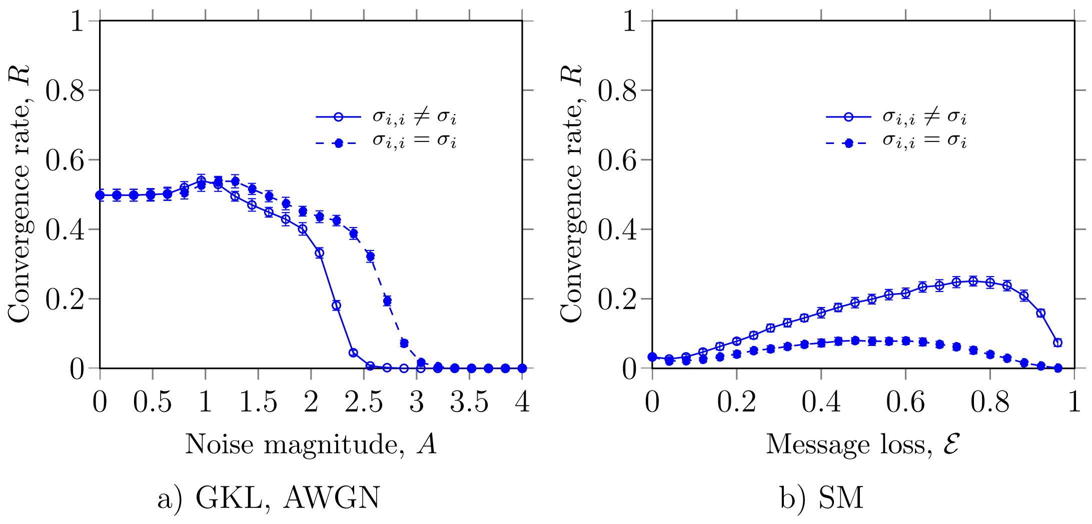


Figure 4.14: Impact of different scope of randomization on GKL and SM in noisy ring lattices. $N = 99, K = 3, P = 0, \mathcal{E} = 0$.

It shows that excluding $\sigma_{i,i}$ from randomization reduces the randomization impact, whether this impact is positive or negative. Next, Figures 4.15 and 4.16 show how excluding $\sigma_{i,i}$ from randomization influences combined randomization by noise, message loss, and topology. A pairwise comparison of Figures 4.15a and 4.15b with Figures 4.11a and 4.11b shows effects similar to that of randomization in ring lattices: randomization shows less influence when $\sigma_{i,i}$ is not affected.

However, a pairwise comparison of Figures 4.16a and 4.16b with Figures 4.12a and 4.12b shows that if all three types of randomization are combined, the positive effect is higher when $\sigma_{i,i}$ is not affected. In other words, the optimal area of consensus promotion by randomization of different origin is broader when $\sigma_{i,i}$ is not affected. Latter effect can be explained by the fact that complementary randomization, which has a positive effect, can reach "saturation" when further growth of stochasticity leads to random state dynamics and decreases the convergence rate. In addition to this, message loss decreases information exchange, inhibiting consensus.

4.3.9 Impact of Randomization on Convergence Speed

In this section we describe the impact of randomization on convergence speed F. It is often argued that randomization destabilizes the convergence process

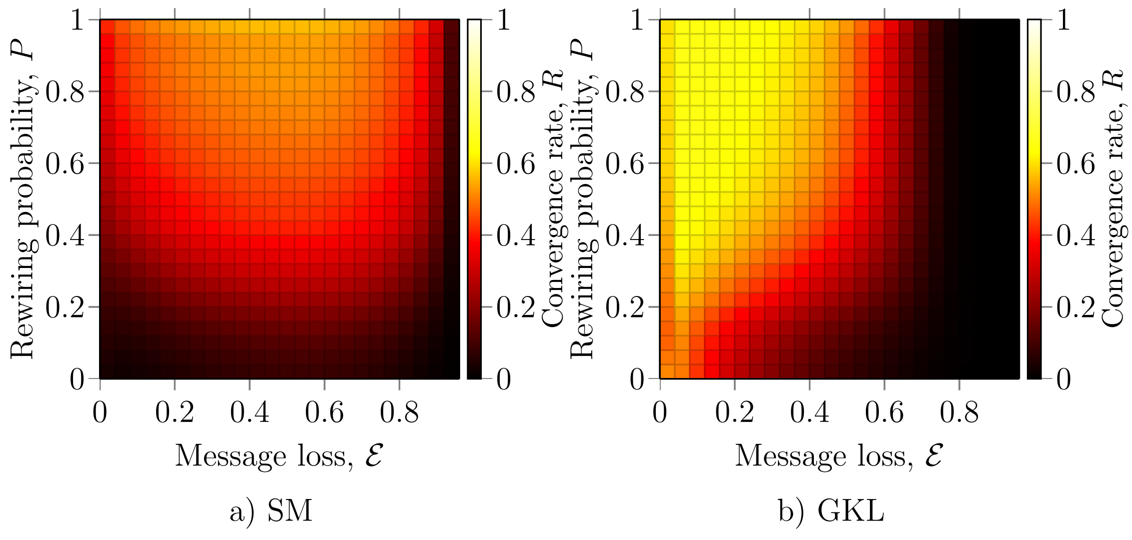


Figure 4.15: Asynchronous SM and GKL. Message loss and topology randomization in noiseless WS networks, $\sigma_{i,i} = \sigma_i$. $K = 3$, $N = 99$.

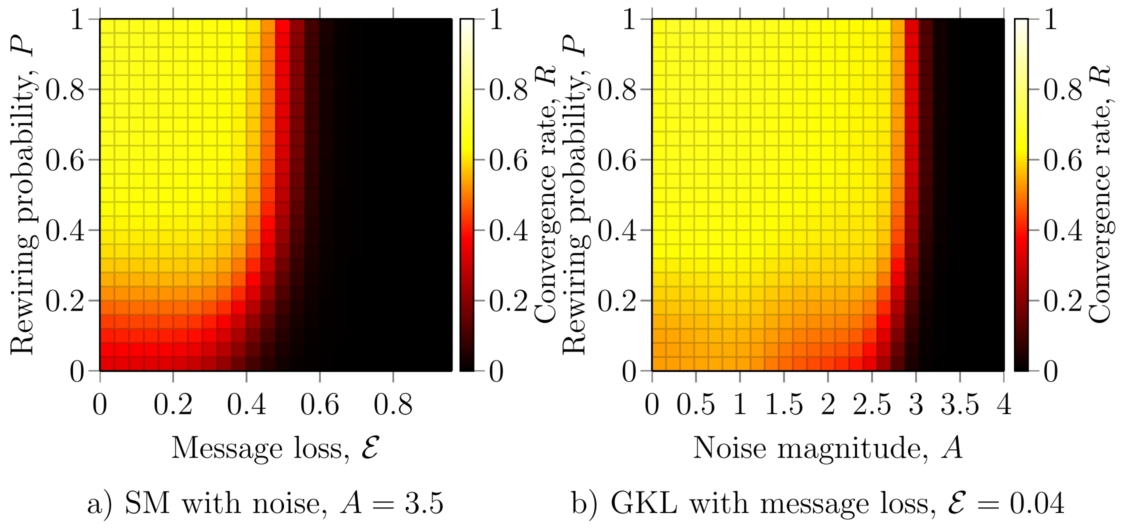


Figure 4.16: Asynchronous SM and GKL. Message loss and topology randomization in noisy WS networks. $K = 3$, $N = 99$, $\sigma_{i,i} = \sigma_i$.

and affects convergence speed (time) [Asp03]. Such influence can be adverse for wait-free consensus where convergence time is bounded. For a wait-free consensus a critical level for convergence speed is close to 0%. Convergence speed $F \simeq 0$ indicates that the network requires more than T time steps to converge and can suggest that an extension of the consensus cycle duration T is required. Recall that the convergence speed F is defined as $F = \frac{t_c}{T}$, where $t_c \in \{0, \ldots, T\}$ is the time step at which the system has reached the agreement. For simplicity we register the convergence speed only for successfully converged networks. Let us show how the randomization by noise, topology and message loss influences the convergence speed of GKL and SM.

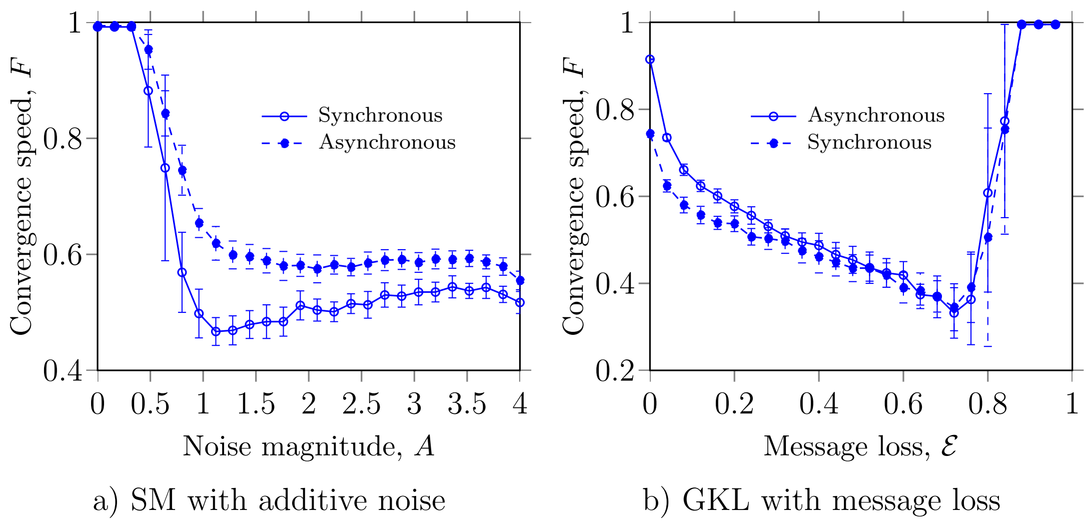


Figure 4.17: Convergence speed of GKL and SM in ring lattices with noise and message loss. $N = 99, K = 3$.

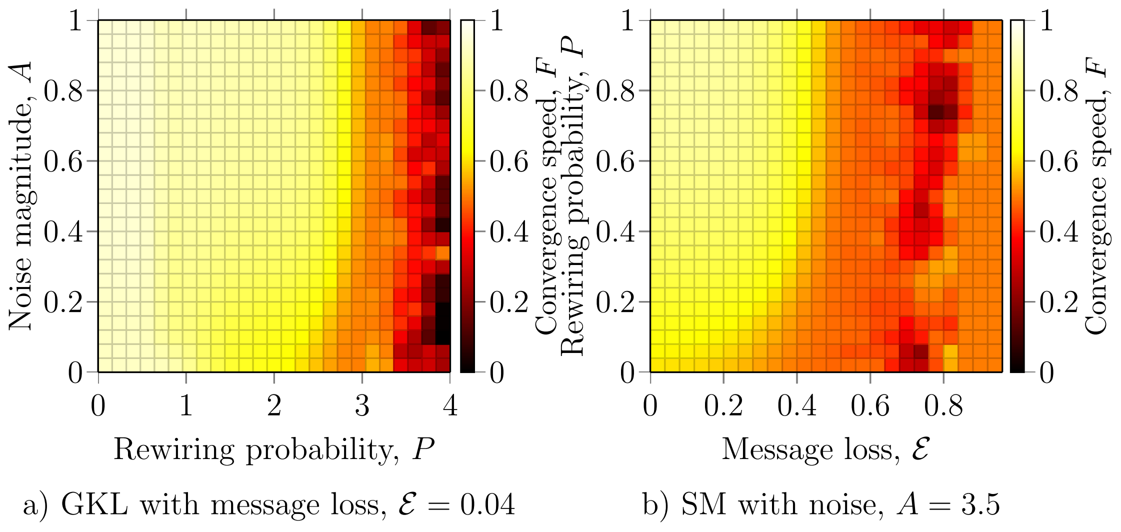


Figure 4.18: Convergence speed of GKL and SM in WS networks with combined noise and message loss. $N = 99, K = 3$.

Figure 4.17a shows that asynchronous SM converges slower with noise than without noise. As mentioned above, randomization can affect convergence time, as agreement itself becomes probabilistic. This can explain a decrease in convergence speed under noise. Figure 4.17b shows how message loss decreases the convergence speed of GKL.

Note that in addition to randomization that affects the convergence, message loss itself contributes to a decrease in convergence speed by reducing the information exchange. This explains the heavier impact of message loss on convergence speed. An increase of convergence speed for GKL with message loss above $\simeq 70\%$ can be explained as follows. We consider the convergence speed only for successfully converged setups, and with $\mathcal{E} \geq 70\%$ GKL shows $R \leq 1\%$. Initial configurations that provide convergence in such conditions have ρ_0 close to N or $-N$, and if such a network converges, it converges quickly.

Figure 4.18 illustrates the impact of combined randomization on convergence speed. It indicates that a combined randomization by noise and message loss decreases the convergence speed stronger than a sole randomization by noise or message loss. It also shows that the randomization by topology has little impact on convergence speed.

However, in all studied setups the convergence rate does not decrease close to 0%. It shows that randomization by topology, noise and message loss increases the convergence rate but does not critically decrease the convergence speed.

4.4 Summary

In this chapter we analyze the impact of randomization by noise, message loss, and topology on convergence rate and speed of two standard algorithms, namely GKL and SM. These are simple one-step algorithms for binary majority consensus. In noiseless ring lattices GKL shows convergence rate of nearly 82%, while under the same conditions SM scores only 1%. We show that randomization by noise, message loss, and topology generally increases the efficiency of SM. These disturbances in some cases can promote the asynchronous GKL as well. However, the positive impact of disturbances on GKL is weaker due to its direction bias.

Further, we observe that the positive impact of randomization is often related to its ability to counter network clustering. Network clustering is a process of cluster formation of the nodes that have the same states. Since consensus algorithms are designed to steer the system to the same state for all nodes, such clusters inhibit their performance.

Additive noise, message loss, and topology randomization produce a cumulative effect when combined, due to the different types of counter-clustering influence they provide. Simulation results indicate that an optimal combination of randomizing disturbances of different origin can be selected, depending on the given system conditions. With known systemic conditions, such as the ratio of random links, level of noise, etc., a combination of disturbances to promote consensus (that compensates and complements the given conditions) can be selected. Multiple sources of randomization can narrow down the optimal area of consensus promotion, but this can be mitigated by excluding $\sigma_{i,i}$ from the scope of randomization. The update scheme has a twofold impact on the evolution of the consensus systems. On the one hand, algorithms designed to perform in certain types of systems, e.g., GKL, rarely converge in asynchronous systems. On the other hand, simple algorithms, e.g., SM in asynchronous systems converge more often. Furthermore, randomization by noise, message loss and topology can promote simple consensus algorithms in asynchronous systems even stronger than in synchronous.

The main contributions of this chapter can be summarized as follows:

- explicit randomizing disturbances can promote binary majority consensus in asynchronous and topologically randomized systems;
- a cumulative promotion effect can be produced by combining different types of randomization;
- an optimal combination of disturbances and their levels can be selected, depending on system conditions;
- these effects are evident in ring lattices, WS and Waxman networks.

These new results can be interpreted in support of the hypothesis that randomizing disturbances are not only a “basic requirement” [MMDA04] for modeling of distributed systems, but their intrinsic feature that can ensure high performance.

Although we show that randomization can be used to increase the convergence rate of wait-free binary majority consensus, several open questions remain. We observe that SM performs efficiently in strongly randomized environments but poorly in ordered and noiseless ones. GKL indicates opposite dynamics, with high convergence rate in synchronized undisturbed ring lattices, but low R in randomized environments. A challenging task therefore is to design a consensus algorithm that can perform efficiently in both ordered noiseless grids and randomized environments. An intuitive approach is to consider a more complex algorithm design. As we mention earlier, it can address, e.g., a deeper analysis of the anti-clustering influences and their fusion at an algorithm level. However, a simple algorithm that directly embeds random elements in its structure can be a possible solution as well. Randomization by noise and message loss is beneficial, and randomization by network topology indicates the most positive effect when combined with noise and message loss. In the following chapter we show how random neighbor selection, embedded in the algorithm, can provide a positive impact similar to that of message loss and topology randomization. We show that such embedded randomization can be further extended by noise and errors to increase the convergence rate.

Another important question, which is not considered in this chapter, is the robustness of simple algorithms towards faulty node behavior, and whether such robustness can be improved by randomization, or not. We address this issue in Chapter 6.

Wait-free distributed consensus often assumes that nodes can access only limited local information, and consensus termination is performed through the consensus termination time T, common for all nodes. In case of deterministic consensus algorithms, e.g., synchronous GKL, this is a valid termination condition: the state-direction bias steadily steers the system to the agreement. In a randomized consensus system, however, stochasticity can steer the system away from an agreement state. Therefore, the question is: if a randomized system has converged, how stable such agreement is? In the following chapters we investigate randomized binary majority consensus in respect to the aforementioned questions.

CHAPTER 5

Consensus Randomized by Neighbor Selection

5.1 Introduction and Motivation

An ideal algorithm for binary majority consensus should provide convergence rate of 100%. Investigations in the domain of binary majority consensus date back for almost forty years to a paper by Gacs *et al.* [GKL78], which proposes an algorithm (later named after the authors, Gacs-Kurdyumov-Levin (GKL)) that solves binary majority consensus problem with efficiency of $\simeq 82\%$. Since then, several deterministic solutions have been proposed with convergence rates up to 86% (for more details, see Table 2.1). Land and Belew [LB95] showed that deterministic algorithms for binary majority consensus cannot reach $R = 100\%$. However, this restriction does not apply to randomized consensus algorithms. Randomization is a technique that uses random processes (often considered as negative disturbances) to increase convergence rate and fault tolerance of consensus algorithms [Asp03]. In some cases, randomization can even guarantee successful agreement, but the convergence itself and the time till the actual agreement become probabilistic (more details given in Section 2.5). This condition limits the applicability of randomization to *wait-free* consensus algorithms (see Section 2.2.1). On one hand, wait-free consensus algorithms can be inhibited by disturbances. Thus, GKL consensus can significantly decrease the convergence rate with noise [5] or errors [MMDA04].

On the other hand, several studies [MMDA04, Cha96, Fat13] indicate that randomization can also be beneficial for wait-free consensus algorithms. In particular, Moreira *et al.* show that Simple Majority (SM) consensus can reach up to 85% convergence rate when randomized by errors and topologically. SM exhibits high convergence rate only in strongly randomized environments, while in synchronous undisturbed networks R is low. A challenging task is, therefore, to design an algorithm with high convergence rate in both ordered undisturbed and randomized networks. A track of studies on distributed consensus show that this task is difficult for systems with restricted time and connectivity. However, some recent works show serious advancements. In addition to the aforementioned paper by Moreira *et al.* [MMDA04], another study [KKBT12] shows that a self-organizing algorithm, accompanied by stochastic disturbances can guarantee a synchronization in a wide range of distributed systems.

Investigations, presented in Chapter 4, show that standard algorithms can increase efficiency in environments, randomized by various disturbances. In particular, we show that SM consensus significantly increases convergence rate if randomized by additive noise, message loss, and topology. We explain this positive impact by the cluster dithering process, which is enforced by randomization. Some types of randomization, such as noise or errors, can be directly embedded in a consensus rule to promote its efficiency in ordered networks. Topology randomization, however, is generally provided by the underlying networking model.

In this chapter we introduce a new binary majority consensus algorithm randomized by neighbor selection. Random neighbor selection can be embedded in the algorithm to provide beneficial topology randomization in the neighborhood of each node, while the global underlying network topology remains intact. Further in this chapter we:

- propose two schemes of random neighbor selection and extend this randomization with noise and errors;
- show that cumulative effect of such a combination can promote convergence rate closely to 100% in random and ordered environments;
- discuss the agreement dynamics and propose a direction to seek for a more advanced solution.

This chapter is organized as follows. Next section describes system modeling and algorithm design. Section 5.4 investigates the proposed randomized algorithm in various conditions. Section 5.4.4 discusses the agreement dynamics. Finally, Section 5.5 concludes the chapter with the summary of results and future research directions. Some of the results presented in this chapter have been published in [4, 5].

5.2 System Model

For the network modeling we implement three types of networks. We model ring lattices as an example of ordered networks, and for randomized networks modeling we use Watts-Strogatz (WS) and Waxman random graphs. Networks are built as described in Sections 3.3 and 4.2.1. In our simulations, network topology is generated once at the beginning of the simulation. After the network generation, each node is connected to its neighboring nodes with bidirectional links. Nodes that have connections to node i form a set of its neighbors $S, ||S|| \leq 2K$. Nodes

$j \in S, j < i$ sorted in ascending order form the vector of the left-side neighbors $S_l, ||S_l|| \leq K$. Similarly, the vector of right-side neighbors S_r is built.

5.2.1 Randomization by Topology

Besides random neighbor selection embedded in RNM, we use randomization by network topology by WS and Waxman graphs, described in Sections 3.3 and 4.2.1. We also use randomization by noise, errors and message loss. Let us briefly describe the latter.

5.2.2 Randomization by Noise, Errors and Message Loss

In Chapter 4, we show that stochastic intrusions of various type can have a cumulative effect when combined.

In this chapter we extend randomization in RNM by adding Additive White Gaussian Noise (AWGN) in message exchange of RNM as described in Section 3.4.1.

Positive impact of randomization by errors reported by Moreira *et al.* [MMDA04] motivated us to investigate RNM with additional randomization by errors. Errors are modeled as described in Section 3.4.3.

Scope of Randomization and Initial Configurations

We apply randomizing disturbances with the full scope of randomization, where own state of the node i can be affected by noise, errors or message loss, i.e., $\sigma_i \neq \sigma i, i$. For more details on the scope of randomization see Section 3.4.4.

In our message exchange system message loss, errors and noise are applied independently from each other. Therefore if a message is lost a receiving node can still receive a noise sample or an erroneous state information. Such a joint scheme allows us not to specifically distinguish, whether a certain randomizing disturbance applied as an “environmental influence” or an “internal” feature of an algorithm. We keep the models simple so that randomizing disturbances can be embedded in the algorithm without a significant increase in its complexity, if needed.

In this chapter we simulate consensus over initial configurations obtained by a coin-flip trial, as described in Sections 2.4.6 and 3.7.

5.3 Random Neighbor Majority

In this section we describe the consensus algorithm with the random neighbor selection and additional randomization by errors and noise — the Random Neighbor Majority (RNM) consensus. In the aforementioned studies of simple algorithms, a randomization by topology is generally provided by a network model, as simple algorithms often posses no data of network structure. Embedding the network topology information in the algorithm increases its complexity and can be done in more sophisticated algorithms. In the following section we describe how a random neighbor selection can be embedded in a simple one-step algorithm without significant increase in its complexity. Such a scheme allows to mimic the effect of global topology randomization on a local scale. It can also provide a "de-clustering" effect if an algorithm is running in an ordered network.

As a basis algorithm we use SM consensus (see Sections 2.4.6 and 4.2.1), arguably the simplest algorithm for the binary majority consensus. With SM consensus, each networked node, connected to its $2K$ neighbors, receives messages from all of its neighbors at each time step. Such communication is illustrated in Figure 5.1.

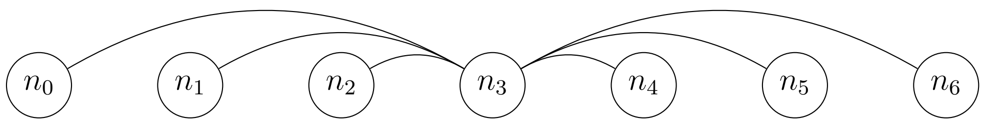


Figure 5.1: The node n_3 is connected to its $2K = 6$ neighbors

In Chapter 4 we show that SM is heavily inhibited by the network clustering and that randomization by topology, provided by the WS model, can produce a "de-clustering" impact. Here we propose randomization by neighbor selection that can dither network clusters without changing the underlying network topology.

We introduce two random neighbor selection schemes, where at each time step $t \in \{0, \ldots, T\}$ nodes follow one of the neighbor selection schemes:

- a balanced (or uniform) neighbor selection, where a node randomly selects C out of $2K$ available neighbors;
- an update-biased neighbor selection, where a node randomly selects C out of K left-side neighbors (that have already performed an update to a "next" time step).

The motivation for these two selection schemes is the following. In environments, where a partial synchrony or, e.g., an update sequence can be established (ring lattices or some WS networks), it can be beneficial for each node to use the information from the nodes that have already updated their state. Random selection of such neighbors can additionally randomize the information exchange and counteract clustering. If the network is fully random and update sequence cannot be established (e.g., Waxman networks and strongly randomized WS networks), uniform neighbor selection can be used to ensure counter-clustering randomization. Therefore, these two schemes of neighbor selection can be used to promote consensus in different types of networks. Let us describe them in more detail.

5.3.1 Uniform Neighbor Selection

We use uniform (balanced) neighbor selection scheme on a basis of a SM, so that expression (4.1) transforms to:

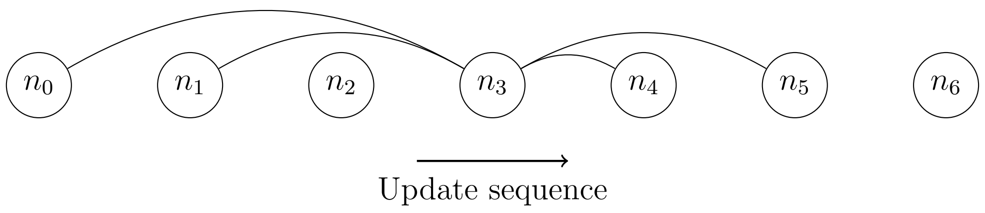
$$\sigma_i[t+1] = G\left(\sigma_{i,i}[t] + \sum_{j=1}^{C} \sigma_{i,n_j}[t]\right) . \tag{5.1}$$

Here, neighbor n_j is randomly selected C times from a set of neighbors S. Neighbor selection is implemented as a random uniform selection with repetition; i.e., neighbors are selected with equal probabilities and the same neighbor can be selected more than once. This scheme is illustrated in Figure 5.2, showing evolution of a neighbor selection of a node n_3 over time in an asynchronous network with sequential updates. It illustrates a balanced neighbor selection of a node that receives information from $C = 4$ neighbors, randomly selected at each time step $t \in \{1, 2, 3\}$. Neighbors are selected uniformly from a set of neighbors S, provided by the network. Selection is uniform and independent from the update sequence (recall that we define asynchronous sequential update according to the nodes' indices, i.e., $0 \rightarrow N$). Thus, Figure 5.2b shows that at a time $t = 2$ node n_3 can receive information from the nodes that have already updated their state (e.g., $\sigma_{n_0}[t = 2]$) and from nodes that have not performed an update yet (e.g., $\sigma_{n_5}[t = 1]$).

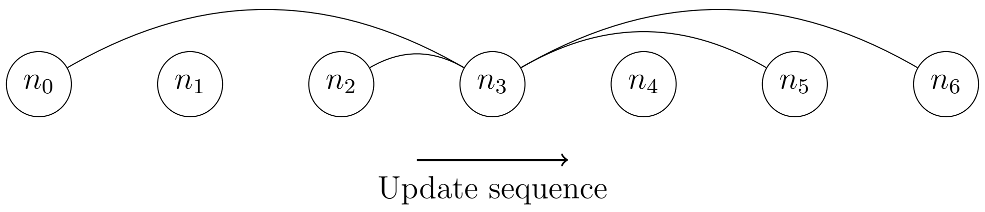
Such a selection scheme can increase the convergence rate in networks, where the update sequence is disrupted by topology randomization (e.g., in Waxman networks).

5.3.2 Update-Biased Neighbor Selection

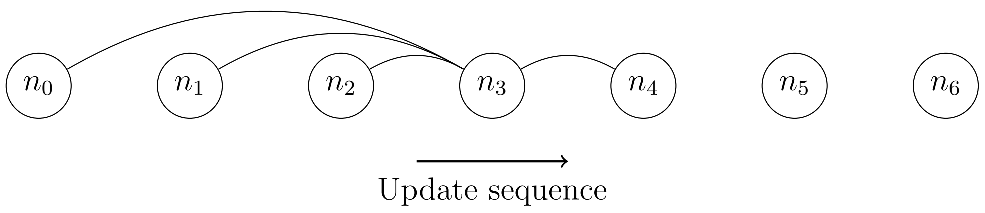
For networks where the update sequence is not disrupted by the topology randomization (or can be established as, e.g., asynchronous sequential update) it can be beneficial to select neighbors from the set of those that already have performed an update. For this we implement an update-biased neighbor selection scheme.



a) Node n_3 is connected to its $C = 4$ randomly selected neighbors at $t = 1$



b) Node n_3 is connected to its $C = 4$ randomly selected neighbors at $t = 2$



c) Node n_3 is connected to its $C = 4$ randomly selected neighbors at $t = 3$

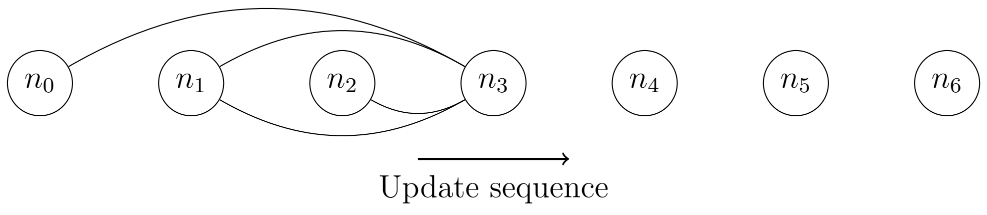
Figure 5.2: The node n_3 is connected to C randomly selected neighbors. $K = 3, C = 4$.

Let us for simplicity assume that the update sequence follows nodes' indices, i.e., from $i = 0$ to $i = N$ or otherwise, from "left" to "right". With such an update sequence, e.g., at time $t = 3$, nodes $j \in S_l$ will send to the node i their state information $\sigma_j[t = 3]$, while nodes $j \in S_r$ can only send their state information $\sigma_j[t = 2]$. With an update-biased selection a node i chooses neighbors only from the set of left-side neighbors S_l. Then, expression (4.1) becomes:

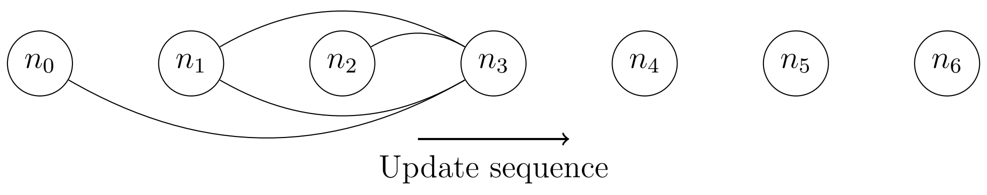
$$\sigma_i[t+1] = G\left(\sigma_{i,i}[t] + \sum_{j=1}^{C} \sigma_{i,n_j}[t]\right) . \tag{5.2}$$

Here, a random neighbor n_j is randomly selected C times from a set of left-side neighbors S_l. The selection procedure is again a uniform one with repetition.

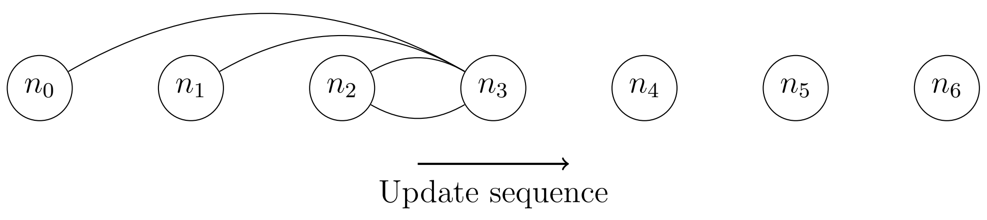
This scheme can be beneficial with asynchronous sequential updates, when



a) $C \geq K$, n_1 selected twice at $t = 1$



b) $C \geq K$, n_1 selected twice at $t = 2$



c) $C \geq K$, n_2 selected twice at $t = 3$

Figure 5.3: A networked node, randomly connected to its $C \geq K$ randomly selected neighbors from the left.

nodes are updated asynchronously, one after another according to their indices. Such selection has two important effects: (a) it reduces the effective radius of a node i, but (b) allows to use the latest available states of the neighboring nodes. This scheme is illustrated in Figure 5.3. It shows that a node n_3 can select C out of K neighbors from its set of left-side neighbors, $S_l, ||S_l|| = K$. Here, if $C \geq K$ nodes will be selected multiple times.

5.4 Performance Analysis of Random Neighbor Majority

In the following sections we study RNM in comparison with SM and GKL in ordered and randomized WS and Waxman networks. We start with RNM in in noiseless and error-free systems of different size and systems with message loss. Next, in Section 5.4.2 we investigate RNM with additional randomization by noise and errors. In Section 5.4.3 we address efficiency of RNM in Waxman networks. In Section 5.4.4 we analyze the convergence dynamics of RNM and explain mechanisms behind its high convergence rate.

5.4.1 Noiseless and Error-Free Environments

Let us analyze how random neighbor selection schemes change R of RNM, compared to SM and GKL consensus in asynchronous noiseless and error-free environments.

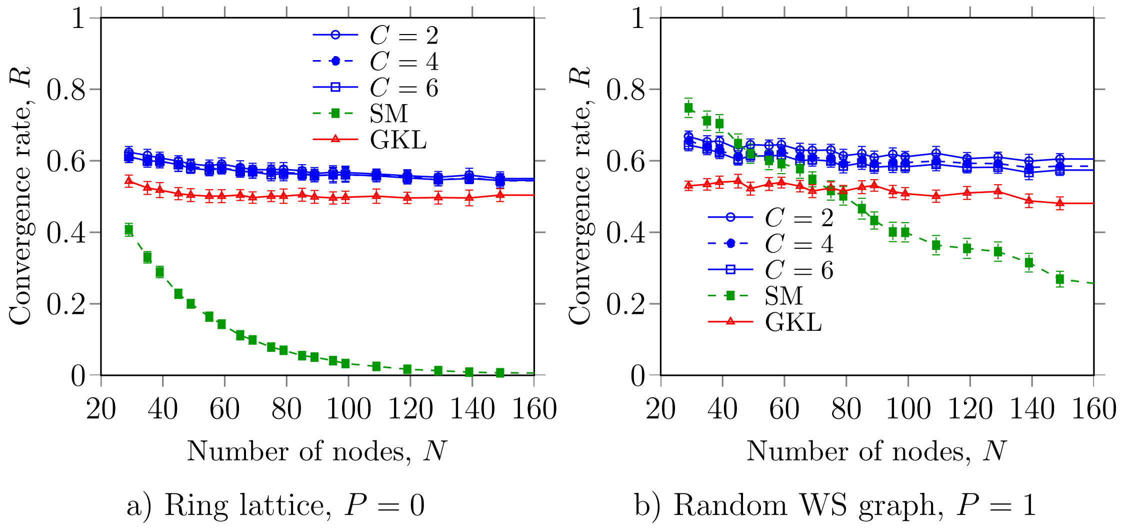


Figure 5.4: Asynchronous RNM, GKL and SM, in noiseless WS networks. $K = 3$, $N \in \{29, \ldots, 160\}$.

We start with WS graphs where the rewiring probability P enables to produce networks ranging from ring lattices ($P = 0$) to fully random networks ($P = 1$). In such networks an update sequence can be established and update-biased neighbor selection can be used.

We focus on RNM with $C \in \{2, 4, 6\}$ randomly selected neighbors, which is lower than the number of active neighbors ($2K = 6$) of GKL and SM. This can exhibit whether a random neighbor selection can be beneficial for consensus with lower number of received state messages.

Figure 5.4 shows the convergence rate of RNM, SM and GKL with a biased neighbor selection in noiseless and error-free WS networks of different size. Results indicate that asynchronous RNM outperforms SM and GKL in both ordered and random networks. They also show that, unlike SM, RNM does not significantly degrade with larger N. This is due to the embedded randomization that hinders the network clustering. RNM randomized solely by neighbor selection shows stable

R independent of system size outperforming the best algorithms in ordered and WS networks (best-performing solutions are described in Table 2.1).

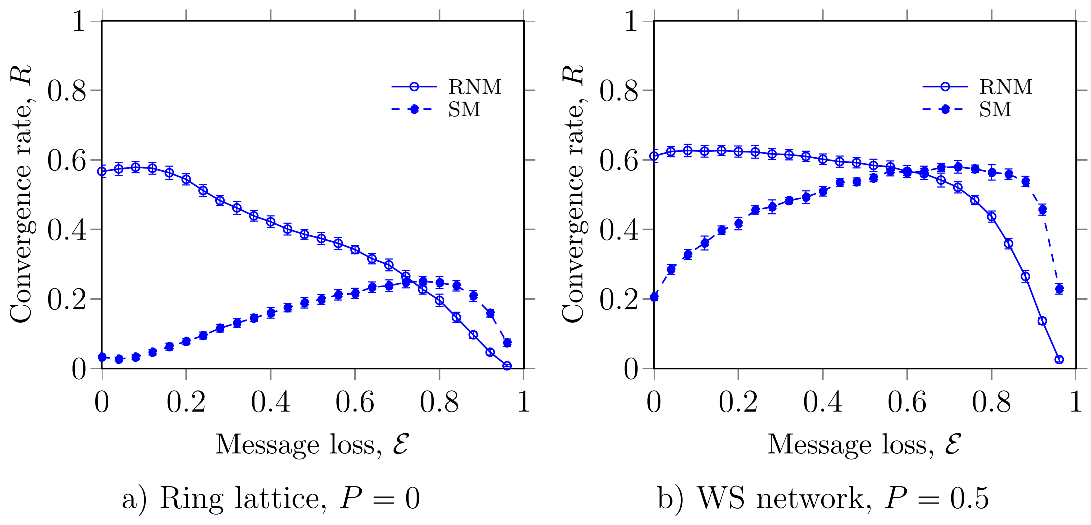


Figure 5.5: Asynchronous RNM, response to message loss in noiseless WS networks. $K = 3$, $N = 99$.

Figure 5.5 shows RNM in comparison with SM in noiseless networks with message loss. It indicates that RNM outperforms SM with message loss below 70% in ordered networks and message loss below 60% in randomized WS networks. We can generalize these new observations as follows. RNM outperforms SM and GKL in asynchronous ordered and randomized networks. Maximum R achieved by RNM is lower than maximum R of the best algorithms for binary majority consensus (see Table 2.1). Further we present results for RNM with additional randomization by noise or errors, showing that such cumulative randomization can increase its R to $\simeq 100\%$.

5.4.2 Noisy and Erroneous Environments

Let us consider the efficiency of RNM with additional randomization by noise and errors. We use the probability of error $\eta \in \{0.05, 0.06, 0.07, 0.08\}$ and fix the noise amplitudes at $A \in \{2, 3\}$. Error probability and noise are chosen empirically to provide the optimum randomization.

Figures 5.6 — 5.12 show the dynamics of Random Neighbor Majority with enforced noise and errors in networks with message loss, noise and topology randomization. Figure 5.6 illustrates the convergence rate of RNM with different levels of additive noise and errors in ring lattices and random networks. It shows

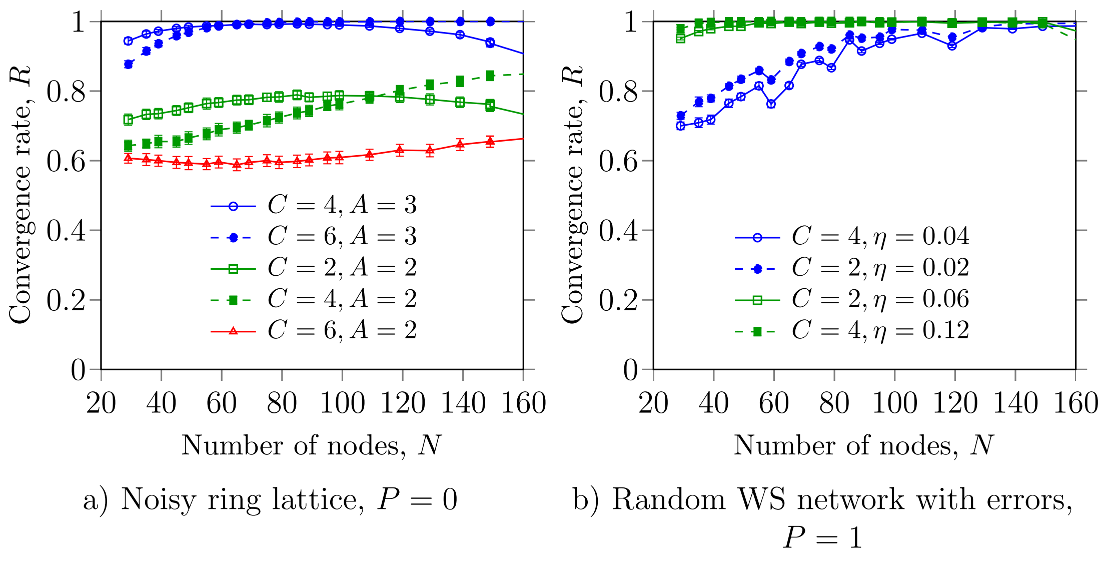


Figure 5.6: Asynchronous RNM, with $C \in \{2, 4, 6\}$ and $A \in \{2, 3\}$ in ring lattices and random networks. $K = 3$, $N = 99$.

that RNM with $C \in \{4, 6\}$ randomly selected neighbors and magnitude of enforced noise $A \in \{2, 3\}$ can reach $R \simeq 100\%$ in ordered networks. It also shows that with such $C \in \{4, 6\}$ and $\eta \in \{0.2, 0.4, 0.6, 0.12\}$ RNM can reach $R \simeq 100\%$ in random WS networks.

This shows that positive effects of random neighbor selection in RNM can be complemented by additional randomization by noise and errors to reach high convergence rate in both ring lattices and random networks. It also indicates that an optimal level of noise or errors can be selected for maximum convergence rate of RNM, depending on system conditions.

However, such combined randomization can reach certain “saturation” when stochasticity outweighs useful information exchange. The following analysis shows that this can happen if an algorithm with optimal level of noise or errors is additionally influenced by other stochastic disturbances. Thus, Figure 5.7 presents convergence rate of RNM in WS networks with growing topology randomization. It shows that topology randomization can inhibit RNM with the optimal randomization by errors, but also can promote the RNM with sub-optimal erroneous randomization until the optimal randomization level is reached. It also indicates that larger C increases maximum R while larger η can promote robustness towards topology randomization.

Figure 5.8 presents R of noisy RNM in ring lattices ($P = 0$) and random WS networks ($P \in \{0.5, 1\}$), indicating that in ring lattices message loss always inhibits RNM, while in randomized networks it can promote consensus.

Noisy RNM with $C = 2$ selected neighbors RNM reaches optimum performance

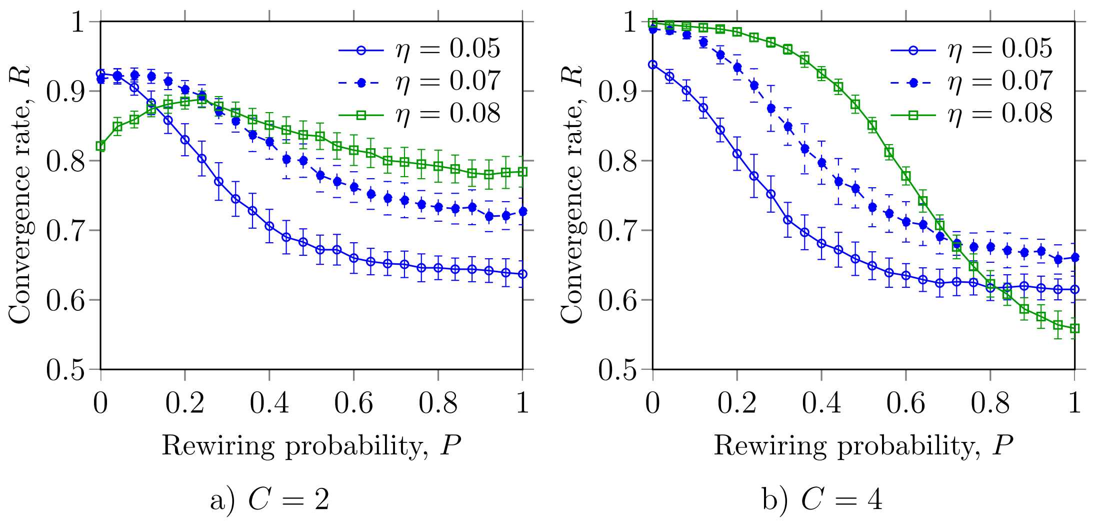


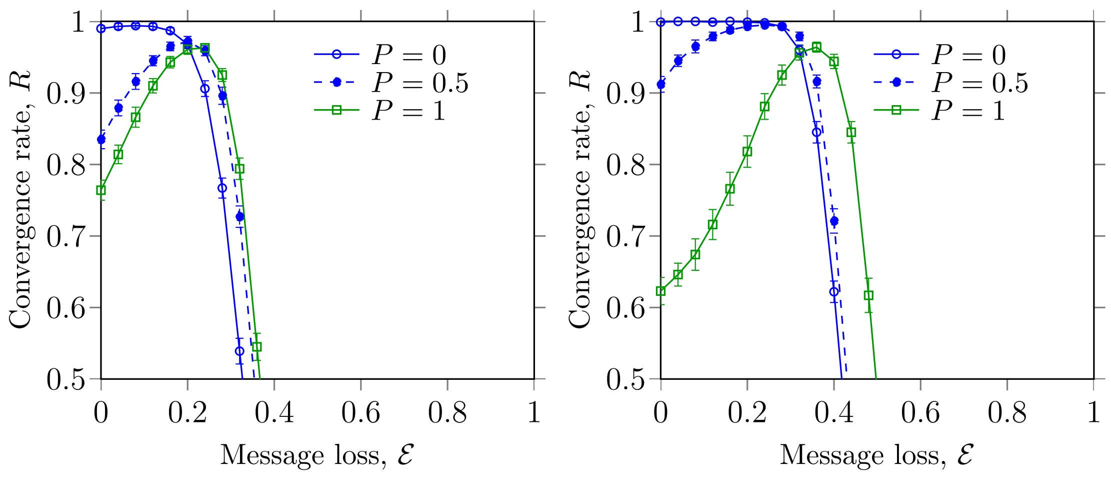
Figure 5.7: Asynchronous RNM with noise, response to topology randomization in WS networks. $K = 3$, $N = 99$.

in ring lattices at noise amplitude $A = 2$, and with $C = 4$ the optimum noise amplitude is $A = 3$. This can be explained as follows. Topology randomization ($P \geq 0$) disrupts the update sequence and inhibits the update-biased neighbor selection, so that RNM requires higher levels of additional randomization to reach an optimum performance. Message loss can provide such an additional stochasticity until its randomizing effect and message loss effect itself outweigh the useful information exchange. With this cumulative additional randomization the range of optimum performance is narrowed — an effect, similar to that earlier observed in Sections 4.3.6 and 4.3.8.

Figure 5.8 also indicates that noisy RNM strongly decreases R with message loss $\mathcal{E} \geq 50\%$. This can be explained by the randomization saturation, similar to that observed in the previous chapter and decrease of information exchange induced by message loss itself.

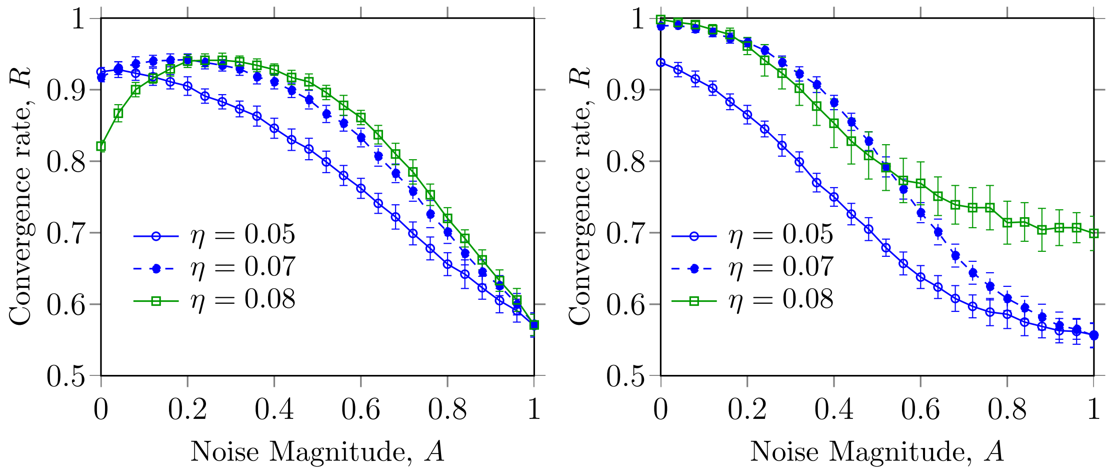
Figure 5.9 presents R of erroneous RNM in ring lattices with noise. It indicates effects, similar to that of Figures 5.7 and 5.8: larger C and η can promote consensus and robustness towards noise respectively.

Figure 5.10 shows R of RNM compared to SM in WS networks with combined influence of randomization by noise, topology, and message loss. It illustrates that RNM with additional randomization by noise shows the convergence rate $R \simeq 100\%$ with message loss $\mathcal{E} \leq 50\%$. It indicates that although RNM also reaches "saturation" of combined randomizing influences, it is more robust towards message loss and shows higher R than SM and GKL (for GKL, see Figures 4.10).



a) $C = 2, A = 2$ b) $C = 4, A = 3$

Figure 5.8: Asynchronous RNM with noise, response to message loss in WS networks. $K = 3$, $N = 99$.

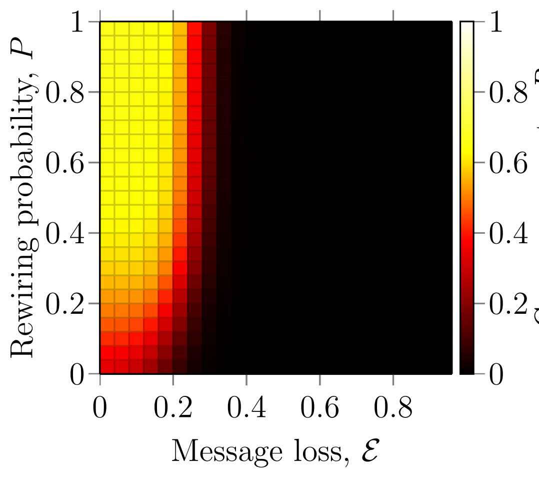


a) $C = 2$ b) $C = 4$

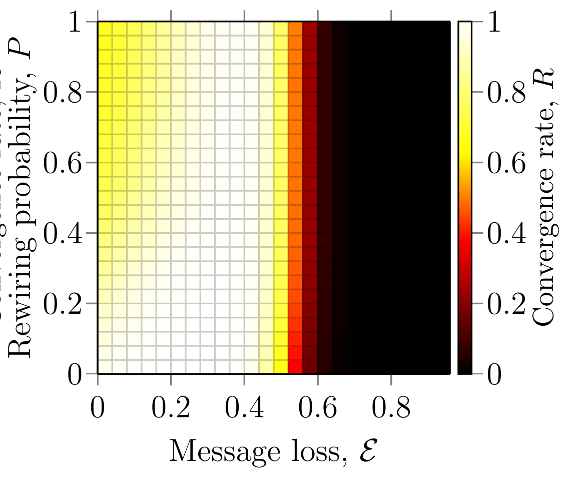
Figure 5.9: Asynchronous RNM with errors, response to noise in WS networks. $K = 3$, $N = 99$.

5.4.3 Waxman Networks

Let us compare asynchronous RNM and SM in Waxman networks with noise and message loss. Recall that Waxman networks are initially random and thus the update sequence cannot be established. In such networks the uniform or "balanced" neighbor selection can be beneficial, when neighbors are randomly selected from the set of all available neighbors. Figure 5.11 shows convergence rate



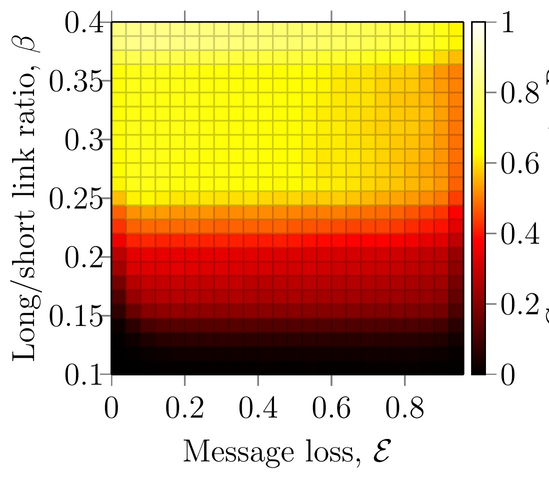
a) SM with noise, $A = 3$



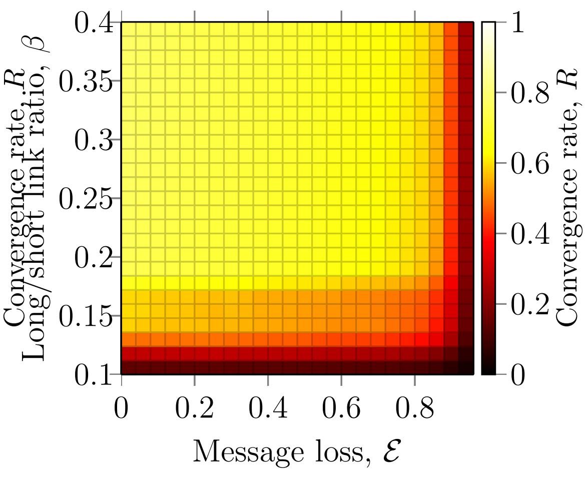
b) RNM, $C = 4, A = 3$

Figure 5.10: Asynchronous SM and RNM, response to message loss and topology randomization in noisy WS networks. $K = 3$, $N = 99$.

of RNM with such balanced neighbor selection in comparison to SM in noiseless Waxman networks.



a) SM



b) RNM, $C = 4$

Figure 5.11: Asynchronous SM and RNM in noiseless Waxman networks. $K = 3$, $N = 99$.

It shows that RNM with balanced random neighbor selection outperforms SM in Waxman networks and shows higher robustness towards message loss. It also indicates that such a random neighbor selection can promote consensus in networks with low connectivity ($\beta \leq 0.25$).

Next, Figure 5.12 shows performance of RNM in noisy Waxman Networks with message loss. Comparison of Figures 5.12a and 5.12b illustrates that in noisy networks RNM again outperforms SM with message loss below 50%. It shows that added noise increases R of RNM in networks with even lower connectivity ($\beta \geq 0.1$), but also decreases robustness towards message loss. These effects can be explained as follows. In Waxman networks with low connectivity (β) there exist clusters of nodes weakly connected to the remaining network. In Section 4.3.2, we show that noise can destabilize clusters by randomizing the information exchange. This latter effect of noise can additionally promote consensus in weakly connected Waxman networks.

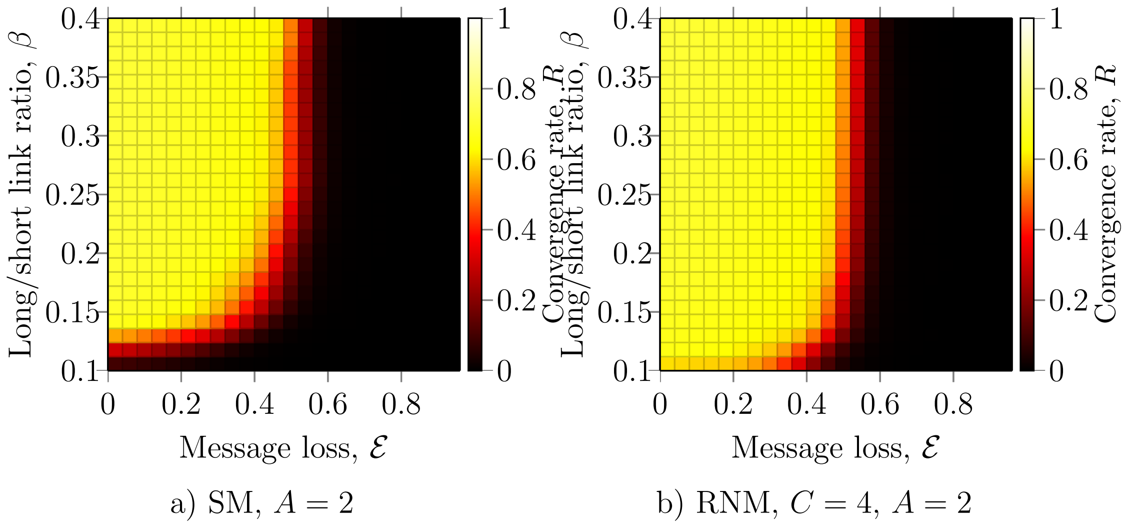


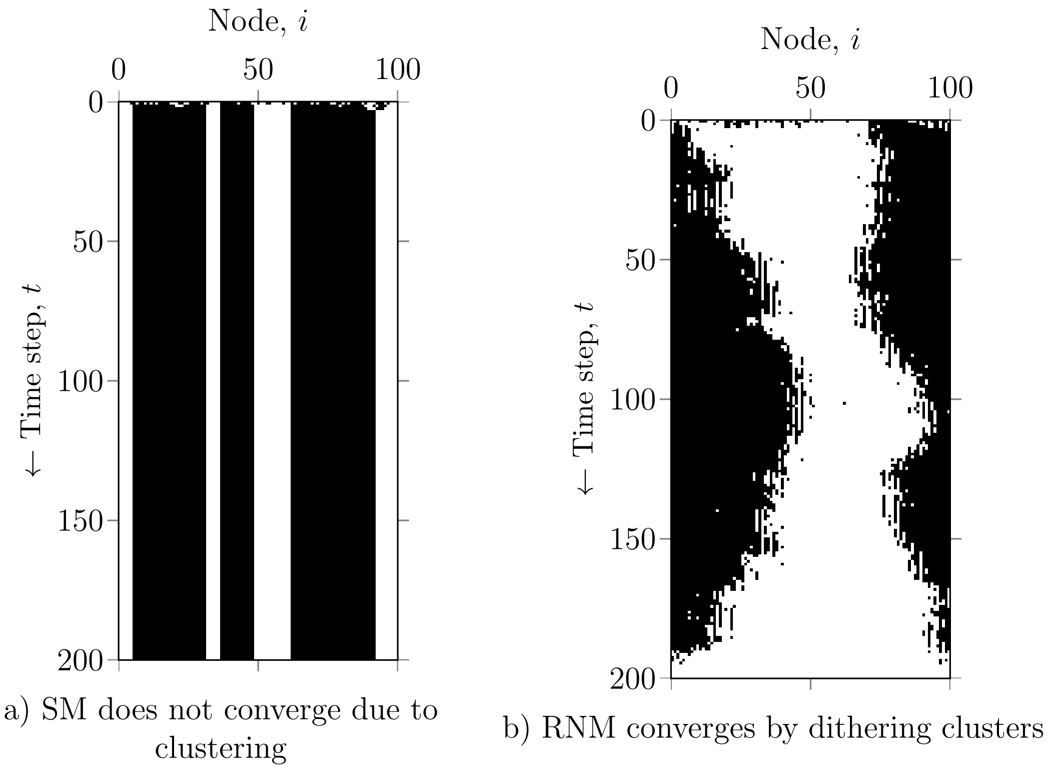
Figure 5.12: Asynchronous RNM and SM with message loss in noisy Waxman networks. $K = 3$, $N = 99$.

5.4.4 Analysis of the System Evolution

In this section we analyze state and density evolution in asynchronous networks with RNM to explain the mechanisms behind its high convergence rate. Figures 5.13 and 5.14 show state and density evolution of RNM and SM in noisy WS and Waxman networks.

Figure 5.13 presents an example of system evolution with asynchronous RNM, resembling state evolution of SM with randomization, presented in Figures 4.5 and 4.7. It illustrates that embedded randomization by neighbor selection and noise can dither clusters of nodes with the same states and promote consensus.

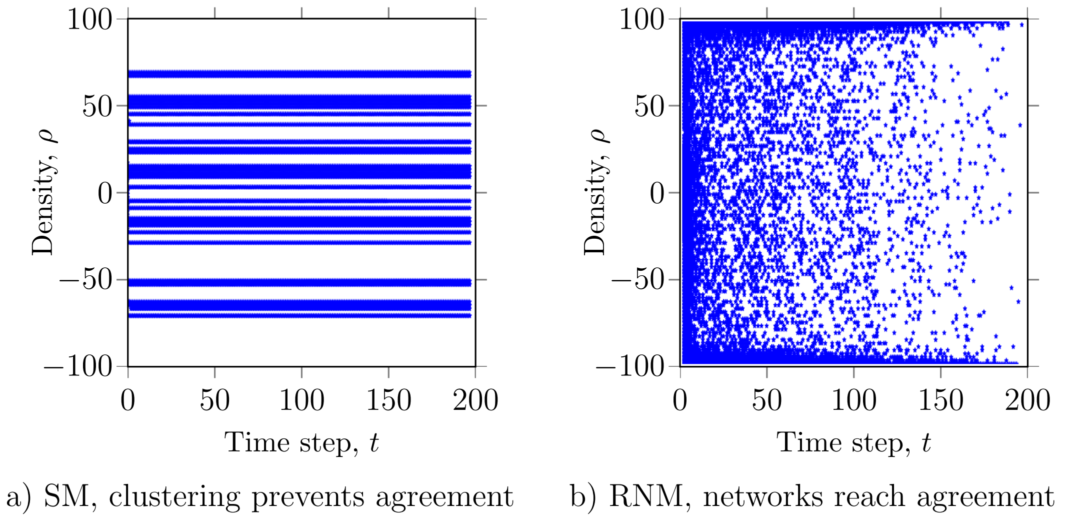
Larger system overview by density evolution over 1.000 initial configurations is presented in Figure 5.14 (each dot corresponds to a system evolving over the



a) SM does not converge due to clustering

b) RNM converges by dithering clusters

Figure 5.13: Examples of state evolutions of asynchronous SM and RNM, $K = 3$, $N = 99$, $A = 2$.



a) SM, clustering prevents agreement

b) RNM, networks reach agreement

Figure 5.14: Density evolution of asynchronous SM and RNM ($A = 2, C = 4$) in ring lattices, $K = 3$, $N = 99$.

unique initial configuration). It shows that RNM with noise converges to the correct majority (Figure 5.14b) while noiseless SM tends to cluster around stable densities (Figure 5.14a). This indicates two important observations: (a) RNM with noise or errors can provide $R \simeq 100\%$ in ordered and randomized systems and (b) RNM cannot guarantee successful convergence within a given consensus time T. Latter effect is due to a significant measure of random switching illustrated by Figure 5.14b. Unlike the direction bias of GKL (see Figure 4.3a and 4.3b) random switching does not steadily steer the system to a stable state of correct majority, but allows for stochastic state changes.

5.5 Summary

In Chapter 4 we show that SM can be significantly promoted by randomizing disturbances, but in ordered noiseless grids it indicates low R. GKL, however, shows the opposite dynamics, with high R in noiseless ring lattices and low R in randomized ones.

In this chapter we propose a Random Neighbor Majority consensus that embeds a local-scope topology randomization by random neighbor selection. We show that RNM outperforms GKL and SM in asynchronous networks. We then show that RNM with additional randomization by an optimal level of noise and errors can reach efficiency close to 100%. We study RNM in asynchronous ordered and random networks with noise and message loss. We show that RNM can achieve $R \simeq 100\%$ convergence rate in both ordered and randomized networks. We also show that these results are robust towards message loss, noise, and topology randomization. This can be generalized as follows. RNM can achieve efficiency of 100% in most types of randomized and disturbance-free environments with optimal selection of neighbor selection scheme, number of neighbors C, and additional randomization by noise and errors.

Although RNM with additional randomization by noise and errors can achieve convergence rate close to 100%, convergence analysis shows that strong randomization yields a significant measure of stochastic convergence. Stochastic convergence is not steady and cannot guarantee a stable agreement on a correct majority. Therefore, if a network is given more time, it can leave the agreed state. The future challenge is, therefore, to design an algorithm that can not only provide convergence but also can "steadily steer" the network to a correct state. This can be tackled with a more complex solution that accounts for a system evolution dynamics with, e.g., adaptively decreasing amplitude of noise.

The main contributions of this chapter can be generalized as follows:

- we show that randomization by neighbor selection can increase the efficiency of consensus without altering the underlying network topology;

- we propose two neighbor selection schemes that can increase the efficiency in both random and ordered environments;
- we show that an optimal level of additional randomization by noise and errors can be selected to accompany random neighbor selection, depending on the system conditions.

These new results show that random neighbor selection can strongly promote consensus in various types of networks and, even though it does not guarantee convergence with explicit randomization by noise or errors, it has a strong potential for future investigation. An intuitive suggestion on how to approach this would consider the adaptive change of the embedded randomization. A scheme that would provide a higher level of disturbances in the beginning of the network evolution that is relaxed towards the time limit. Obviously, such a scheme requires additional research. We did not yet cover the issue of robustness of consensus algorithms towards faulty node behavior, mentioned in Chapter 4: it is addressed in the next chapter.

CHAPTER 6 Consensus Algorithms with Faulty Nodes

6.1 Introduction and Motivation

Algorithms for distributed decision making, operating in real-life systems should be robust towards various disturbances. Studies on robustness of consensus algorithms investigate the influence of noise [Asp00, CFF$^+$07], message loss [KM07], random topologies [KB06], and faulty node behavior [US04]. Faulty nodes are often considered as one of the main impediments to consensus [PSL80], as even a single faulty node in a network can prevent agreement [FLP85, FLM85].

An early study by Pease *et al.* [PSL80] shows that in a *synchronized* networked system of N nodes, M of them being faulty, consensus is possible if $M < \frac{N-1}{3}$. However, Fischer *et al.* [FLP85] show that in *asynchronous* systems consensus may become impossible with already $M = 1$. Such faulty nodes are generally represented as Byzantine faulty nodes — nodes that can have any arbitrary failure, except full failure. Later studies compare the impact of Byzantine faulty nodes with dormant faulty nodes [MP91] and crash-failing faulty nodes [AFJ06], showing that consensus can stabilize and overcome the impact of such non-Byzantine nodes.

Scholars approach the problem of fault tolerance with fault-detection [CT96, CHT96], increasing system-wide synchrony [DDS87, DLS88], and randomization [BO83, Asp03]. Randomization is the technique that utilizes random processes (that are often considered as negative disturbances) to increase fault tolerance [Asp03]. Compared to fault-detection, randomization is less complex and does not require system-wide adjustments, e.g., as imposed system-wide synchrony. Investigations show that randomization can be beneficial for various consensus algorithms both in terms of convergence rate [Asp03, MMDA04] and fault tolerance [BO83, Asp03]. Such randomization, e.g., for binary majority consensus, can be provided explicitly by noise [5, 4] or errors [MMDA04].

Randomized consensus algorithms show higher robustness towards faulty node behavior [MNC10] than deterministic solutions. Some algorithms can guarantee the agreement in systems with $M \leq \frac{N}{2}$ faulty nodes in the system [BO83]. Such algorithms provide convergence rate of 100%, but often have relaxed connectivity or time restrictions [BO83].

Binary majority consensus has more strict termination requirements than general consensus considered in the aforementioned studies, and can be more sensitive to the faulty node behavior. This motivated us to investigate the impact of faulty nodes on binary majority consensus. In this chapter we study faulty nodes with persistent and random failure, and different layout over the network. We study faulty nodes influence in ring lattices, random Watts-Strogatz [WS98], and Waxman [Wax88] networks with noise, errors, and message loss.

We show that:

- performance decrease induced by faulty nodes can be mitigated by randomization of different origin;
- commonly-used faulty nodes with random failure are less adverse for consensus than faulty nodes with persistent failure;
- in some cases faulty nodes with random failures can provide randomization beneficial for consensus.

The chapter is organized as follows. Section 6.2 explains system modeling and adjustments. Section 6.3 presents simulation results and analysis. Finally, Section 6.4 concludes the chapter with summary of results. Main contributions of this chapter are published in [3, 2, 5].

6.2 System Model

We investigate consensus with faulty nodes in ordered networks, Watts-Strogatz and Waxman networks. Ordered networks modeled as ring lattices, set up as described in Section 3.3.1. Random Watts-Strogatz and Waxman networks are modeled as described in Sections 3.3.2 and 4.2.1, respectively.

6.2.1 Randomization by Noise, Message Loss, and Topology

We introduce noise and message loss as described in Sections 4.2.2 and 4.2.2, respectively. Randomizing disturbances are applied as described in Section 4.2.3. Randomization by topology is provided by underlying network topology in WS and Waxman networks, as described in Sections 3.3 and 4.2.1, respectively.

6.2.2 Consensus Algorithms

We study the impact of faulty nodes on three consensus algorithms: Simple Majority (SM), Gacs-Kurdyumov-Levin (GKL), and Random Neighbor Majority (RNM). We modify these algorithms to perform in randomized systems as described in Sections 4.2.1, 4.2.1 and 5.3, respectively.

To accommodate random disturbances the system model has undergone adjustments described in Section 4.2.1. Further details on the system modeling and the implementation of faulty nodes can be found in Chapter 3.

6.2.3 Faulty Nodes

We study faulty nodes with two failure models: faulty nodes with *random failure*, modeled after Byzantine failure model, and faulty nodes with *persistent failure*.

We implement faulty nodes as follows. At the starting time $t = 0$, M faulty nodes are added to N non-faulty nodes to avoid a bias of the initial configuration. The network topology is then created for all $N+M$ nodes. After adding M faulty nodes to the system they are labeled as faulty and counter consensus according to their failure model.

The Layout of Faulty Nodes

We use two layouts of faulty nodes over the network: clustered and distributed. With a clustered layout all faulty nodes are located next to each other. The location of the cluster is randomly chosen at each simulation run. With a distributed layout all faulty nodes are randomly placed over the network, independently from each other. Further details on the layout of faulty nodes can be found in Section 3.5.

Faulty Nodes with Persistent Failure

Faulty nodes with persistent failure (persistent faulty nodes) are modeled as follows. After M faulty nodes are added, they are assigned with a faulty value σ_M, opposite to the initial majority. I.e., if $\sum_{i=0}^{i=N} \sigma_i[0] < 0, \sigma_M = 1$ and if $\sum_{i=0}^{i=N} \sigma_i[0] > 0, \sigma_M = -1$. During the consensus process such faulty nodes broadcast their states but do not update them. Unlike faulty nodes with random failure, persistent faulty nodes provide enduring inhibition for consensus.

Faulty Nodes with Random Failure

We implement faulty nodes with random failure (non-persistent faulty nodes) after Byzantine random failure model with a reduced state space. Such nodes randomly change their broadcasted state, independently from state information received from their neighbors. We investigate two types of faulty nodes with reduced state space:

- two-state faulty nodes, randomly switching between states $\sigma_M \in \{-1, 1\}$;
- three-state faulty nodes, switching between $\sigma_M \in \{-1, 0, 1\}$.

The first case presents a faulty node that broadcasts correct and erroneous state information with equal probabilities. The second case additionally implements a state of sending no information, i.e., full failure. Full failure, or a "crash" failure is often considered as a separate type of failure [Fis83] which, depending on a system design, can be less adverse than Byzantine failure. Impact of fully crashed nodes added to the system is predictably negative, as it will decrease of effective average connectivity (as non-faulty nodes will obtain "useless" links instead of active ones). In Chapter 4 we show that a random message loss can promote consensus, and randomly crashing faulty nodes, added to the system can be represented as a local disturbance with a similar impact. In our simulations we incorporate the crash failure into a state space of randomly failing faulty nodes to exhibit whether there is a difference, compared to a two-state random failure.

6.2.4 Consensus Termination with Faulty Nodes

We consider the system to be converged if all N non-faulty nodes have agreed on a state that corresponds to initial majority within the consensus period $T = 2N$ time steps. Detailed description of termination condition can be found in Section 3.6.

6.3 Performance Analysis with Faulty Nodes

We investigate the impact of faulty nodes on binary majority consensus in networks with randomization. In the following sections we consequently analyze the impact of:

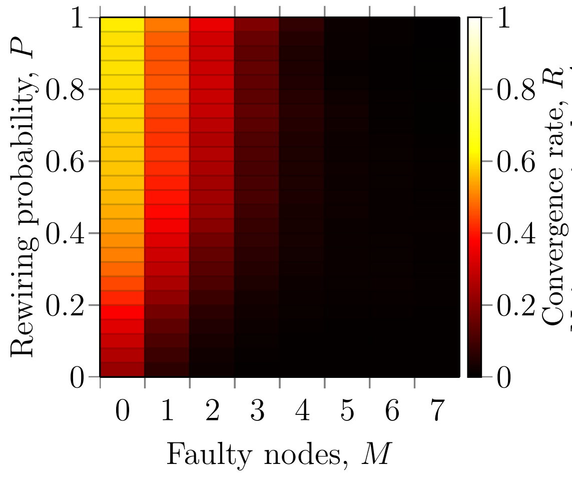
- persistently failing faulty nodes with randomization by noise, message loss and topology,
- randomly failing faulty nodes;
- faulty nodes positioning over the network;
- faulty nodes with random failures in asynchronous ring lattices.

As the main performance metric of the algorithms we employ the convergence rate R as described in Section 2.3. We use initial configurations, obtained by a coin-flip trial, described in Section 3.7.

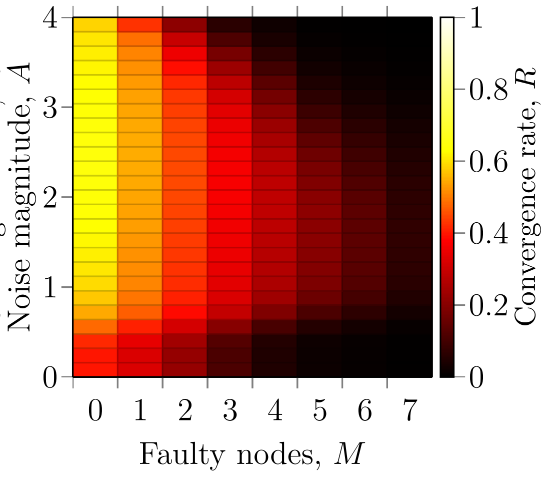
In the following section we analyze the impact of faulty nodes with persistent failure. In Section 6.3.2 we study the impact of persistently failing nodes in Waxman networks with additive noise and stochastic message loss. In Section 6.3.3 we compare the impact of persistent faulty nodes with commonly-used non-persistent faulty nodes. In Section 6.3.4 we investigate whether a random placement of the faulty nodes can mitigate their impact. In Section 6.3.5 we investigate the effect of GKL consensus promotion by randomly failing faulty nodes and explain its mechanism. Finally, Section 6.4 concludes the chapter.

6.3.1 Impact of Faulty Nodes with Persistent Failure

Figure 6.1 shows R of asynchronous SM with persistently failing faulty nodes. It shows that noise and topology randomization can promote robustness of SM consensus towards faulty node behavior. This important observation indicates that "de-clustering" influence of randomization by topology and noise not only increases R in systems with $M = 0$, but can contribute to a higher R in systems with faulty nodes.



a) Response to topology randomization with faulty nodes in noisy networks, $A = 3.5$



b) Response to additive noise with faulty nodes in random WS network, $P = 1.0$

Figure 6.1: Convergence rate of asynchronous SM with M faulty nodes. Noise and topology randomization in WS networks. $K = 3$, $N = 99$.

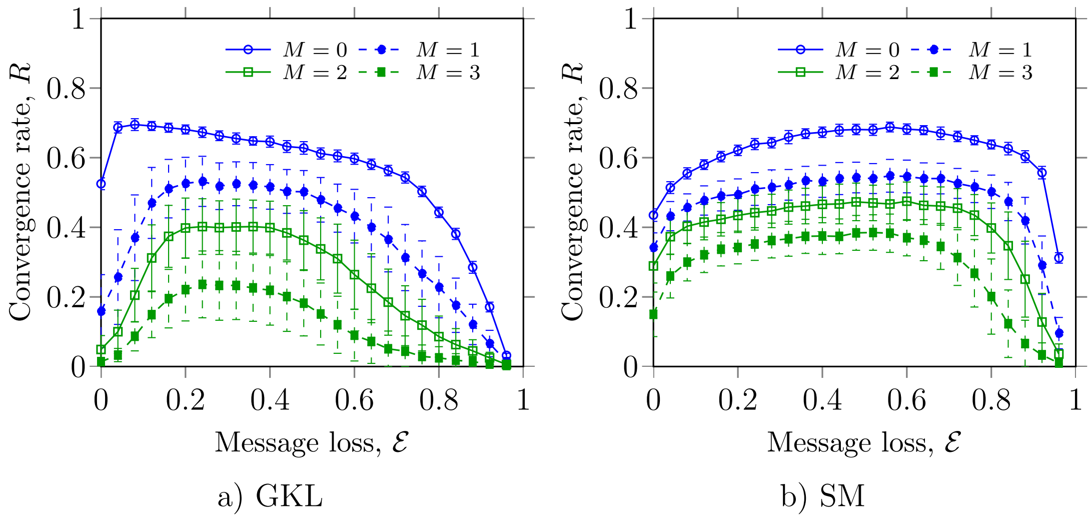


Figure 6.2: Convergence rate of asynchronous SM and GKL with M faulty nodes and message loss in random WS networks, $P = 1$, $K = 3$, $N = 99$.

Figure 6.2 indicates a similar impact of message loss: in topologically randomized environments it can increase R of SM and GKL with faulty nodes. Latter effects can be explained as follows. Topology randomization in WS networks connects a faulty node with random neighbors enabling the latter ones to overcome the reduced negative impact.

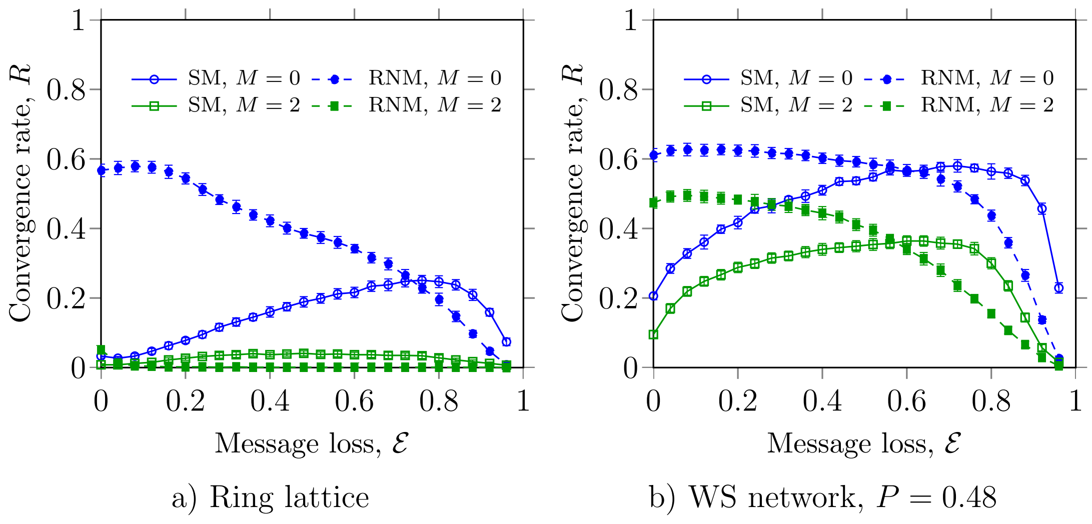


Figure 6.3: Asynchronous RNM and SM, response to message loss in noiseless WS networks with faulty nodes. $K = 3$, $N = 99$, $C = 4$.

Additive noise and message loss mitigate the negative impact of each faulty node and promote consensus in a similar manner.

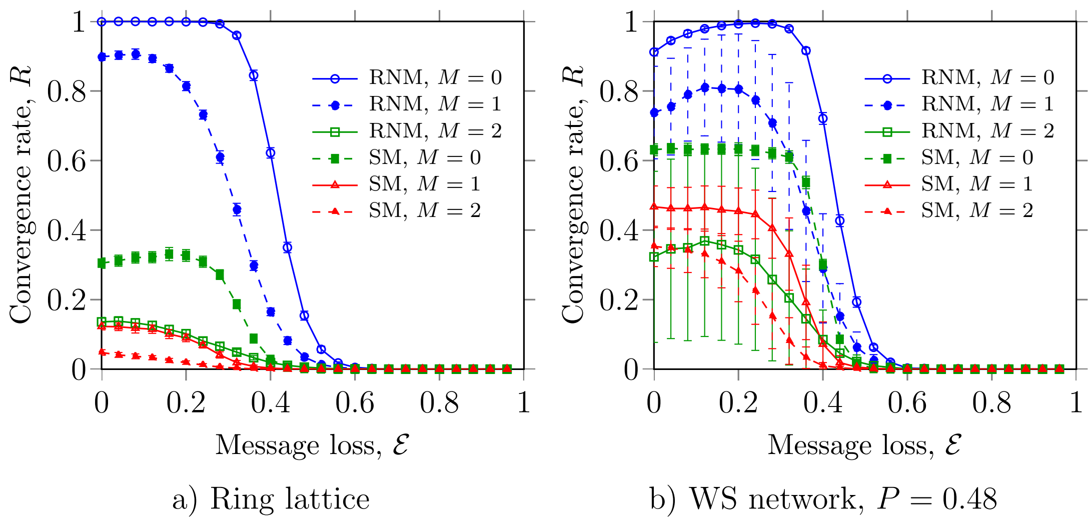


Figure 6.4: Asynchronous RNM and SM, response to message loss in noisy WS networks with faulty nodes. $K = 3$, $N = 99$, $A = 3$, $C = 4$.

Simple Majority and Random Neighbor Majority

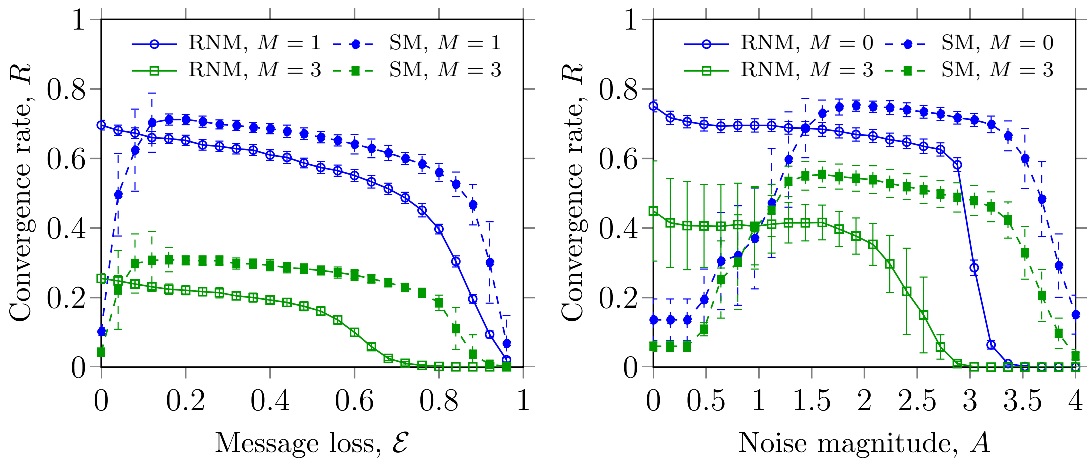
Figures 6.3 and 6.4 show R of SM in comparison to RNM. Figure 6.3a shows that although in noiseless ring lattices RNM outperforms SM with message loss $\mathcal{E} \leq 70\%$, with presence of faulty nodes both algorithms drastically decrease the convergence rate.

Figure 6.3b, however, illustrates that in topologically randomized networks, RNM outperforms SM both with and without faulty nodes, with message loss $\mathcal{E} \leq 30\%$ and $\mathcal{E} \leq 60\%$, respectively. This can be explained by the nature of RNM itself — random neighbor selection at each time step can mitigate the impact of a faulty node to its neighbors, as they receive less incorrect state information.

Figure 6.4 shows convergence rate of RNM and SM with faulty nodes in noisy ring lattices and WS networks. It indicates that in noisy networks with a single faulty node, $M = 1$, RNM outperforms SM. However, already $M = 2$ leads to a drastic decrease in R for both algorithms. Figure 6.4 compared to Figure 6.3 illustrates that the impact of additive noise is twofold: a) it increases R of the algorithms, and b) it decreases robustness towards message loss, which was observed earlier in Chapters 4 and 5. It can be explained by combined randomizing impact of noise and message loss that outweighs the useful information exchange. Analysis of Figure 6.4 shows that in random WS networks (Figure 6.4b) algorithms show higher variation in R than in ring lattices (Figure 6.4a). This can be explained by the positive impact of randomized topologies, discussed earlier.

6.3.2 Impact of Faulty Nodes in Waxman Networks

Figure 6.5 presents the convergence rate of SM and RNM with faulty nodes in weakly connected Waxman networks with noise and message loss. Figure 6.5a indicates that in a Waxman network with low message loss ($\mathcal{E} \leq 5\%$) RNM outperforms SM with faulty nodes. Figure 6.5b demonstrates that RNM outperforms SM with faulty nodes while the noise level is below $A \simeq 1$.



a) Response to message loss, $A = 0$ b) Response to noise, $\mathcal{E} = 0$

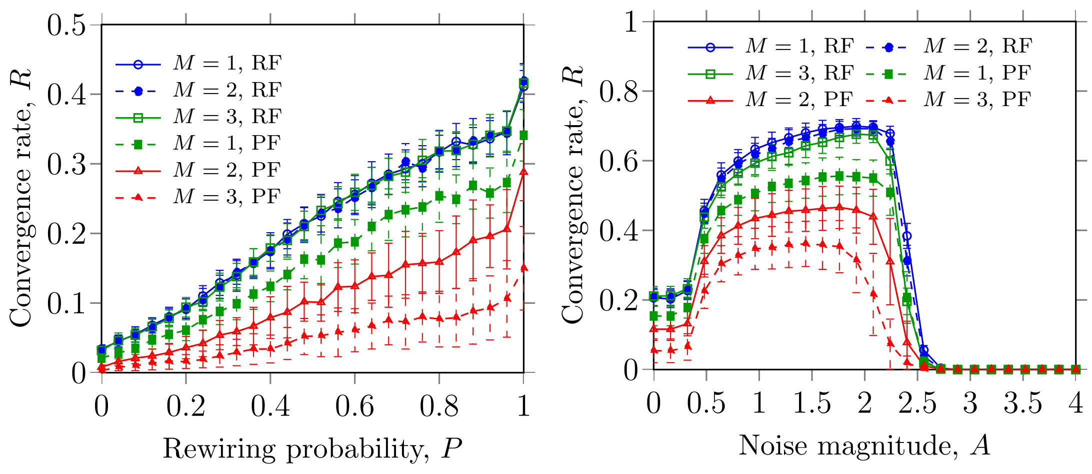
Figure 6.5: Asynchronous SM and RNM in loosely connected Waxman networks with faulty nodes. $N = 99$, $K = 3$, $\alpha = 0.05$, $\beta = 0.18$, $C = 4$.

Figure 6.5 shows that RNM is slightly less robust towards noise and message loss than SM. This effect is similar to that observed in WS networks and can be explained by the lower information exchange of RNM (four state information messages instead of six in SM), and the excessive randomization that decreases R. We omit GKL analysis in Waxman networks as it achieves $R \leq 10\%$ with $M = 0$ and $R \simeq 1\%$ with $M \geq 1$.

6.3.3 Comparison of Persistent and Non-persistent Faulty Nodes

Let us analyze the impact of commonly-used faulty nodes with random failure compared to faulty nodes with persistent failure. Again, we simulate SM and GKL in WS and Waxman networks with noise and message loss. Figures 6.6a and 6.6b show that randomization by topology and noise can promote robustness of SM consensus towards both types of faulty nodes.

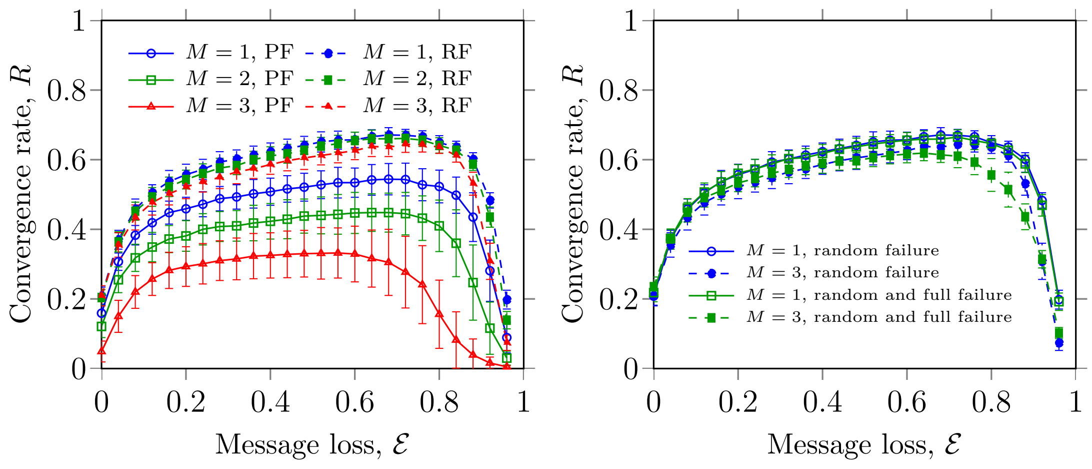
Figure 6.6 also shows that faulty nodes with persistent failure inhibit consensus stronger than faulty nodes with random failure.



a) Topology randomization in noiseless networks

b) Response to noise in random networks, $P = 1$

Figure 6.6: Asynchronous SM with M faulty nodes. Noise (AWUN) and topology randomization in WS networks. PF and RF stand for faulty nodes with persistent and random failure models, respectively. $K = 3$, $N = 99$.



a) Faulty nodes with persistent and random failure model

b) Faulty nodes with random and random with full failure model

Figure 6.7: Asynchronous SM with M faulty nodes and message loss in random WS networks ($P = 1$). PF and RF stand for faulty nodes with persistent and 2-state random failure models, respectively. $K = 3$, $N = 99$.

Figure 6.7 shows R of SM in random WS networks with message loss and non-persistent faulty nodes with two-state and three-state failure. Figure 6.7a shows that in random WS networks message loss can increase R of SM and GKL with faulty nodes of both types. Figure 6.7b indicates a small difference in the impact between faulty nodes with two-state and three-state random failure.

Figure 6.8 presents the convergence rate of SM in Waxman networks with randomization by noise and message loss.

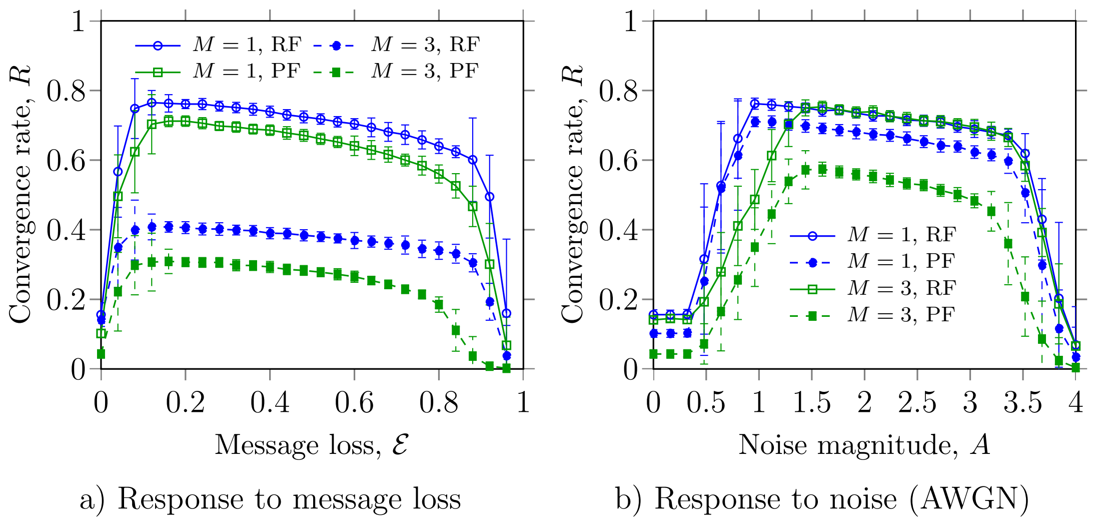


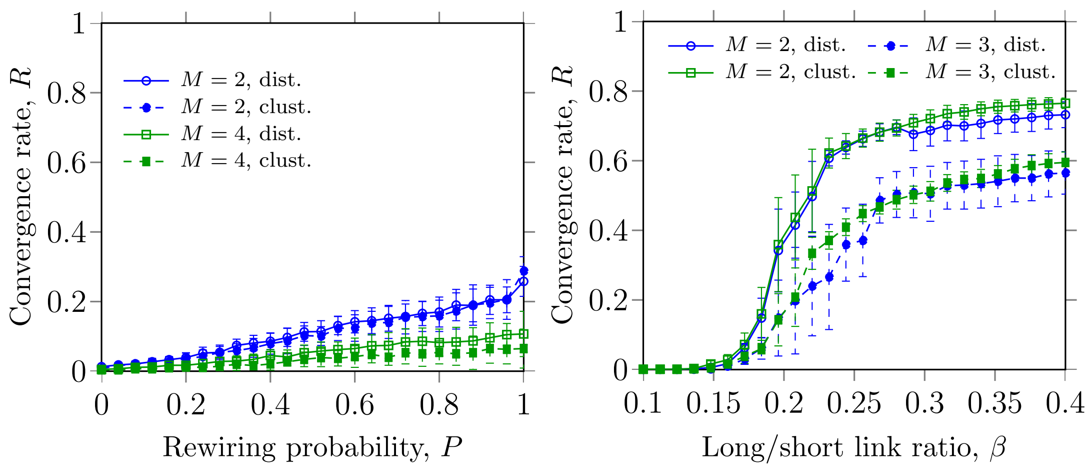
Figure 6.8: Asynchronous SM with M faulty nodes in loosely connected Waxman networks. PF and RF stand for faulty nodes with persistent and random failure models, respectively. $K = 3$, $N = 99$, $\alpha = 0.05$, $\beta = 0.18$.

It indicates that in Waxman networks faulty nodes with persistent failure inhibit consensus stronger than faulty nodes with random failure — the effect earlier observed for WS networks.

These observations indicate that persistent faulty nodes inhibit binary majority consensus stronger than commonly used faulty nodes with arbitrary Byzantine failure. This effect is observed with all types of randomization in ring lattices, random WS, and Waxman networks.

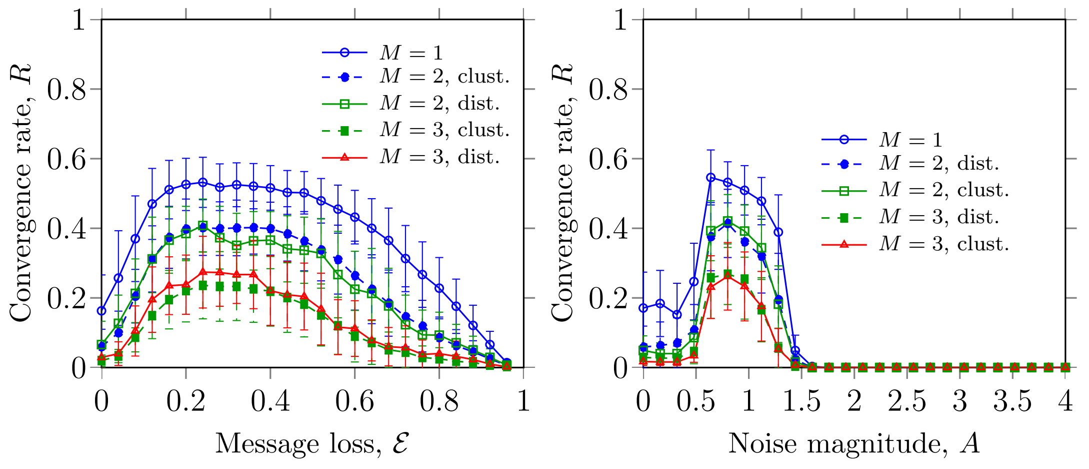
These effects can be explained by the nature of persistently failing faulty nodes: such nodes always send the state information that counters the consensus process. Faulty nodes with random failure can also send the correct information and, thus, contribute to a correct convergence process. Moreover, it was shown that random binary errors and message loss can provide beneficial stochasticity [MMDA04, 4, 2], which can further weaken the impact of non-persistent faulty nodes.

The stronger impact of persistent faulty nodes has the following effect. A number of persistent faulty nodes (e.g., $M = 2$) with both possible faulty states $\sigma_M \in \{-1, 1\}$ inhibit the a binary majority consensus system with an arbitrary initial density stronger than $M = 2$ non-persistent faulty nodes, randomly switching between two or three possible states. Therefore, to inhibit a binary majority



a) WS network b) Waxman network

Figure 6.9: Asynchronous SM with topology randomization and M faulty nodes. $K = 3$, $N = 99$.



a) Response to message loss b) Response to noise (AWUN)

Figure 6.10: Asynchronous GKL in random WS networks ($P = 1$) with M faulty nodes. “Clust.” and “dist.” stand for clustered and random faulty node placement. $K = 3$, $N = 99$.

consensus system with unknown initial density an intruder should rather use an equal number of persistent faulty nodes with both available σ_M (e.g., $M = 4$: 2 nodes with $\sigma_M = -1$ and 2 nodes with $\sigma_M = 1$) rather than a bigger number of non-persistent faulty nodes ($M = 4$ nodes with $\sigma_M \in \{-1, 1\}$ or $\sigma_M \in \{-1, 0, 1\}$).

6.3.4 Faulty Nodes with Random and Clustered Layout

In Figures 6.1 and 6.6 we observed that topology randomization can mitigate the negative impact of faulty nodes. This motivated us to determine whether a static random placement of faulty nodes can produce similar effect.

We simulate networks with two types of faulty nodes layout on the network where (a) all faulty nodes are located in a single cluster, and (b) faulty nodes are randomly placed over the network.

Figures 6.9-6.11 present R of asynchronous GKL and SM in WS and Waxman networks with persistent faulty nodes with random and clustered layouts.

Topology Randomization

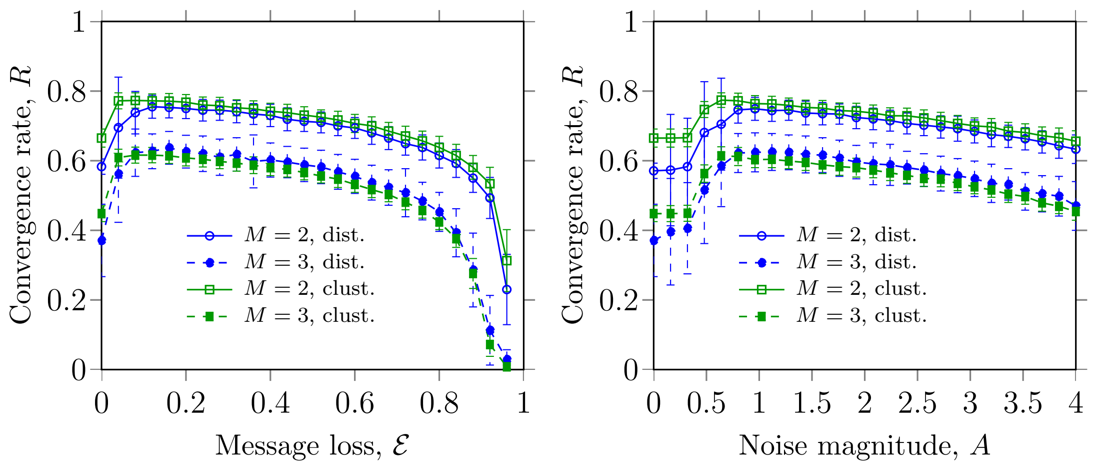
Figure 6.9 shows dynamics of the SM consensus with clustered and randomly placed faulty nodes in Watts-Strogatz and Waxman networks with topology randomization (increasing P and β).

Figure 6.9a indicates that in WS networks $M = 4$ faulty nodes with clustered layout inhibit consensus slightly stronger than faulty nodes randomly placed over the network. The observed difference in impact lies within the confidence intervals and thus cannot be considered as determinative. Further, Figure 6.9b does not indicate such difference in impact for Waxman networks. This difference in impact of clustered and distributed faulty nodes, observed for SM in WS networks can be explained as follows. SM with $M \geq K$ (Figure 6.9a) faulty nodes achieves $R \in (5, 10)\%$, and at this level of convergence the contribution of randomization can lead to serious fluctuations, which is partially indicated by the wide confidence intervals. This is further illustrated by Figure 6.10. It shows that in random WS networks ($P = 1$) with additional randomization by message loss faulty nodes with clustered and distributed layout have similar impact.

Randomization by Noise and Message Loss in Waxman Networks

In Waxman networks with additional randomization by noise or message loss the impact of clustered faulty nodes is similar to that of randomly placed ones, as can be seen from Figures 6.9b, 6.10, and 6.11. This can be explained by topology randomization that distributes the impact of the clustered faulty nodes into a wider set of nodes. This leads to “de-clustering” of the faulty nodes and mitigates the difference in impact with randomly placed faulty nodes.

This effect is observed with different types of randomization in both WS and Waxman networks, as can be seen from Figures 6.10 and 6.11. This can infer that the observed effect of increased robustness with faulty node random placement,



a) Response to message loss b) Response to noise (AWGN)

Figure 6.11: Asynchronous SM with M faulty nodes in connected Waxman networks. $K = 3$, $N = 99$, $\alpha = 0.05$, $\beta = 0.26$.

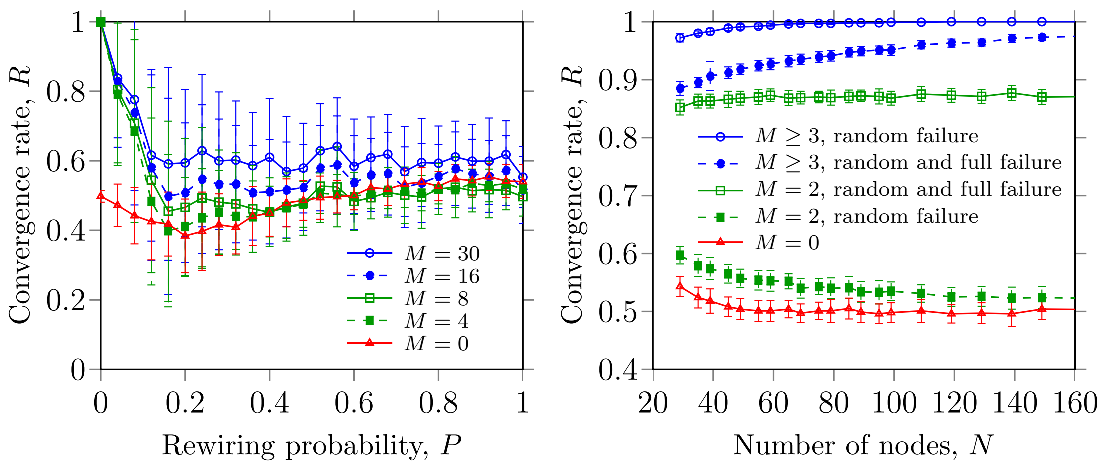
weakly expressed for asynchronous SM in WS networks, is a specific feature of such a setup.

6.3.5 Consensus Promotion with Non-Persistent Faulty Nodes

Moreira *et al.* [MMDA04] study binary majority consensus with binary errors, showing that such random intrusions can significantly promote consensus. Randomly failing faulty nodes with reduced state space essentially act as locally placed error generators, and can produce a similar effect. In our simulations we observe consensus promotion by randomly failing faulty nodes in ring lattices with asynchronous GKL. Figure 6.12 shows R of asynchronous GKL with clustered non-persistent faulty nodes in WS networks and ring lattices of different sizes. Figure 6.12a shows that $M \geq K$ of such faulty nodes located in a single cluster can significantly increase R in ring lattices ($P = 0$). Figure 6.12b shows that this effect scales with the system size and remains with three-state non-persistent faulty nodes.

Two-state non-persistent faulty nodes promote consensus stronger than three-state faulty nodes, although both types indicate similar dependencies.

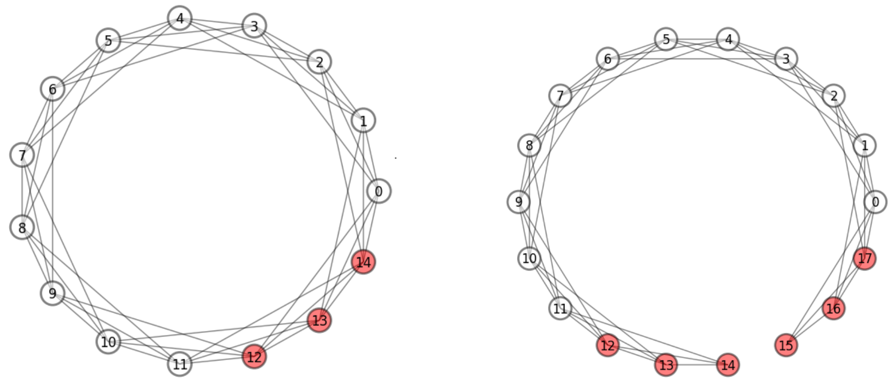
This can infer that randomization within the consensus state space can be more efficient [5, 1, MMDA04]. Positive impact of faulty nodes on GKL consensus can be explained by the explicit randomization they impose on the information exchange. It was previously shown that randomization by binary errors can promote consensus [MMDA04]. Consensus promotion by faulty nodes reaches maximum



a) Response to topology randomization in WS networks, $N = 99$

b) Response to system growth in ring lattices, $P = 0$

Figure 6.12: Asynchronous GKL with M clustered non-persistent faulty nodes. $N \in \{29, \ldots, 999\}$, $K = 3$.



a) The ring is logically disconnected by the cluster of $M = K$ faulty nodes

b) An actually disconnected ring

Figure 6.13: A ring lattice with $M = K$ clustered faulty nodes, $N = 15$, $K = 3$.

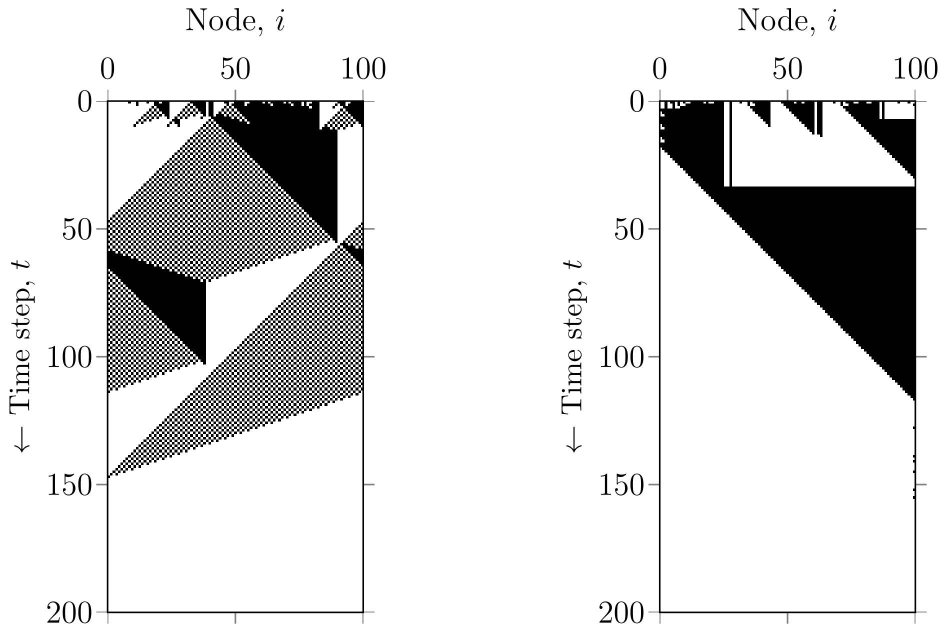
with $M \geq K$ faulty nodes allocated in a single cluster. Figure 6.13a) illustrates such a case: in a presented network, nodes 11 and 0 cannot access the state information from each other because the ring is logically disconnected by a cluster of faulty nodes. Figure 6.13b) shows similar situation in a ring that is actually disconnected: nodes 11 and 0 cannot access the mutual state information in a way similar to that of Figure 6.13a).

Such setup can be presented as an open one-dimensional lattice with M faulty nodes at both ends (see Figure 6.13b). This can also be interpreted as a solution accounting for boundary effects: consensus can be promoted by providing sufficient random inputs ($M \geq K$) on network borders.

Analysis of the System Evolution

The evolution of asynchronous GKL in networks with faulty nodes is presented in Figures 6.14 and 6.15.

Figures 6.14a and 6.15a show the state and density evolution of the synchronous GKL without faulty nodes over time, respectively.

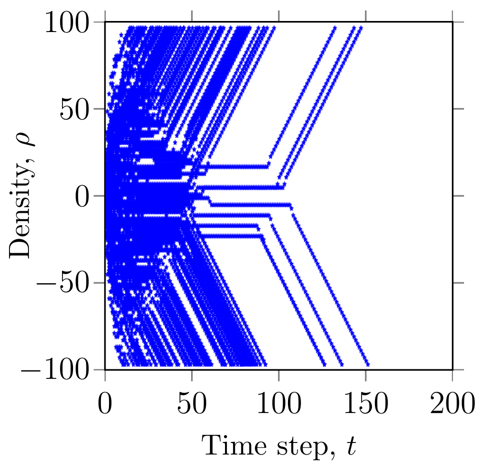


a) Synchronous GKL with $M = 0$, clusters migrate over the ring

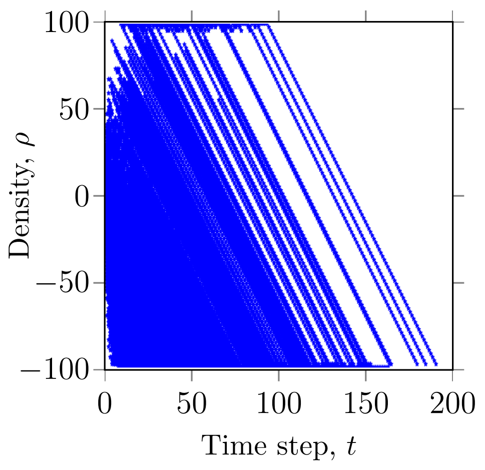
b) Asynchronous GKL with $M \geq K$, clusters are destroyed at the border of the ring

Figure 6.14: State evolution of GKL. $K = 3$, $N = 99$.

Figure 6.14a illustrates that in a connected synchronized ring clusters can migrate over the network. Figure 6.15a shows that $\simeq 82\%$ of all systems with various densities steadily evolve to the state of correct majority. Figures 6.14b and 6.15b show the state and density evolution asynchronous GKL with $M \geq K$ faulty nodes. Figure 6.14b shows that a cluster in a logically disconnected ring (see Figure 6.13) does not migrate and is destroyed faster. Figure 6.15b shows that in such a setup systems evolution is biased with an update sequence and that there exist random shifts of density.



a) Synchronous GKL, $\simeq$ 82% of networks converge with $M = 0$ by steady density evolution



b) Asynchronous GKL, $\simeq$ 100% of networks converge with $M \geq K$, density evolution is biased and disrupted by randomization

Figure 6.15: Density evolution of synchronous and asynchronous GKL in ring lattices. $K = 3$, $N = 99$.

This indicates that the state direction bias of the GKL combined with asynchronous updates can steer the system to the expected state. It also shows that density often evolves closely to the opposite majority, and then it is steered to the correct one. This happens due to the steering effect of the asynchronous update and the contribution of the faulty nodes state information messages.

Latter observations can be explained as follows. Even though asynchronous GKL with additional randomization by faulty nodes can reach $R \simeq 100\%$, it can not be considered as a solution to the binary majority consensus problem: the system exhibits significant random dynamics and cannot guarantee correct convergence in the given consensus time T.

6.4 Summary

In this chapter we investigate the influence of faulty nodes on binary majority consensus with randomization. We simulate two standard and one randomized algorithm in different setups, including ordered and topologically randomized networks with noise and message loss. We study faulty nodes with persistent failure in comparison with faulty nodes with random failure.

New results presented in this chapter can be summarized as follows:

- faulty nodes with persistent failure inhibit binary majority consensus stronger than faulty nodes with random failure;
- randomization by noise, message loss, and topology can mitigate such a decrease with $M \leq 3$ in the system;
- although randomization can help mitigate the decrease in R, with $M > 3$ the decrease in convergence rate is drastic;
- Random Neighbor Majority consensus, proposed in Chapter 5, exhibits higher robustness towards faulty nodes in all studied setups.

The stronger impact of persistent faulty nodes makes them more adverse. A number of persistent faulty nodes (e.g., $M = 2$) with both possible faulty states $\sigma_M \in \{-1, 1\}$ will inhibit the binary majority consensus system with arbitrary initial density stronger than $M = 4$ non-persistent faulty nodes, randomly switching between two or three possible states.

We show that in certain conditions (e.g., with GKL in asynchronous ring lattices) randomly failing faulty nodes can even promote consensus, due to local boundary effects and cluster-dithering impact. We should note, however, that latter result cannot be considered as a solution to the binary majority consensus problem as the system exhibits significant measure of stochastic dynamics.

In Chapter 4 we show that randomizing disturbances can promote wait-free binary majority consensus and that an optimal combination can be selected depending on a system conditions. In Chapter 5 we propose a random neighbor selection scheme for a consensus algorithm that with additional randomization by noise and errors can reach 100% convergence rate.

New results presented in this chapter show that randomizing disturbances not only increase the convergence rate but also promote robustness of wait-free binary majority consensus. We present a setup where faulty nodes themselves promote consensus due to local boundary effects and counter-clustering influence.

Ben-Or [BO83] offered a randomized algorithm that can tolerate up to a $M = \frac{N}{/}2$ faulty nodes in a fully asynchronous system. Important remaining question is whether an algorithm with the similar robustness can be designed for a wait-free system with limited connectivity, such as a binary majority consensus system. This issue again can be approached with a more complex solution than the ones studied in this work.

CHAPTER 7

Conclusions

Binary consensus algorithms can be used in systems where centralized decision-making is difficult or impossible. In technical systems, such use was reported in distributed database management [BG84], mission planning [AH06], wireless sensor networking [MNC10] and cognitive networking [AMM11]. Such real-life networked and distributed systems, such as wireless sensor networks, seldom operate in a synchronous regime, or with the full connectivity. Algorithms with bounded time can be beneficial in such networks as they are not suited for long-running algorithms, due to scarce resources of networked nodes. Due to restrictions in connectivity, execution time, and lack of synchrony, it is difficult to provide a solution with a 100% convergence rate. In the last several decades new algorithms advanced the convergence rate from $\simeq$ 82% to $\simeq$ 90%. Some of the best results were achieved by randomized algorithms. Randomization is a technique that utilizes stochastic disturbances to promote consensus.

In this thesis we investigate simple binary majority consensus with stochastic disturbances and faulty nodes. We study simple algorithms with explicit disturbances in different systems, including:

- ring lattices, as an example of ordered networks;
- random Watts-Strogatz and Waxman networks;
- synchronous and asynchronous networks.

We investigate the influence of explicit randomizing disturbances such as:

- Additive noise and random errors;
- Stochastic message loss;
- Randomly and persistently failing faulty nodes.

We simulate binary majority consensus with a focus on its convergence rate and speed. We analyze the system evolution dynamics and propose a random neighbor selection scheme that can promote consensus. With additional randomization by noise or errors, the proposed Random Neighbor Majority (RNM) algorithm shows higher convergence rate and fault tolerance in both ordered and randomized environments.

7.1 Summary of Contributions

We present new results in Chapters 4, 5 and 6. Chapter 4 investigates standard algorithms with various types of randomization. Chapter 5 proposes a new randomized algorithm and compares it with standard algorithms. Finally, Chapter 6 studies fault tolerance of aforementioned algorithms in various setups.

Chapter 4 investigates Gacs-Kurdyumov-Levin and Simple Majority consensus in ordered and topologically randomized Watts-Strogatz and Waxman Networks. Presented results show that topology randomization, noise, and message loss can promote distributed binary majority consensus. The analysis of system evolution indicates that consensus is promoted due to different types of "de-clustering" influence provided by stochastic disturbances. Different types of randomization show cumulative effect when combined, and an optimal combination of disturbances can be selected depending on system conditions. Although randomization can strongly promote consensus, it does not critically influence its convergence time. These observations can infer that some of the randomizing disturbances can be embedded in simple consensus algorithms.

Further, in Chapter 5 we propose an algorithm with embedded random neighbor selection that can counter the network clustering without changing the underlying network topology. We study RNM with additional randomization by noise and errors in comparison with standard and randomized SM and GKL in ordered and topologically randomized Watts-Strogatz and Waxman networks. We explain that RNM outperforms SM and GKL due to embedded random neighbor selection . We show that an optimal level of additional randomization by noise or errors can be selected, depending on system conditions. We analyze system evolution dynamics and show that although RNM with additional randomization can reach R of up to $\simeq 100\%$, it cannot guarantee a stable agreement in a wait-free manner. Finally we suggest a further research direction to improve the stability of RNM through an adaptive levels of embedded randomization.

Chapter 6 investigates the impact of the faulty node behavior on binary majority consensus. It considers faulty nodes with random and persistent failure, including faulty nodes with clustered and random layout over the network. We show that GKL, SM and RNM strongly degrade with presence of faulty nodes, although RNM shows higher robustness with low number of faulty nodes. Next, we show that faulty nodes with persistent failure are more adverse for binary majority consensus, and that faulty node impact can be mitigated by randomization of various nature. We show that although in some setups GKL, randomized by clustered non-persistent faulty nodes, reaches almost 100% convergence rate, this cannot be considered as a wait-free solution for a binary majority consensus. Finally, we explain these effects with the analysis of system evolution.

Throughout the analysis we do not specifically accentuate whether the disturbances, such as noise, errors and message loss, are “embedded” in the algorithm or “external”. However, we keep the respective models simple and allow for their independent combinations so that if needed, a disturbance can be easily embedded in the algorithm. This can be done to increase the efficiency and robustness of the algorithm by complementing the disturbance profile of a given system without significant increase in complexity of the algorithm itself.

7.2 Open Questions and Future Work

New results presented in this thesis extend and complement the previous investigations on binary majority consensus with disturbances. We show that explicit randomization can promote simple algorithms for binary majority consensus, and increase their robustness towards faulty nodes. However, open challenges remain.

One of the most important ones is whether a stable consensus on arbitrary initial configurations with a limited network connectivity and bounded time is possible. In Chapters 4 and 5 we mention two possible approaches for the further research in this direction. The first approach assumes the fusion of the counter-clustering influences provided by different disturbances. The second intuitive direction for the further research is to embed an adaptive mechanism for disturbance levels. Such a mechanism could trigger a high level of the appropriate disturbance in the beginning of the system evolution (or if the node’s state is unchanged for a certain number of time steps). Then a disturbance level can be relaxed in time to ensure the stability of the new achieved state of the overall system. Both approaches will significantly increase the complexity of the algorithms and require additional extensive research that exceeds the scope of this thesis.

Another important issue is related to the robustness of the consensus towards faulty nodes: although we show that randomization can mitigate their impact, already at $M \geq 3$ of faulty nodes added to system convergence rate decreases below 50%. Therefore, it is important to investigate if a solution for a wait-free binary majority consensus can be proposed with the robustness close to that of Ben-Or’s algorithm.

These questions can be addressed analytically and by means of simulations. Interested scholars are advised to introduce themselves to works of Mitchell, Fukś, and Aspnes.

List of Symbols

N	Number of nodes in the system
M	Number of faulty nodes added to the system
i	Number of the node
K	Number of neighbors on each side of the lattice
P	Rewiring probability in Watts-Strogatz networks: a probability of the node i to have a link to a random node $z, z \notin \{i-K, \dots, i-1, i+1, \dots, i+K\}$
α	A parameter defining the average connectivity in Waxman networks
β	A parameter defining the average link length in Waxman networks
d	A distance between nodes i and j in Waxman networks, randomly chosen from $(0,1]$
S	A list of all neighbors of the node i
S_l	A list of left-side neighbors of the node i
S_r	A list of right-side neighbors of the node i
$\sigma_i[0]$	Initial state of the node i
$\sigma_i[t]$	State of the node i at the time t
$\sigma_{i,j}$	State of the node j received by the node i at the time t
$\sigma_{i,i}$	State of the node i received by the node itself, $\sigma_{i,i} \neq \sigma_i$ in presence of randomizing disturbances
σ_M	State of the faulty node: (a) with a persistent failure model $\sigma_M = -1$ if $\rho_0 > 0$, $\sigma_M = 1$ if $\rho_0 < 0$, (b) with a random failure: $\sigma_M \in \{-1, 1\}$, (c) With a random and full failure: $\sigma_M \in \{-1, 0, 1\}$
$G(x)$	Decision step function, $G = -1$ if $x < 0$, $G = 1$ if $x > 0$
$\mathcal{E}$	Message loss probability

A	Additive noise amplitude
η	Probability of receiving erroneous message
$\phi_{i,j}$	Sample of noise added to a transmitted state message $\sigma_{i,j}$
I	Initial system configuration: a combination of N states $\sigma_i \in \{-1, 1\}$
ρ	System density, the sum of all states in I; $\rho \in \{-N, \dots, N\}$
$\rho[t]$	System density at the time t
ρ_0	Initial system density at the time $t = 0$
R	Convergence rate of the algorithm: a fraction of initial configurations that result in a successful agreement
T	Algorithm termination time, $T = 2N$
t_c	The convergence (agreement) moment, if $\rho_0 < 0, \sigma_i[t_c] = -1, \forall i$, if $\rho_0 > 0, \sigma_i[t_c] = 1, \forall i$
F	Convergence speed of the algorithm, $F = 1 - \frac{t_c}{T}$ registered for successfully converged networks
n_j	A node randomly chosen from the list of available neighbors

List of Own Publications

[1] Alexander Gogolev and Christian Bettstetter. Robustness of self-organizing consensus algorithms: Initial results from a simulation-based study. In *Self-Organizing Systems*, volume 7166 of *LNCS*, pages 104–108. Springer, 2012.

[2] Alexander Gogolev and Lucio Marcenaro. Density classification in asynchronous random networks with faulty nodes. In *Proceedings of the 22nd International Conference on Parallel, Distributed and Network-Based Processing*, pages 256–261, 2014.

[3] Alexander Gogolev and Lucio Marcenaro. Randomized binary consensus with faulty nodes. *Entropy*, 16:2820–2838, 2014.

[4] Alexander Gogolev and Lucio Marcenaro. Efficient binary consensus in randomized and noisy networks. In *Proceedings of the 9th IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, 2014.

[5] Alexander Gogolev, Nikolaj Marchenko, Christian Bettstetter, and Lucio Marcenaro. Distributed binary consensus in networks with faults. Under review.

Bibliography

[ABK96] David Andre, Forrest H. Bennett III, and John R. Koza. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 3–11, 1996.

[AC07] Hagit Attiya and Keren Censor. Tight bounds for asynchronous randomized consensus. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 155–164, 2007.

[ACT98] Marcos Aguilera, Wei Chen, and Sam Toueg. Failure detection and consensus in the crash-recovery model. In *Distributed Computing*, volume 1499 of *LNCS*, pages 231–245. 1998.

[AFJ06] Dana Angluin, Michael J. Fischer, and Hong Jiang. Stabilizing consensus in mobile networks. In *Distributed Computing in Sensor Systems*, volume 4026 of *LNCS*, pages 37–50. 2006.

[AH06] Mehdi Alighanbari and Jonathan P. How. An unbiased Kalman consensus algorithm. In *Proceedings of American Control Conference*, pages 3519–3524, 2006.

[ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[AMM11] Seyed Ashrafi, Mehrzad Malmirchegini, and Yyasamin Mostofi. Binary consensus for cooperative spectrum sensing in cognitive radio networks. In *Proceedings of Global Telecommunications Conference*, pages 1–6, 2011.

[Asp00] James Aspnes. Fast deterministic consensus in a noisy environment. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 299–308, 2000.

[Asp02] James Aspnes. Fast deterministic consensus in a noisy environment. *Journal of Algorithms*, 45(1):16–39, 2002.

[Asp03] James Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003.

[AT12] Marcos K. Aguilera and Sam Toueg. The correctness proof of Ben-Ors randomized consensus algorithm. *Distributed Computing*, 25(5):371–381, 2012.

[BDG12] Andrea Baronchelli and Albert Diaz-Guilera. Consensus in networks of mobile communicating agents. *Physical Review E*, 85:016113, 2012.

[Bea05] Randall W. Beard. Consensus seeking in multi-agent systems under dynamically changing interaction topologies. *IEEE Transactions on Automatic Control*, 50(5):655–661, 2005.

[BG84] Philip A. Bernstein and Nathan Goodman. An algorithm for concurrency control and recovery in replicated distributed databases. *ACM Transactions on Database Systems*, 9(4):596–615, 1984.

[BHKPS06] Janna Burman, Ted Herman, Shay Kutten, and Boaz Patt-Shamir. Asynchronous and fully self-stabilizing time-adaptive majority consensus. In *Principles of Distributed Systems*, volume 3974 of *LNCS*, pages 146–160. 2006.

[BO83] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 27–30, 1983.

[BR92] Gabriel Bracha and Ophir Rachman. Randomized consensus in expected O(n2log n) operations. In *Distributed Algorithms*, volume 579 of *LNCS*, pages 143–150. 1992.

[BR99] Albert-Laszlo Barabási and Albert Reka. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[BT85] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840, 1985.

[BTV11] Florence Benezit, Patrick Thiran, and Martin Vetterli. The distributed multiple voting problem. *IEEE Journal of Selected Topics in Signal Processing*, 5(4):791–804, 2011.

[CFF$^+$07] Ruggero Carli, Fabio Fagnani, Paolo Frasca, Tom Taylor, and Ro Zampieri. Average consensus on networks with transmission noise or quantization. In *Proceedings of European Control Conference*, pages 4189–4194, 2007.

[Cha96] Tushar Deepak Chandra. Polylog randomized wait-free consensus. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 166–175, 1996.

[CHT96] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, 1996.

[CIL94] Benny Chor, Amos Israeli, and Ming Li. Wait-free consensus using asynchronous hardware. *SIAM Journal on Computing*, 23(4):701–712, 1994.

[CT96] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[CUBS02] Andrea Coccoli, Peter Urban, Andrea Bondavalli, and Andre Schiper. Performance analysis of a consensus algorithm combining stochastic activity networks and measurements. In *Proceedings of International Conference on Dependable Systems and Networks*, pages 551–560, 2002.

[DCMH95] Rajarshi Das, James P. Crutchfield, Melanie Mitchell, and James E. Hanson. Evolving globally synchronized cellular automata. In *Proceedings of the International Conference on Genetic Algorithms*, pages 336–343, 1995.

[DDS87] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, 1987.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[DMC94] Rajarshi Das, Melanie Mitchell, and James P. Crutchfield. A genetic algorithm discovers particle-based computation in cellular automata. In *Parallel Problem Solving from Nature*, volume 866 of *LNCS*, pages 344–353. 1994.

[DMS09] Giovanna Di Marzo Serugendo. Robustness and dependability of self-organizing systems — a safety engineering perspective. In *Stabilization, Safety, and Security of Distributed Systems*, volume 5873 of *LNCS*, pages 254–268. 2009.

[Fat13] Nazim Fates. Stochastic cellular automata solutions to the density classification problem. *Theory of Computing Systems*, 53(2):223–242, 2013.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[FD88] Charles E. Furman and Robert A. Duke. Effect of Majority Consensus on Preferences for Recorded Orchestral and Popular Music. *Journal of Research in Music Education*, 36(4):220–231, 1988.

[FGEM11] Juan Fernández-Gracia, Victor M. Eguíluz, and Maxi San Miguel. Update rules and interevent time distributions: slow ordering vs. no ordering in the voter model. *Physical Review E*, 84:3118, 2011.

[FH02] Serge Fenet and Salima Hassas. A distributed Intrusion Detection and Response System based on mobile autonomous agents using social insects communication paradigm. *Electronic Notes in Theoretical Computer Science*, 63:41–58, 2002.

[Fis83] Michael J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *Foundations of Computation Theory*, volume 158 of *LNCS*, pages 127–140. 1983.

[FLM85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Merritt. Easy impossibility proofs for distributed consensus problems. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 59–70, 1985.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[FS08] Antonio Fasano and Gesualdo Scutari. The effect of additive noise on consensus achievement in wireless sensor networks. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2277–2280, 2008.

[Fuk97] Henryk Fukś. Solution of the density classification problem with two cellular automata rules. *Physical Review Letters E*, 55:R2081–R2084, 1997.

[FZ08] Fabio Fagnani and Sandro Zampieri. Randomized consensus algorithms over large scale networks. *IEEE Journal on Selected Areas in Communications*, 26(4):634–649, 2008.

[Gac01] Peter Gacs. Reliable cellular automata with self-organization. *Journal of Statistical Physics*, 103:45–267, 2001.

[GC96] Andrew Gee and Roberto Cipolla. Fast visual tracking by temporal consensus. *Image and Vision Computing*, 14(2):105–114, 1996.

[GF08] Andrey Goder and Vladimir Filkov. Consensus clustering algorithms: comparison and refinement. In *Proceedings of Workshop on Algorithm Engineering and Experiments*, pages 109–117, 2008.

[GKL78] Peter Gacs, Georgy L. Kurdyumov, and Leonid A. Levin. One-dimensional uniform arrays that wash out finite islands. *Problemy Peredachi Informacii*, 14:92–98, 1978.

[GKPS11] Alexander Gogolev, Anatoliy Khakhaev, Alexander Pergament, and Alexey Shtykov. Effects of harmonic modulation of current in glow discharge dusty plasma with ordered structures. *Contributions to Plasma Physics*, 51(6):498–504, 2011.

[GWR$^+$06] Thomas Grotkjaer, Ole Winther, Birgitte Regenberg, Jens Nielsen, and Lars Kai Hansen. Robust multi-scale clustering of large DNA microarray datasets with the consensus algorithm. *Bioinformatics*, 22(1):58–67, 2006.

[GY89] Ronald L. Graham and Andrew C. Yao. On the improbability of reaching Byzantine agreements. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 467–478, 1989.

[HM07a] Minyi Huang and Jonathan H. Manton. Stochastic double array analysis and convergence of consensus algorithms with noisy measurements. In *Proceedings of American Control Conference*, pages 705–710, 2007.

[HM07b] Minyi Huang and Jonathan H. Manton. Stochastic Lyapunov analysis for consensus algorithms with noisy measurements. *Proceedings of American Control Conference*, pages 1419–1424, 2007.

[HMB08] Richard Holzer, Hermann Meer, and Christian Bettstetter. On Autonomy and Emergence in Self-Organizing Systems. In *Self-Organizing Systems*, volume 5343 of *LNCS*, pages 157–169. 2008.

[HMSKC06] Salima Hassas, Giovanna Di Marzo-Serugendo, Anthony Karageorgos, and Cristiano Castelfranchi. On Self-Organising Mechanisms from Social, Business and Economic Domains. *Informatica*, 30:63–71, 2006.

[JP98] Hugues Juille and Jordan B. Pollack. Coevolving the "ideal" trainer: Application to the discovery of cellular automata rules. In *Proceedings of the Annual Conference on Genetic Programming*, pages 519–527, 1998.

[KB06] Derek B. Kingston and Randall W. Beard. Discrete-time average consensus under switching network topologies. In *Proceedings of American Control Conference*, pages 3551–3556, 2006.

[KCC05] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. Manet simulation studies: The incredibles. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(4):50–61, 2005.

[KFN92] Janusz Kacprzyk, Mario Fedrizzi, and Hannu Nurmi. Group decision making and consensus under fuzzy preferences and fuzzy majority. *Fuzzy Sets and Systems*, 49(1):21–31, 1992.

[KKBT12] Johannes Klinglmayr, Christoph Kirst, Christian Bettstetter, and Marc Timme. Guaranteeing global synchronization in networks with stochastic interactions. *New Journal of Physics*, 14(7):073031, 2012.

[KM07] Soummya Kar and Jose M. F. Moura. Distributed average consensus in sensor networks with random link failures and communication channel noise. In *Proceedings of Asilomar Conference on Signals, Systems and Computers*, pages 676–680, 2007.

[Kum91] Akhil Kumar. Hierarchical quorum consensus: a new algorithm for managing replicated data. *IEEE Transactions on Computers*, 40(9):996–1004, 1991.

[Lap95] Jean-Claude Laprie. Dependable computing and fault-tolerance: concepts and terminology. In *Proceedings of International Symposium on Fault-Tolerant Computing*, pages 2–11, 1995.

[LB95] Mark Land and Richard K. Belew. No perfect two-state cellular automata for density classification exists. *Physical Review Letters E*, 74:5148–5150, 1995.

[LDCH10] Zhongkui Li, Zhisheng Duan, Guanrong Chen, and Lin Huang. Consensus of multiagent systems and synchronization of complex networks: A unified viewpoint. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(1):213–224, 2010.

[LEWV10] Jixin Li, Ehsan Elhamifar, I-Jeng Wang, and Rene Vidal. Consensus with robustness to outliers via distributed optimization. In *Proceedings of IEEE Conference on Decision and Control*, pages 2111–2117, 2010.

[LM05] David W. Leverington and Wooil M. Moon. On the use of consensus algorithms to address variability in the results of neural network classifications: preliminary tests involving two northern study areas. *Canadian Journal of Remote Sensing*, 31(4):269–273, 2005.

[LS97] Louisa Lam and Ching Y. Suen. Application of majority voting to pattern recognition: an analysis of its behavior and performance. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 27(5):553–568, 1997.

[MCH94] Melanie Mitchell, James P. Crutchfield, and Peter T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D: Nonlinear Phenomena*, 75(1-3):361–391, 1994.

[Men11] J. Ricardo G. Mendonçia. Sensitivity to noise and ergodicity of an assembly line of cellular automata that classifies density. *Physical Review Letters E*, 83(3):31112–31117, 2011.

[MHC93] Melanie Mitchell, Peter T. Hraber, and James P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130, 1993.

[MM81] Timothy Margush and Fred R. McMorris. Consensus n-trees. *Bulletin of Mathematical Biology*, 43(2):239–244, 1981.

[MM09] Yasamin Mostofi and Richard M. Murray. Kalman filtering over wireless fading channels — How to handle packet drop. *International Journal of Robust and Nonlinear Control*, 19(18):1993–2015, 2009.

[MMDA04] André A. Moreira, Abhishek Mathur, Daniel Diermeier, and Luís A. N. Amaral. Efficient system-wide coordination in noisy environments. *Proceedings of the National Academy of Sciences of the United States of America*, 101(33):12085–12090, 2004.

[MN04] Chip Martel and Van Nguyen. Analyzing Kleinberg's (and other) small-world models. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 179–188, 2004.

[MNC10] Henrique Moniz, Nuno Ferreira Neves, and Miguel Correia. Turquois: Byzantine consensus in wireless ad hoc networks. In *Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 537–546, 2010.

[MP91] Fred J. Meyer and Dhiraj K. Pradhan. Consensus with dual failure modes. *IEEE Transactions on Parallel and Distributed Systems*, 2(2):214–222, 1991.

[MS90] Renato Mirollo and Steven Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, 1990.

[NBD$^+$09] David M. Nathan, John B. Buse, Mayer B. Davidson, Ele Ferrannini, Rury R. Holman, Robert Sherwin, and Bernard Zinman. Medical management of hyperglycaemia in type 2 diabetes mellitus: a consensus algorithm for the initiation and adjustment of therapy. *Diabetologia*, 52(1):17–30, 2009.

[NM05] Van Nguyen and Chip Martel. Analyzing and characterizing small-world graphs. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 311–320, 2005.

[OSS05] Reza Olfati-Saber and Jeff S. Shamma. Consensus filters for sensor networks and distributed sensor fusion. In *Proceedings of IEEE Conference on Decision and Control and European Control Conference*, pages 6698–6703, 2005.

[PJJJS02] Krzysztof Pawlikowski, Hae-Duck Joshua Jeong, and Ruth Lee Jong-Suk. On credibility of simulation studies of telecommunication networks. *IEEE Communications Magazine*, 40:132–139, 2002.

[PSL80] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[RBA05] Wei Ren, Randall W. Beard, and Ella M. Atkins. A survey of consensus problems in multi-agent coordination. In *Proceedings of American Control Conference*, volume 3, pages 1859–1864, June 2005.

[Ren07] Wei Ren. Multi-vehicle consensus with a time-varying reference state. *Systems and Control Letters*, 56(78):474–483, 2007.

[RGH77] Lee Ross, David Greene, and Pamela House. The "false consensus effect": An egocentric bias in social perception and attribution processes. *Journal of Experimental Social Psychology*, 13(3):279–301, 1977.

[RM08] Yongxiang Ruan and Yasamin Mostofi. Binary consensus with soft information processing in cooperative networks. In *Proceedings of IEEE Conference on Decision and Control*, pages 3613–3619, 2008.

[RW11] Ram Rajagopal and Martin J. Wainwright. Network-based consensus averaging with general noisy channels. *IEEE Transactions on Signal Processing*, 59(1):373–385, 2011.

[SKB10] Andrea Simonetto, Tamas Keviczky, and Robert Babuska. Distributed nonlinear estimation for robot localization using weighted consensus. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3026–3031, 2010.

[SSW91] Michael Saks, Nir Shavit, and Heather Woll. Optimal time randomized consensus-making resilient algorithms fast in practice. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, pages 351–362, 1991.

[SZ93] Michael Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 101–110, 1993.

[TF83] Dean Tjosvold and Richard H. G. Field. Effects of social context on consensus and majority vote decision making. *Academy of Management Journal*, 26(3):500–506, 1983.

[Tho79] Robert H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems*, 4(2):180–209, 1979.

[TInY$^+$13] János Török, Gerardo Iñiguez, Taha Yasseri, Maxi San Miguel, Kimmo Kaski, and János Kertész. Opinions, conflicts, and consensus: modelling social dynamics in a collaborative environment. *Physical Review Letters*, 110(8):88701–88708, 2013.

[TK12a] Kevin Topley and Vikram Krishnamurthy. Average-consensus in a deterministic framework — part i: strong connectivity. *IEEE Transactions on Signal Processing*, 60(12):6590–6603, 2012.

[TK12b] Kevin Topley and Vikram Krishnamurthy. Average-consensus in a deterministic framework — part ii: central connectivity. *IEEE Transactions on Signal Processing*, 60(12):6604–6616, 2012.

[TN09] Behrouz Touri and Angelia Nedic. Distributed consensus over network with noisy links. In *Proceedings of International Conference on Information Fusion*, pages 146–154, 2009.

[US04] Péter Urbán and André Schiper. Comparing distributed consensus algorithms. In *Proceedings of International Conference on Applied Simulation and Modelling*, pages 474–480, 2004.

[Van01] Piet Van Mieghem. Paths in the simple random graph and the Waxman graph. *Probability in the Engineering and Informational Sciences*, 15(4):535–555, 2001.

[VD02] Igor G. Vladimirov and Phil Diamond. A uniform white-noise model for fixed-point roundoff errors in digital systems. *Automation and Remote Control*, 63(5):753–765, 2002.

[Wat99] Duncan J. Watts. *Small worlds: the dynamics of networks between order and randomness.* 1999.

[Wax88] Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.

[WE10] Jing Wang and Nicola Elia. Dynamic average consensus over random networks with additive noise. In *Proceedings of IEEE Conference on Decision and Control*, pages 4789–4794, 2010.

[Wil96] Mark Wilkinson. Majority-rule reduced consensus trees and their use in bootstrapping. *Molecular biology and evolution*, 13(3):437–44, 1996.

[Wol02] Stephen Wolfram. *A new kind of science.* 2002.

[WS98] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of “small-world” networks. *Nature*, 393(6684):440–442, 1998.

[Yan01] Xin-She Yang. Small-world networks in geophysics. *Geophysical Research Letters*, 28(13):2549–2552, 2001.

[YWW07] Zhiwen Yu, Hau-San Wong, and Hongqiang Wang. Graph-based consensus clustering for class discovery from gene expression data. *Bioinformatics*, 23(21):2888–2896, 2007.