UNIVERSITÀ
DEGLI STUDI
FIRENZE

DOTTORATO DI RICERCA IN INFORMATICA, SISTEMI E TELECOMUNICAZIONI

Indirizzo: INGEGNERIA INFORMATICA MULTIMEDIALITA' E TELECOMUNICAZIONI

CICLO XXVII

COORDINATORE Prof. Luigi Chisci

Wireless Sensor Networks (WSNs) for Critical Applications: analysis and enhancements

Settore Scientifico Disciplinare ING-INF/05

Dottorando
De Guglielmo Domenico

Tutori
Prof. Anastasi Giuseppe

Prof. Vicario Enrico

Ing. Francavilla Gianpiero

Coordinatore
Prof. Luigi Chisci

Anni 2012/2014

Abstract - In this thesis we study the suitability of *Wireless Sensor Networks* (WSNs) for suppporting *critical* applications, i.e. applications where, in addition to *energy efficiency*, other requirements such as *reliability*, *timeliness*, *security* and *scalability* must be taken into consideration. We organize the thesis into three main parts. First, we consider WSNs compliant to the IEEE 802.15.4 standard that is currently the most popular technology for commercial WSNs. We show that 802.15.4 WSNs suffer from severe limitations in terms of reliability and scalability that make them unsuitable for critical applications. Since these limitations are mainly due to an improper setting of the CSMA/CA algorithm used to regulate channel access in 802.15.4 networks, we propose JIT-LEAP, an adaptive and learning-based algorithm that adaptively tunes the 802.15.4 CSMA/CA parameters so as to provide the level of reliability required by the application with the minimum energy consumption. JIT-LEAP makes 802.15.4 networks suitable for applications having reliability as the main concern. However, it is not a viable solution for time-critical applications since the increase in reliability comes at the cost of increased packet latency. When low and predictable latencies are required, a *Time Division Multiple Access* (TDMA) scheme is typically used for channel access. Hence, the second part of this thesis focuses on TDMA-based WSNs. Despite TDMA provides guaranteed bandwidth, high energy efficiency, low and predictable latency, it also has a number of limitations (e.g. it requires a strict synchronization between sensor nodes, an allocation of slots to sensor nodes and, suffers from selective jamming attack). Thus, we provide original solutions to overcome some of them. We first propose LOCALL, a localized algorithm for allocation of transmission slots to sensor nodes. Thanks to its localized approach, LOCALL does not require the exchange of messages to establish a communication schedule and, hence, minimizes energy consumption of sensor nodes. Then, we propose JAMMY, a *distributed* solution to contrast the selective jamming attack in TDMA-based WSNs. The solutions we propose in the first two parts of the thesis significantly improve the performance/robustness of both 802.15.4 and TDMA-based networks. However, the fact that these technologies rely on one single channel for communication can significanlty degrade their performance in real-world scenarios. WSNs share the wireless spectrum with other ambient technologies such as WiFi, Bluetooth and, hence, suffer from external interference. Moreover, the performance of WSNs is usually affected by multi-path fading since any object in their surroundings acts as a reflector for RF signals. Using multiple channels for communication and a channel hopping scheme, has been shown to be an effective way to mitigate both external interference and multi-path fading. For this reason, IEEE has recently proposed the IEEE 802.15.4e *Time Slotted Channel Hopping* (TSCH) mode, a new channel access mechanism that combines *time slotted* access, with *channel hopping* capabilities. Given these characteristics TSCH is one of the most promising technologies to support real-world WSN critical applications. Therefore, the last part of the thesis is devoted to analyze it.

Publications

This thesis is based on the following publications.

Journals

- M. Tiloca, D. De Guglielmo, G. Dini, G. Anastasi, S.K. Das, **JAMMY: a Distributed and Self-Adaptive Solution against Selective Jamming Attack in TDMA WSNs**, Submitted to IEEE Transactions on Dependable and Secure Computing (TDSC). Major Revision decision received on 08/11/2014.

- D. De Guglielmo, F. Restuccia, G. Anastasi, M. Conti, S.K. Das, **Accurate and Efficient Modeling of 802.15.4 Unslotted CSMA/CA through Event Chains Computation**, Submitted to IEEE Transactions on Mobile Computing (TMC).

- S. Brienza, M. Roveri, D. De Guglielmo, G. Anastasi, **Just-in-Time Adaptive Algorithm for Optimal Parameters Setting in 802.15.4 WSNs**, Submitted to ACM Transactions on Autonomous and Adaptive Systems (TAAS).

- D. De Guglielmo, Beshr al Nahas, S. Duquennoy, T. Voigt, G. Anastasi, **Analysis and Experimental Evaluation of IEEE 802.15.4e TSCH CSMA-CA Algorithm**, In preparation.

Conferences

- D. De Guglielmo, A. Seghetti, G. Anastasi, M. Conti, **A Performance Analysis of the Network Formation Process in IEEE 802.15.4e TSCH Wireless Sensor/Actuator Networks**, Proceedings of IEEE International Symposium on Computers and Communications (ISCC 2014), Madeira, Portugal, June 23-26, 2014.

- S. Brienza , D. De Guglielmo, C. Alippi, G. Anastasi, M. Roveri, **A Learning-based Algorithm for Optimal MAC Parameters Setting in IEEE 802.15.4 Wireless Sensor Networks**, Proceedings of ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN 2013), Barcelona, Spain, November 3-7, 2013.

- G. Anastasi, M. Antonelli, A. Bechini, S. Brienza, E. D'Andrea, D. De Guglielmo, P. Ducange, B. Lazzerini, F. Marcelloni, A. Segatori, **Urban and Social Sensing for Sustainable Mobility in Smart Cities**, Proceedings of IFIP/IEEE International Conference on Sustainable Internet and ICT for Sustainability (SustainIT 2013), Palermo, Italy, October 29-31, 2013.

- M. Tiloca, D. De Guglielmo, G. Dini, G. Anastasi, **SAD-SJ: a Self-Adaptive Decentralized solution against Selective Jamming attack in Wireless Sensor Networks**, Proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2013), Cagliari, Italy, September 10-13, 2013.

- D. De Guglielmo, G. Anastasi, M. Conti, **A Localized Slot Allocation Algorithm for Wireless Sensor Networks**, Proceedings of IEEE/IFIP Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net 2013), Ajaccio, France, June 24-26, 2013.

- S. Brienza, D. De Guglielmo, G. Anastasi, M. Conti, V. Neri, **Strategies for Optimal MAC Parameter Setting in IEEE 802.15.4 Wireless Sensor Networks: a Performance Comparison**, Proceedings of IEEE International Symposium on Computers and Communications (ISCC 2013), Split, Croatia, July 7-10, 2013.

- D. De Guglielmo, G. Anastasi, **Wireless Sensor and Actuator Networks for Energy Efficiency in Buildings**, Proceedings of IFIP/IEEE International Conference on Sustainable Internet and ICT for Sustainability (SustainIT 2012), Pisa, Italy, October 4-5, 2012.

- D. De Guglielmo, G. Anastasi, M. Conti, **A Localized Desynchronization Algorithm for Periodic Data Reporting in IEEE 802.15.4 WSNs**, Proceedings of IEEE International Symposium on Computers and Communications (ISCC 2012), Cappadocia, Turkey, July 1-4, 2012.

- D. De Guglielmo, G. Anastasi, M. Conti, **Performance Analysis of a Localized Desynchronization Algorithm for Periodic Data Collection in Wireless Sensor Networks** , 3rd InfQ Workshop, Lucca, Italy, July 2012. Paper selected for journal extension.

Book chapters

- D. De Guglielmo, G. Anastasi, A Seghetti, **From IEEE 802.15.4 to IEEE 802.15.4e: a Step towards the Internet of Things**, Chapter 10 in Advances onto the Internet of Things (S. Gaglio, G. Lo Re, Editors), Series on Advances in Intelligent Systems and Computing, N. 260, January 2014. Springer.

Contents

Chapter 1

Introduction

A *Wireless Sensor Network* (WSN) is composed of a (large) number of sensor devices deployed over a certain geographical area and interconnected through wireless links. Sensor devices gather information from the physical environment or a monitored system (e.g. temperature, pressure, vibrations), optionally perform a preliminary local processing of acquired information, and send (raw or processed) data to a central controller. Based on the received information, the controller takes intelligent decisions and performs appropriate actions, through actuator devices, to change the behavior of the physical environment or the monitored system.

Nowadays, WSNs are used in many applications domains ranging from environmental monitoring and location/tracking applications to industrial [1] and healthcare applications [2]. In the industrial field WSNs are used in factory automation [3], distributed and process control [4–6], real-time monitoring of machinery health, detection of liquid/-gas leakage, radiation check [7] an so on. In the healthcare domain WSNs have been considered for the monitoring of physiological data in chronicle patients and transparent interaction with the healthcare system.

Energy efficiency is usually the main concern in the design of WSNs-based systems. This is because sensor devices are typically powered by batteries, with a limited energy budget, and their replacement can be very expensive or, even, impossible [8]. However, in many relevant applications, hereafter referred to as *critical* applications, additional requirements such as *reliability*, *timeliness*, *scalability* and *security* must be considered as well [1, 9]. Reliability and timeliness are critical issues in industrial and healthcare

applications. In fact, in these kinds of applications, if data packets are not correctly delivered to the final destination within a pre-defined deadline, the correct behavior of the system (e.g. the timely detection of a critical event) may be compromised. The maximum allowed latency depends on the specific application and ranges from tens of milliseconds (e.g for discrete manufacturing and factory automation), to seconds (e.g. for process control) and even minutes (e.g. for asset monitoring) [9]. Security merits special attention in WSNs-based systems since sensor devices are often deployed in open, unattended, possibly hostile environments that makes them extremely prone to both cyber and physical attacks. Finally, scalability of algorithms and protocols for WSNs is also very important since WSNs may be composed of a large number of sensor nodes (e.g. for the monitoring of large geographical areas).

In this thesis we study the suitability of WSNs for supporting critical applications. The thesis is organized into three main parts. First, we consider WSNs compliant to the IEEE 802.15.4 standard [10] that is currently the most popular technology for commercial WSNs. We show that 802.15.4 WSNs suffer from severe limitations in terms of reliability and scalability that make them unsuitable for critical applications. Also, we highlight that these limitations are mainly due to an improper setting of the CSMA/CA algorithm used to regulate channel access in 802.15.4 networks. Therefore, we propose JIT-LEAP, an adaptive and learning-based algorithm that adaptively tunes the 802.15.4 CSMA/CA parameters so as to provide the level of reliability required by the application with the minimum energy consumption for sensor nodes. JIT-LEAP makes 802.15.4 networks suitable for applications having reliability as the main concern. However, it is not a viable solution for time-critical applications since, the increase in reliability comes at the cost of increased packet latency. In scenarios where low and predictable latencies are required, a *Time Division Multiple Access* (TDMA) scheme is typically used for regulating channel access. Hence, in the second part of the thesis we focus on TDMA-based WSNs. We highlight that, despite TDMA provides guaranteed bandwidth, high energy efficiency, absence of collisions (i.e. high reliability), low and predictable latency, it also has a number of limitations (e.g. it requires a strict synchronization between nodes and an allocation of communications slots). Hence, we provide original solutions to overcome some of them. Specifically, we first propose LOCALL, a localized algorithm for allocation of transmission slots to sensor nodes that, thanks to its localized approach, does not require the exchange of messages to establish a communication schedule. This minimizes

energy consumption of sensor nodes and makes LOCALL particularly suitable both for environments where packets can be corrupted or missed and dynamic networks. Then, we propose JAMMY, a distributed and self-adaptive solution to contrast the selective jamming attack [11, 12] in TDMA-based WSNs. Selective jamming is a particularly insidious form of Denial-of-Service (DoS) that allows an adversary to completely thwart the communication of a victim node with a very low probability to be detected.

Although the proposed solutions significantly improve the performance/robustness of both 802.15.4 and TDMA-based networks, the fact that these networks rely on one single channel for communication can significanlty degrade their performance in real-world applications. WSNs typically share their radio medium with other ambient technologies such as WiFi [13], Bluetooth [14] or even cordless phones and microwave ovens [15] and, hence, suffer from external interference. In addition, when WSNs operate in indoor environments, their performance is affected by multi-path fading since any wall, person, object in their surroundings acts as a reflector for RF signals [16]. Using multiple channels for communication, together with a channel hopping scheme, has been shown to be an effective way to mitigate both external interference and multi-path fading [16, 17]. For this reason, channel hopping is adopted by many industrial wireless technologies such as ISA [18] and WirelessHART [19]. In this perspective, IEEE has recently proposed the IEEE 802.15.4e *Time Slotted Channel Hopping* (TSCH) mode [20], a new channel access mechanism that combines *time slotted* access, with *multi-channel* and *channel hopping* capabilities. Given these characteristics TSCH is one of the most promising technologies to support real-world WSNs critical applications. For this reason, the last part of the thesis is devoted to analyze it. In the following, we describe the contributions of each single part of this thesis in detail and we outline its structure.

1.1 Contributions of this thesis

Part I. Analysis of 802.15.4 WSNs

In the first part of this thesis, we analyze the suitability of IEEE 802.15.4 WSNs for supporting critical applications. Products compliant to the IEEE 802.15.4 standard are largely available on the market and it is considered the reference technology for commercial WSNs applications. The 802.15.4 standard defines the *physical* (PHY) and *Medium*

Access Control Layer (MAC) of the protocol stack. In order to cope with different application requirements the IEEE 802.15.4 provides two different MAC operation modes namely the *Beacon-Enabled* (BE) mode and the *Non-Beacon-Enabled* (NBE) mode. The BE mode relies on a periodic superframe bounded by *beacons*, which are special synchronization frames generated by the coordinator nodes. In BE mode each sensor node waits for the reception of a beacon and, then, starts transmitting its data frames using a slotted carrier sense multiple access with collision avoidance (CSMA/CA) algorithm. Conversely, in the NBE mode there is no superframe; nodes are not synchronized, and an unslotted CSMA/CA algorithm is used for frame transmissions. The 802.15.4 standard and the CSMA/CA algorithms used in BE and NBE mode are described in chapter 2 of this thesis.

The performance of 802.15.4 MAC has been thoroughly investigated in the literature through analysis [21, 22], simulations [23–27] and real experiments [27, 28]. However, the majority of studies focused on the BE mode of operation only, while significant less attention has been devoted to analyze the performance of the NBE mode. Specifically, literature still lacks an analytical model of the unslotted 802.15.4 CSMA/CA algorithm (used in NBE mode) that is both accurate and tractable. Since the NBE mode is the most suitable access method for applications generating sporadic and/or irregular traffic (e.g event-driven WSNs and upcoming IoT applications) having an accurate and tractable analytical model of the unslotted CSMA/CA algorithm is imperative to investigate the quality of service that can be provided to this kind of applications. For this reason, in chapter 3 of this thesis we present a new analytical model of 802.15.4 unslotted CSMA/CA algorithm that, unlike previous proposed analytical models, does not introduce over-simplifying assumptions, is able to investigate the performance of WSNs composed of a large number of sensor nodes, and allows to compute a number of different performance metrics such as delivery ratio, average packet latency and energy consumption of sensor nodes.

The results presented in chapter 3 highlight that the unslotted 802.15.4 CSMA/CA algorithm suffers from serious limitations in terms of reliability and scalability that make it unsuitable for critical applications. Specifically, the delivery ratio provided by the protocol is very low even when the number of contending nodes is not so high. In addition, it sharply drops when the number of contending nodes increases. A similar behavior has been observed also for the slotted 802.15.4 CSMA/CA algorithm used in

BE mode as reported in [27]. Our results, and those reported in [27], demonstrate that the reliability provided by the 802.15.4 CSMA/CA algorithm (both slotted and unslotted) can be significantly increased by using higher CSMA/CA parameter values. However, this comes at the cost of a higher latency and/or energy consumption.

Regarding the BE mode, a number of approaches [22, 29, 30] have been proposed in the literature to provide the *optimal* setting of CSMA/CA parameter values, i.e. the set of values for CSMA/CA parameters through which it is possible to satisfy the application requirements (in terms of reliability) with the minimum energy consumption for sensor nodes. In real WSNs the identication of such optimal setting is not a trivial task, as reliability strongly depends on time-varying factors - such as number of sensor nodes, offered load and packet error rate (PER) - that can neither be controlled nor predicted. In chapter 4 of this thesis we compare (through simulation) the most relevant approaches presented in the literature for the tuning of 802.15.4 CSMA/CA algorithm parameters values. Then, in chapter 5, we present the *Just-in-Time LEarning-based Adaptive Parameter tuning* (JIT-LEAP) algorithm, an adaptive algorithm for the tuning of 802.15.4 CSMA/CA parameters that, leveraging a learning-based approach and using a theoretically-grounded *Change Detection Test* (CDT), outperforms all the previous solutions presented in the literature for the tuning of 802.15.4 CSMA/CA parameters. Our results demonstrate that through the use of JIT-LEAP is possible to significantly increase the reliability of 802.15.4 networks, making them suitable for applications having reliability as the main concern. However, JIT-LEAP is not a viable solution for time-critical applications since, in addition to reliability, it also increases packet latency. When applications have stringent requirements in terms of both reliability and timeliness, TDMA is typically used for regulating channel access. Therefore, the second part of this thesis is dedicated to analyze TDMA-based WSNs.

Part II. Analysis of TDMA-based WSNs

Many application domains require very high reliability and, at the same time, low and predictable latencies as well as energy efficiency. In such scenarios a *Time Division Multiple Access* (TDMA) scheme is typically used for regulating channel access. As well known, in TDMA-based systems time is divided into a sequence of periodic *superframes*, each one of which consists of a fixed number of transmission *slots*. Slots are allocated

to sensor nodes so that each node needs to be active only during its own slot(s), while it can sleep for the rest of the time. Therefore, TDMA provides guaranteed bandwidth, high energy efficiency, absence of collisions (i.e., high reliability), low and predictable latency. On the other side, TDMA has also a number of limitations. First of all, a strict synchronization among sensor nodes is needed [31, 32]. Second, TDMA has a limited *flexibility* since an allocation of slots to sensor nodes has to be performed in order for the network to operate [33] and a different slot allocation pattern (and hence a re-execution of the slot allocation procedure) is usually required if a change in the network topology occurs. Finally, TDMA suffers from *selective jamming* attack [12, 34], a particularly insidious form of Denial-of-Service (DoS) that allows an adversary to completely thwart the communication of a victim node with a very low probability to be detected. In this thesis we present two original proposals to both provide a flexible solution to the problem of slot allocation and to contrast selective jamming attacks in TDMA-based WSNs. Specifically, in chapter 7 we propose LOCALL, a localized slot allocation algorithm which allows sensor nodes of a TDMA-based WSN to select their own transmission slot(s) in a completely autonomous way, just relying on local information. Thanks to its localized approach LOCALL does not require the exchange of messages to establish a communication schedule. This minimizes energy consumption of sensor nodes and makes LOCALL particularly suitable both for environments where packets can be corrupted or missed and dynamic networks. In chapter 8, we propose JAMMY, a novel distributed and self-adaptive solution against selective jamming attacks in TDMA-based WSNs. Unlike previous approaches, JAMMY is completely *distributed*, i.e. it does not suffer from typical limitations of centralized solutions (e.g. single point of failure). Moreover, JAMMY introduces a negligible computation overhead and no communication overhead at all.

Although the solutions proposed in part I and II of this thesis significantly improve the performance/robustness of both 802.15.4 and TDMA-based networks, the fact that these networks rely on one single channel for communication can significanlty degrade their performance in real-world applications. WSNs typically share their radio medium with other ambient technologies such as WiFi [13], Bluetooth [14] or even cordless phones and microwave ovens [15] and, hence, suffer from external interference. In addition, the performance of indoor WSNs is affected by multi-path fading since any wall, person, object in their surroundings acts as a reflector for RF signals [16]. Using multiple

channels for communication, together with a channel hopping scheme, has been shown to be an effective way to mitigate both external interference and multi-path fading [16, 17]. For this reason, many industrial wireless technologies such as ISA [18] and WirelessHART [19] adopt channel hopping. In this perspective, IEEE has recently proposed the IEEE 802.15.4e *Time Slotted Channel Hopping* (TSCH) mode [20], a new channel access mechanism that combines *time slotted* access, with *multi-channel* and *channel hopping* capabilities. Given these characteristics TSCH is expected to be one of the major enabling technologies for future real-world WSNs applications. For this reason, the last part of the thesis is devoted to analyze it.

Part III. Analysis of IEEE 802.15.4e TSCH

The IEEE 802.15.4e standard [20] enhances and adds functionalities to the 802.15.4 MAC layer, in order to address the emerging needs of embedded industrial applications. IEEE 802.15.4e defines a number of MAC (Medium Access Control) *behavior modes*, to support specific application domains, and some general *functional improvements* not tied to any specific application domain. A general description of the new mechanisms introduced by the standard is reported in chapter 9. In this thesis we focus on the *Time Slotted Channel Hopping* (TSCH) behavior mode of IEEE 802.15.4e, which combines *time slotted* access, with *multi-channel* and *channel hopping* capabilities, thus providing increased network capacity, high reliability and predictable latency, while maintaining very low duty cycles (i.e., energy efficiency). These unique characteristics make TSCH one of the most promising technologies for future real-world WSNs applications. In chapter 9, 10 and 11 of this thesis we focus on analyzing TSCH networks. The majority of studies on TSCH focus on link scheduling [35][36][37], i.e. on the assignment of communication links to sensor nodes and, hence, consider a fully operative network. Conversely, in this thesis we investigate the mechanisms offered by TSCH to bootstrap/build the network. Specifically, in chapter 10 we analyze the network formation process of IEEE 802.15.4e TSCH network. We define a simple *random-based network advertisement* algorithm and analyze its performance, through both analysis and simulations, in terms of *joining time*, i.e. the total time taken by a new device to join the TSCH network. Then, in chapter 11, we focus on TSCH shared links, i.e special communication slots assigned to more than one transmitter. Shared links are expected to play a key role in future TSCH networks since they will be used - in combination with, or as an alternative to, dedicated links

(i.e. slots assigned to one single transmitter) - during the network formation process (e.g. to exchange routing/scheduling information) and also in case of network failure (e.g. when a free-of-collision communication schedule is not available). We analyze the CSMA/CA algorithm used by TSCH nodes to concurrently access shared slots. Specifically, we develop an analytical model of TSCH CSMA/CA, based on Discrete Time Markov Chains (DTMC) and use it to predict the performance experienced by nodes when accessing shared links. Since capture effect - i.e. the ability of some radios to correctly receive a strong signal from one transmitter, despite significant interference from other transmitters - has a significant impact on the performance of real wireless networks, we also consider this aspect in our model [38]. Our model has been validated through both simulation experiments and measurements in a real testbed. The obtained results clearly show the limitations of the TSCH CSMA/CA algorithm and the impact of different parameters on its performance. Also, it is shown that capture effect significantly improves the performance of the algorithm.

We conclude the thesis in chapter 12 by drawing conclusion and outlining possible directions of future work.

1.2 Outline of the thesis

This thesis is organized as follows. In chapters 2, 3, 4 and 5 we analyze the suitability of IEEE 802.15.4 WSNs for supporting critical applications. In chapter 2 we introduce the IEEE 802.15.4 standard, review literature on IEEE 802.15.4 and motivate the work we present in chapters 3, 4 and 5. In chapter 3 we present a new analytical model of the IEEE 802.15.4 unslotted CSMA/CA algorithm that, differently from previous analytical models, is both accurate and tractable. Through the model it is possible to compute a number of different performance metrics such as delivery ratio, average packet latency and energy consumption of sensor nodes and to investigate the impact of different parameters on the network performance. In chapter 4 we compare the performance of the different strategies that have been proposed in the literature for the tuning of 802.15.4 CSMA/CA algorithm parameters. The analysis highlights the pros and the cons of the various approaches but, also, reveals some limitations of current solutions. For this reason in chapter 5 we present JIT-LEAP, a new learning-based and adaptive algorithm for the tuning of 802.15.4 CSMA/CA parameters that overcomes

the limitations of previous solutions and outperforms their performance. JIT-LEAP makes 802.15.4 networks suitable for applications having reliability as the main concern. However, it is not a viable solution for time-critical applications.

In scenarios where low and predictable latencies are required TDMA is typically used for regulating channel access. Hence, the second part of this thesis (chapters 6, 7 and 8) focuses on TDMA-based WSNs. In chapter 6 we highlight the limitations of TDMA-based WSNs. Then, in chapters 7 and 8 we present two solutions to overcome some of them. Specifically, in chapter 7 we present LOCALL, a localized slot allocation algorithm which allows sensor nodes of a TDMA-based WSN to select their own transmission slot(s) in a completely autonomous way, just relying on local information. This minimizes energy consumption of sensor nodes and makes LOCALL particularly suitable both for environments where packets can be corrupted or missed and dynamic networks. Then, in chapter 8 we propose JAMMY, a novel distributed and self-adaptive solution against selective jamming attacks in TDMA-based WSNs. Unlike previous approaches, JAMMY is completely *distributed*, i.e. it does not suffer from typical limitations of centralized solutions (e.g. single point of failure). Moreover, JAMMY introduces a negligible computation overhead and no communication overhead at all.

The solutions we propose in part I and II of this thesis significantly improve the performance of both 802.15.4 WSNs and TDMA-based WSNs. However, the fact that these networks rely on one single channel for communication significantly limits their performance in real environments. The last part of this thesis (chapters 9, 10, 11) analyzes the IEEE 802.15.4e TSCH MAC behavior mode, a new multi-channel MAC protocol specifically designed to support critical industrial applications. In chapter 9 we introduce the IEEE 802.15.4e standard with a special emphasis on the *Time Slotted Channel Hopping* (TSCH) MAC behavior mode. Then, in chapter 10 and 11 we present our contributions. Specifically, in chapter 10 we study the nework formation process of TSCH networks. We define a simple *random-based network advertisement* algorithm and analyze its performance, through both analysis and simulations. Then, in chapter 11 we propose an analytical model of the TSCH CSMA/CA algorithm, i.e. the algorithm used by nodes of TSCH networks to access shared links.

Chapter 12 concludes this thesis with a summary of our research work, and points out some future research directions.

Part I
IEEE 802.15.4 WSNs

Chapter 2

IEEE 802.15.4 WSNs: background and literature review

2.1 Introduction

The first part of this thesis is devoted to analyze the suitability of 802.15.4 networks for supporting critical applications. In this chapter we first describe the IEEE 802.15.4 standard and the *Carrier Sense Multiple Access* with *Collision Avoidance* (CSMA/CA) algorithms it uses for regulating channel access. Then, we review the most relevant works analyzing the performance of 802.15.4 networks and motivate the work we present in chapters 3, 4 and 5.

2.2 IEEE 802.15.4 standard

The IEEE 802.15.4 [10] is a standard for low-rate, low-power, and low-cost Personal Area Networks (PANs). It defines the *physical* (PHY) and *Medium Access Control* (MAC) layer of the protocol stack. Products compliant to the IEEE 802.15.4 standard are largely available on the market and 802.15.4 is considered the reference technology for commercial WSNs.

A PAN is formed by one PAN coordinator which is in charge of managing the whole network, and, optionally, by one or more coordinators that are responsible for a subset

of nodes in the network. Regular nodes must associate with a (PAN) coordinator in order to communicate. The supported network topologies are *star* (single-hop), *cluster-tree* and *mesh* (multi-hop) (see Figure 2.1). The standard defines two different channel access methods: a *beacon enabled* (BE) mode and a *non-beacon enabled* (NBE) mode (see Figure 2.2). The beacon enabled mode provides a power management mechanism based on a duty cycle. It uses a superframe structure (see Figure 2.3) which is bounded by *beacons*, i.e., special synchronization frames generated periodically by the coordinator node(s).

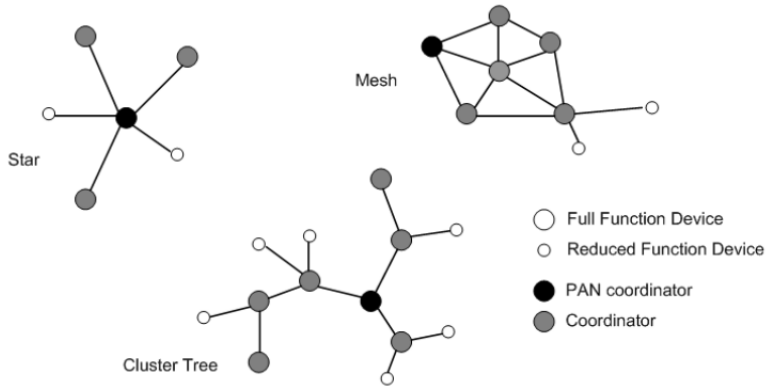


FIGURE 2.1: IEEE 802.15.4 network topologies

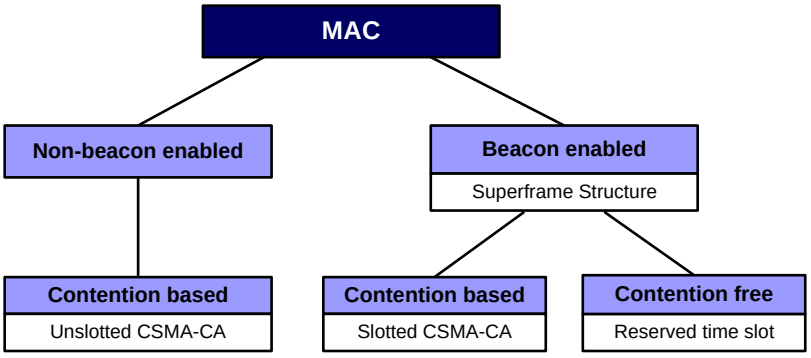


FIGURE 2.2: IEEE 802.15.4 MAC operation modes

The time between two consecutive beacons is called *Beacon Interval* (BI), and is defined through the *Beacon Order* (BO) parameter ($BI = 15.36 \cdot 2^{BO}$ ms, with $0 \leq BO \leq 14$). Each superframe consists of an active period and an inactive period. In the active period nodes communicate with the coordinator they are associated with, while during the inactive period they enter a low power state to save energy. The active period is denoted as *Superframe Duration* (SD) and its size is defined by the *Superframe Order* (SO) parameter ($SD = 15.36 \cdot 2^{SO}$ ms, with $0 \leq SO \leq BO \leq 14$). It can be further divided into a *Contention Access Period* (CAP) and a *Contention Free Period* (CFP).

During CAP a slotted CSMA/CA algorithm is used for channel access, while in the CFP communication occurs in a TDMA (Time Division Multiple Access) style by using a number of *Guaranteed Time Slots* (GTSs), pre-assigned to individual nodes. In the non-beacon enabled mode there is no superframe, nodes are always active (energy conservation is delegated to the layers above the MAC protocol) and use an unslotted CSMA/CA algorithm for channel access.

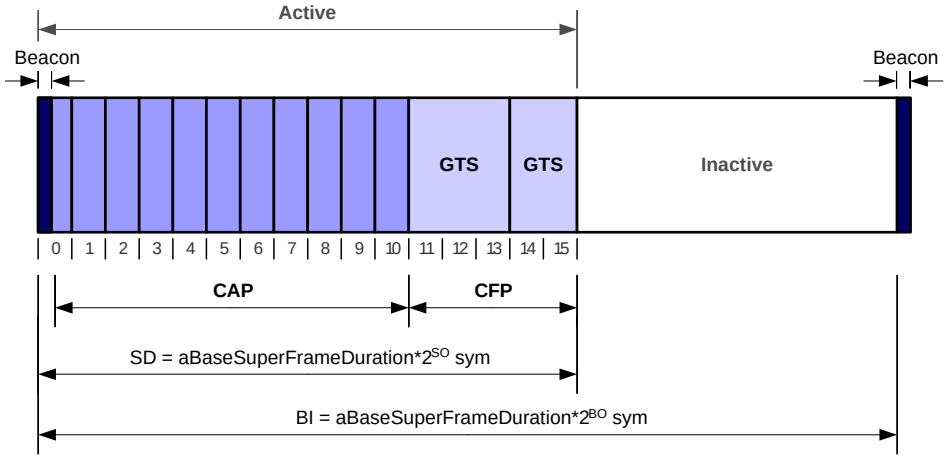


FIGURE 2.3: IEEE 802.15.4 Superframe

2.3 IEEE 802.15.4 CSMA/CA algorithm

The CSMA/CA algorithm is used in both the *beacon enabled* mode (during the CAP portion of the active period) and the *non-beacon enabled* mode. In the beacon enabled mode a slotted scheme is used - i.e., all operations are aligned to backoff period slots (whose duration is $320\mu s$) - while in the non-beacon enabled mode there is no such alignment. Upon receiving a data frame to be transmitted, the CSMA/CA algorithm performs the following steps.

1. A set of state variables is initialized, i.e., the contention window size ($CW = 2$, only for the slotted variant), the number of backoff stages carried out for the on-going transmission ($NB = 0$), and the backoff exponent ($BE = macMinBE$).

2. A random backoff time, uniformly distributed in the range $[0,\ 2^{BE} - 1] \cdot 320\mu s$, is generated and used to initialize a backoff timer. In the beacon-enabled mode, the starting time of the backoff timer is aligned with the beginning of the next backoff slot. In addition, if the backoff time is larger than the residual CAP duration, the

backoff timer is stopped at the end of the CAP and resumed at the beginning of the next superframe. When the backoff timer expires, the algorithm proceeds to step 3.

3. A Clear Channel Assessment (CCA) is performed to check the state of the wireless medium.

   a If the medium is busy, the state variables are updated as follows: $NB = NB+1$, $BE = min(BE+1,\ macMaxBE)$ and $CW = 2$ (only for the slotted variant). If the number of backoff stages has exceeded the maximum admissible value (i.e. $NB > macMaxCSMABackoffs$), the frame is dropped. Otherwise, the algorithm falls back to step 2.

   b If the medium is free and the access mode is unslotted, the frame is immediately transmitted.

   c If the medium is free and the access mode is slotted, then $CW = CW - 1$. If $CW = 0$ then the frame is transmitted [1]. Otherwise the algorithm falls back to step 3 to perform a second CCA.

The complete CSMA/CA algorithm, both in the slotted and unslotted version, is depicted in Figure 2.4. The 802.15.4 CSMA/CA algorithm also includes an optional retransmission mechanisms for improving reliability. When retransmissions are enabled, the destination node must send an acknowledgement whenever it correctly receives a data frame (the acknowledgement is not sent in case of collision and corrupted frame reception). On the sender side, if the acknowledgment is not (correctly) received within the pre-defined timeout, a retransmission is scheduled. The frame can be re-transmitted up to a maximum number of times, specified by the MAC parameter $macMaxFrameRetries$. Upon exceeding this value, the data frame is rejected and a failure notification is sent by the MAC sublayer to the upper layers.

2.4 Literature review

In this section we review the most relevant works analyzing the performance of 802.15.4 networks. Specifically, in section 2.4.1 we review works analyzing the unslotted 802.15.4

[1] In the beacon-enabled mode, before starting the frame transmission, the algorithm calculates whether it is able to complete the operation within the current CAP. If there is not enough time, the transmission is deferred to the next superframe.

CSMA/CA algorithm used in *non-beacon enabled* (NBE) WSNs. We highlight that an accurate and tractable analytical model of the unslotted 802.15.4 CSMA/CA algorithm is still missing in the literature. Therefore, in chapter 3 of this thesis we present a new analytical model that does not introduce simplifying assumptions in the modeling of the CSMA/CA algorithm and is able to study WSNs composed of a large number of sensor nodes. In section 2.4.2 we focus on the 802.15.4 *beacon-enabled* (BE) mode of operation. First, we survey studies regarding the analysis of the 802.15.4 BE WSNs. Then, we describe a number of approaches proposed in the literature to improve the

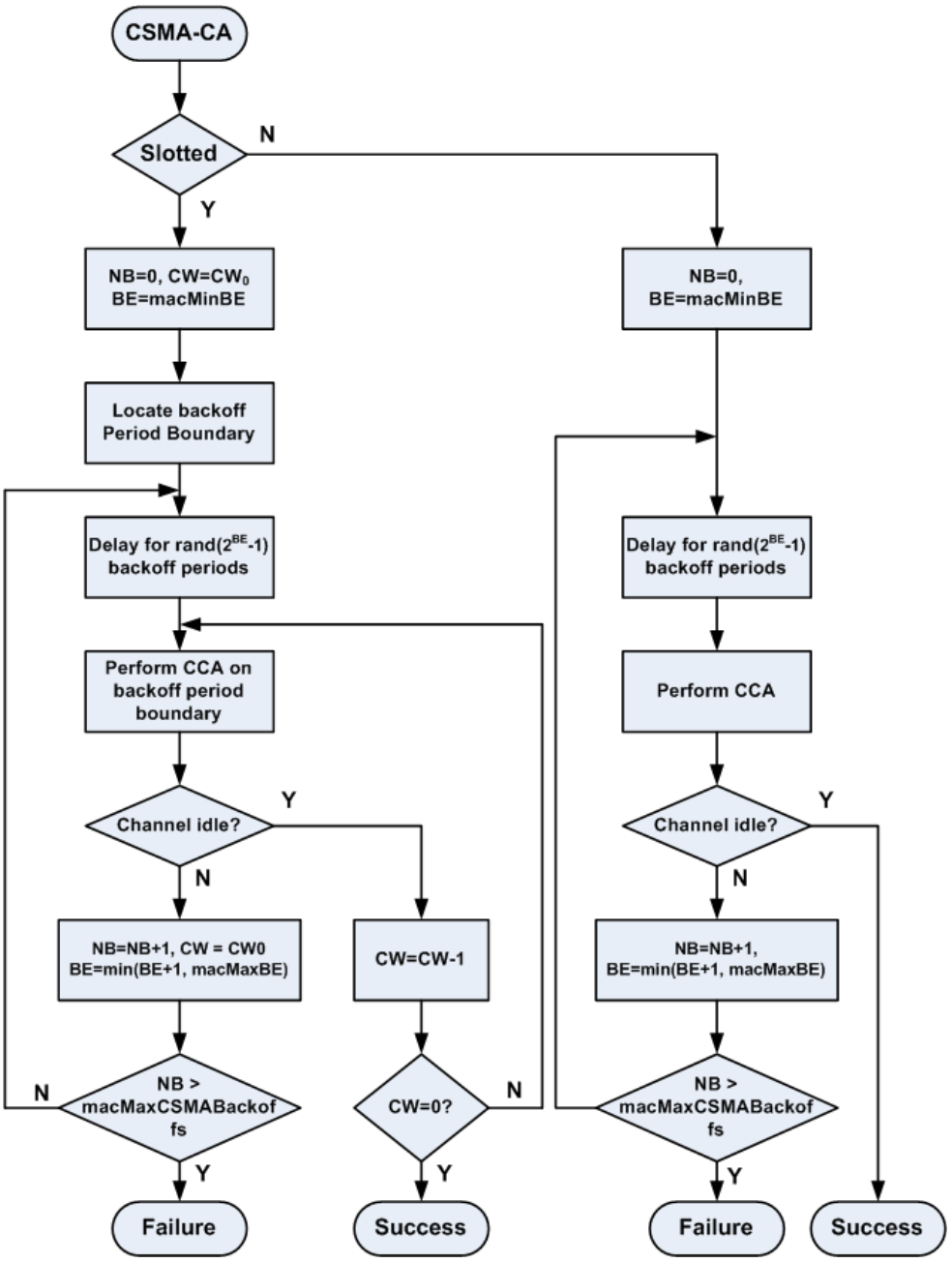


FIGURE 2.4: IEEE 802.15.4 CSMA/CA

performance of 802.15.4 BE WSNs. A performance comparison of these approaches is performed in chapter 4. Then, in chapter 5 we propose a solution outperforming all the previous proposed approaches.

2.4.1 Literature on 802.15.4 NBE networks

Although the 802.15.4 NBE mode of operation is a suitable access method for applications generating sporadic and/or irregular traffic such as event-driven WSN applications and upcoming IoT applications, little attention has been devoted to analyze its performance. In the following, we describe the most relevant studies analyzing the unslotted 802.15.4 CSMA/CA algorithm (used in NBE mode) and highlight their limitations.

One of the earliest studies of the unslotted CSMA/CA was carried out by Latré *et al.* in [39]. However, this analysis is based on simplifying assumptions, e.g. one transmitter and one receiver only, which make the study not completely realistic.

In [40], a star network operating in non-saturated conditions is considered, where the authors developed a model based on a discrete-time Markov chain (DTMC) to derive performance metrics of interest. They used this model to find the optimal number of nodes satisfying some pre-defined QoS requirements. However, as in [39], the analysis is not completely realistic. Specifically, the authors consider two consecutive clear channel assessment (CCA) operations, instead of a single one (as stated in the 802.15.4 standard). Goyal *et al.* [41] proposed a stochastic model of the unslotted CSMA/CA algorithm assuming that packet inter-arrival times follow an exponential distribution, and considering the effect of packet retransmissions. Finally the authors in [42] analyzed the unslotted CSMA/CA algorithm in single and multi-hop scenarios, by proposing an accurate model based on a DTMC, and investigated the joint impact of the routing and MAC protocols.

In [40–42] the authors assume that sensor nodes generate data packets following a given probability distribution. However, this assumption does not apply to the majority of practical WSN scenarios, as sensor nodes typically follow either a *periodic* or *event-driven* reporting paradigm.

Regarding single-hop, event-driven WSNs, the most relevant works are [43] and [44]. In [43] the authors derive the packet latency and delivery ratio experienced by N sensor nodes attempting to transmit a packet simultaneously to the sink node using the unslotted 802.15.4 CSMA/CA algorithm. However, they do not consider the effects of acknowledgements and retransmissions in their analysis. In [44], Gribaudo *et al.* provide a very accurate and complete analytical model of the unslotted CSMA/CA algorithm using stochastic automata networks (SANs) [45]. The analysis is mainly aimed at deriving the packet delay distribution and on-time delivery ratio (percentage of packets received by the sink node within a pre-defined threshold). As in [43], the analysis of the energy consumption of sensor nodes is neglected. Furthermore, although the use of SANs makes the analysis very accurate, it also raises serious complexity issues. In fact, the analysis in [44] is limited to WSNs with a low number of nodes (i.e. n less than 6). This is because the size of the WSN global descriptor (i.e., a matrix) increases exponentially with the network size. Hence, the computation time, as well as the memory needed to solve the model, increases accordingly.

From the previous discussion it emerges that existing studies of the unslotted CSMA/CA algorithm do not provide accurate analysis of large-scale WSNs, due mainly to tractability issues and/or simplifying assumptions. For this reason in chapter 3 we propose a new analytical model of the unslotted 802.15.4 CSMA/CA algorithm. Likewise the models proposed in [43, 44] our model considers event-driven applications. Also, it is very accurate since it considers acknowledgements and retransmissions, and no over-simplifying assumptions are made on the CSMA/CA algorithm. In terms of performance metrics, through the model is possible to derive delivery ratio, packet latency, *and* energy consumption of sensor nodes. However, the most important contribution is that the model is able to analyze WSNs with a large number of nodes. This is because the model is not matrix-based (like DTMCs or SaNs). Instead it undertakes an approach called *Event Chains Computation* (ECC), that makes the analysis very accurate yet computationally tractable. As we will show in chapter 3, ECC is scalable and, unlike previous techniques, is particularly suitable for parallelization, due to its intrinsic concurrent structure. This contributes to drastic reduction of computation time. Table 2.1 compares past work with ours.

TABLE 2.1: Comparison of Related Works and ours.

Papers	Traffic	ACKs	RTXs	Delay	Energy	Scalable
Latré[39]	Single node	✓	×	×	×	✓
Kim[40]	Bernoulli	✓	✓	✓	✓	✓
Goyal[41]	Exponential	✓	✓	✓	×	✓
DiMarco[42]	Poisson	✓	✓	✓	✓	✓
Buratti[43]	Event-Driven	×	×	✓	×	✓
Gribaudo[44]	Event-Driven	✓	✓	✓	×	×
Our model	Event-Driven	✓	✓	✓	✓	✓

2.4.2 Literature on 802.15.4 BE networks

The performance of 802.15.4 networks operating in beacon enabled (BE) mode has been extensively investigated in the past. First studies on 802.15.4 used an analytical approach to investigate its performance. However many of them fail in properly characterizing the protocol due to simplifying assumptions on the CSMA/CA algorithm and/or on the packet generation process. For instance, the model proposed in [46] does not match simulation results since the authors used the same assumptions of Bianchi's 802.11 model [47] despite the 802.15.4 protocol significantly differs from 802.11. In [48] the authors assume that the probability to find the channel free during the first and second CCA are independent. This approximation causes significant differences between simulative and analytical results. Finally, the models presented in [21, 49] assume that packets are generated according to an exponential distribution. This way they do not consider the case of simultaneous transmissions by part of many sensor nodes as for example occurs after an inactive period. This aspect has been emphasized in [50] where the authors propose to introduce a random delay before channel access to avoid potential congestion after an inactive period.

Problems of reliability and scalability of 802.15.4 have been observed in [26, 51–53]. In [51] the authors consider a star network topology and assume that all nodes in the network try to transmit a packet at the beginning of the active period. They show that the packet drop probability is very high in this case. However, their analysis does not consider the effects of using acknowledgements and retransmissions. Also, no solutions to the problem are proposed.

In [52] the authors analyze the performance of an 802.15.4 network in terms of throughput and energy consumption. They show that 802.15.4 has poor performance when the number of contending nodes is high. In addition, they propose an enhanced MAC

protocol which offers better scalability. However, they do not consider the impact of CSMA/CA parameters on protocol performance.

Both in [26] and [53] a star network topology and saturated traffic conditions are considered. Both studies report that a large fraction of packets is dropped during the channel access and that the drop probability increases with the number of contending nodes. Also, they both show that increasing the backoff windows can alleviate the problem.

A comprehensive analysis of 802.15.4 MAC based on both simulations and real experiments is reported in [27]. The authors highlight that the IEEE 802.15.4 operating in BE mode suffers from severe limitations in terms of reliability and scalability that are mainly due to its CSMA/CA algorithm. They show that the performance of 802.15.4 degrades sharply when the number of sensor nodes increases. However, they point out that in 802.15.4 this degradation is very strong due to the default CSMA/CA parameter values suggested by the standard. The authors demonstrate that these default values are not appropriate, even for a network composed of a low number of sensor nodes. Also, they show that the packet delivery probability (and, hence, reliability) can be significantly increased by using higher CSMA/CA parameter values. However, this comes at the cost of a higher latency and/or energy consumption. Hence, an appropriate parameters setting should be found, depending on the application requirements.

Ideally, the CSMA/CA parameter setting should be chosen in such a way to guarantee the reliability level required by the application with the minimum energy consumption for sensor nodes. However, in real WSNs the identication of such optimal setting is not a trivial task, as reliability strongly depends on time-varying factors - such as number of sensor nodes, offered load and packet error rate (PER) - that can neither be controlled nor predicted. Several solutions have been proposed to identify the optimal CSMA/CA setting in 802.15.4 WSNs. They can be broadly classified as *model-based offline computation* [22], *model-based adaptation* [29] and *measurements-based adaptation* [30]. A detailed description of these strategies is reported in the following. Then, in chapter 4 we compare their performance in order to better understand pros and cons of the various approaches.

2.4.2.1 802.15.4 CSMA/CA Parameters Setting

From the description of the CSMA/CA algorithm reported in section 2.3 it emerges that the 802.15.4 CSMA/CA behavior is regulated by four parameters that are summarized in Table 2.2, along with the corresponding values allowed by the standard. The problem to address is: how to select the *optimal* parameter setting, i.e., the set of values that can provide the reliability level required by the application with the minimum energy consumption for the sensor nodes (under time-varying operating conditions). To solve this problem three solutions have been proposed [22, 29, 30] namely *model-based offline computation* [22], *model-based adaptation* [29], and *measurement-based adaptation* [30].

TABLE 2.2: CSMA/CA Parameters and allowed values

Parameter	**Values**	**Description**
macMaxFrameRetries	Range: 0-7 Default: 3	Maximum number of retransmissions
macMaxCSMABackoffs	Range: 0-5 Default: 4	Maximum number of backoff stages -1
macMaxBE	Range: 3-8 Default: 5	Maximum backoff window exponent
macMinBE	Range: 0-7 Default: 3	Minimum backoff window exponent

Model-based Offline Algorithm

The solution proposed in [22] takes an offline computation approach and leverages on an analytical model of the IEEE 802.15.4 CSMA/CA algorithm, based on a Discrete Time Markov Chain (DTMC). For the details of the analytical model please refer to [22]. The model considers a sensor network with a star topology and assumes that the operating conditions (number of sensor nodes, packet size) are known. In addition, the wireless medium is assumed to be ideal. Under such hypothesis, the analytical model is able to provide, for any CSMA/CA parameter setting, the following performance indexes: (**i**) packet delivery ratio, (**ii**) average packet latency, and (**iii**) average energy consumed by a sensor node for correctly delivering a packet to the sink. Then, the optimal parameter setting can be easily identified as the setting capable of providing the delivery ratio required by the application with the minimum energy consumption.

In practice, the effectiveness of the algorithm is strongly influenced by the accuracy of the analytic model it leverages on. In other words, the parameter setting computed by

the algorithm may not be optimal if the underlying model is not accurate. Furthermore, we need to point out that the analytical model developed in [22] assumes that the communication channel is ideal (i.e., transmission errors never occurs). In addition, it assumes the following packet generation process:

1. At any sensor node, after a frame transmission (and at startup time) a new packet is generated with a probability equal to (1-q_0), while, after a frame transmission, the transmission queue remains empty with probability q_0;
2. if there are no packets to transmit, the node goes to the *idle* state. While in this state, a new packet is generated with a probability equal to (1-q_0).

From assumptions (i) and (ii) it follows that q_0=0 corresponds to a saturated traffic condition, i.e., all sensor nodes always have a packet ready for transmission.

Model-based Adaptive Algorithm

The model-based algorithm described in the previous section works offline and, in addition, it requires to know the operating conditions of the sensor network. Hence, this solution is not suitable for real environments where operating conditions typically vary over time and are, often, difficult to predict in advance. Finally, the algorithm requires a significant amount of time to solve the model and to derive the optimal parameter setting. To overcome these limitations in [29] a model-based adaptive algorithm is proposed by the same authors. The latter algorithm is still based on a DTMC model of the sensor network. However, the model is a simplified version of the one developed in [22] and, thus, it can be solved by sensor nodes with limited computational and memory resources. In particular, sensor nodes estimate some congestion indexes through online measurements, instead of deriving them analytically from the model. Operationally, at the beginning of each Beacon Interval, a generic sensor node performs the following steps:

1. Estimates the probability to find the channel busy during the first and second CCA (referred to as α and β, respectively), as well as the probability τ that the node will attempt the first CCA in a generic backoff slot; to this end, it uses local measurements collected in previous Beacon Intervals.

2. Introduces α, β, and τ in the analytical model and derives delivery ratio, average packet latency and energy consumption, for any CSMA/CA parameter set;

3. Selects the optimal parameter setting.

To improve the accuracy of the estimates derived at step (i), each index is estimated on the basis of m samples (collected in previous Beacon Intervals), using the moving average method. Since the previous actions are repeated at each Beacon Interval, the algorithm is able to react to possible changes in the operating conditions.

Measurement-based Adaptive Algorithm

The main limitation of a model-based approach is that (**i**) its effectiveness strongly depends on the accuracy of the related model, and (**ii**) it typically takes a long time to solve the model and provide the optimal solution. An alternative strategy is using a measurement-based approach, as in the *ADaptive Access Parameters Tuning* (ADAPT) algorithm presented in [30]. ADAPT is a heuristic algorithm that dynamically adapts the CSMA/CA parameter setting, based on local measurements of the performance index that must be guaranteed. The algorithm presented in [30] focuses on reliability and tries to guarantee a packet delivery probability d^{des}, specified by the application, with minimum energy consumption.

In [29] it is shown that the packet delivery probability increases monotonically with the minimum contention window size (*macMinBE*), the number of backoff trials (*macMaxCSMABackoffs*), and the number of retransmissions (*macMaxFrameRetries*). However, increasing *macMaxCSMABackoffs* results in a higher energy consumption than increasing *macMinBE*, while increasing *macMaxFrameRetries* results in a higher energy consumption than *macMaxCSMABackoffs*. On the basis of these results, ADAPT dynamically increases and decreases the value of the above-mentioned parameters, depending on the measured delivery probability, which also depends on the time-varying operating conditions. Specifically, on the basis of d^{des}, ADAPT identifies the following two thresholds that define the reliability region within which the delivery ratio should be confined:

$$d^{low} = d^{des} \cdot (1+\sigma)$$

	ADAPT Algorithm
1	**If** $d_i^{est} < d^{low}$ **then**
2	**If** $macMinBE < macMinBE^{max}$ **then**
3	$macMinBE = \min(macMinBE + 2, macMinBE^{max})$
4	**Else If** $macMaxCSMABackoffs < macMaxCSMABackoffs^{max}$ **then**
5	$macMaxCSMABackoffs = \min(macMaxCSMABackoffs + 2$, $macMaxCSMABackoffs^{max})$
6	**Else If** $d_i^{est} > d^{high}$ **then**
7	**If** $macMaxCSMABackoffs > macMaxCSMABackoffs^{min}$ **then**
8	$macMaxCSMABackoffs = macMaxCSMABackoffs - 1$
9	**Else If** $macMinBE > macMinBE^{min}$ **then**
10	$macMinBE = macMinBE - 1$

$$d^{high} = d^{des} \cdot (1+\sigma+\gamma)\,, \sigma, \gamma \in [0, 1]$$

Then, at each Beacon Interval i, the algorithm estimates the current delivery probability d_i^{est} experienced by the sensor node as follows

$$d_i^{est} = \delta d_{i-1}^{est} + (1-\delta) \cdot d_i^{meas} \tag{2.1}$$

where d_i^{meas} is the delivery probability measured during the i-th Beacon Interval (ratio between number of acknowledged packets and number of transmitted packets), and δ is a memory factor in the range [0, 1]. To guarantee the reliability constraint d^{des}, ADAPT compares the estimated delivery probability d_i^{est} against the two thresholds d^{low} and d^{high}, and applies the tuning strategy reported by **ADAPT Algorithm** to compensate possible variations in the delivery probability, due to congestion. ADAPT also includes an additional module to contrast the effect of packets dropped due to transmission errors. Basically, each sensor node also measures the packet loss probability $l_{\mathrm{i}}^{\mathrm{est}}$ due to transmission errors, using an approach similar to eq. 2.1. If $(1 - l_{\mathrm{i}}^{\mathrm{est}}) < d^{des}$ it is not possible to guarantee the required delivery ratio even using the maximum values for $macMinBE$ and $macMaxCSMABackoffs$. Hence, the retransmission mechanism must be enabled.

Chapter 3

Modeling the unslotted 802.15.4 CSMA/CA Algorithm

The 802.15.4 *non-beacon enabled* (NBE) mode of operation is a suitable access method for applications generating sporadic and/or irregular traffic such as event-driven WSN applications and upcoming IoT applications. In order to investigate the quality of service that can be provided to this kind of applications, in this chapter we develop an accurate and tractable analytical model of the unslotted 802.15.4 CSMA/CA algorithm through which it is possible to derive performance metrics of interest such as delivery ratio, packet latency and energy consumption of sensor nodes. In order to deal with the significant complexity of the algorithm, we use an approach called *Event Chains Computation* (ECC). ECC relies on the idea that *outcomes* of the CSMA/CA algorithm can be represented as a sequence (chain) of transmissions (events) that subsequently occur in the network. ECC is able to iteratively build all the possible sequences of events that can be experienced by sensor nodes while transmitting a data packet. However, to reduce complexity, only the event chains whose probability to occur is above a predefined threshold are considered. By an appropriate selection of the threshold, it is possible to reduce the model complexity, which in turn reduces the computational resources needed to calculate the performance metrics, with a limited loss of accuracy. In addition, the computation of each single event chain is done independently from the others. Hence, it is possible to parallelize the algorithm, which further reduces the computation time.

To summarize, this chapter makes the following contributions. First, we introduce the Event Chains Computation (ECC) approach that leverages the analysis of the most likely events, thus reducing drastically the computation time. Next, we use ECC to derive performance metrics of interest such as delay, energy consumption, and packet reception probability. Finally, we validate our model through simulation and analyze the performance of the unslotted 802.5.4 CSMA/CA algorithm as a function of different operating parameters. Our results demonstrate that the ECC approach allows to reduce drastically the computation time, while guaranteeing a good accuracy for the computed performance metrics.

The chapter is organized as follows. Next section recalls the details of the unslotted CSMA/CA algorithm. Section 3.2 presents the model assumptions. Section 3.3 and 3.4 detail the ECC algorithm and derive performance metrics. Section 3.5 presents the obtained results. Finally, Section 3.6 draws conclusions.

3.1 CSMA/CA Algorithm

According to the IEEE 802.15.4 standard, in WSNs operating in the NBE mode, sensor nodes must associate with a special coordinator node and send their packets to it, using the unslotted CSMA/CA algorithm. Unlike regular sensor nodes, coordinator nodes are energy-unconstrained devices that form a higher-level network aimed at forwarding data to the final destination. Specifically, coordinator nodes are always on and, thus, sensor nodes are allowed to start a packet transmission at any time. In addition, no synchronization is required. The unslotted CSMA/CA algorithm has been described in section 2.3. However, for reader convenience, below we recall its most important steps.

Upon receiving a data packet, the MAC layer at the sensor node performs the following steps.

1. A set of state variables is initialized, namely the number of backoff stages carried out for the on-going transmission (NB = 0) and the backoff exponent (BE = *macMinBE*).

2. A random backoff time is generated and used to initialize a timer. The backoff time is obtained by multiplying an integer number uniformly distributed in $[0,\ 2^{BE-1}]$

by the the duration of the *backoff-period* (D_{bp}). As soon as the timer expires the algorithm moves to step 3.

3. A Clear Channel Assessment (CCA) is performed to check the state of the wireless medium.

   (a) If the medium is free, the packet is immediately transmitted.

   (b) If the medium is busy, state variables are updated as follows: NB = NB+1 and BE = min(BE+1,*macMaxBE*). If the number of backoff stages has exceeded the maximum allowed value (i.e. NB > *macMaxCSMABackoffs*), the packet is dropped. Otherwise, the algorithm falls back to step 2.

The CSMA/CA algorithm supports an optional retransmission scheme based on acknowledgements and timeouts. When the retransmission mechanism is enabled, the destination node must send an acknowledgement upon receiving a correct data packet. On the sender side, if the acknowledgment is not (correctly) received within a pre-defined timeout, the packet is retransmitted, unless the maximum number of allowed retransmissions (*macMaxFrameRetries*) has been reached. Otherwise, the packet is dropped.

3.2 Model Assumptions

In the following, we focus on the communication between sensor nodes and the coordinator node they are associated with. We assume that there are N nodes associated with the considered coordinator node. We refer to event-driven applications in which *all* nodes start transmitting a data packet simultaneously to their coordinator node, to report a certain detected physical event. This is the most challenging scenario in terms of performance and energy consumption. We assume that each sensor node is in the carrier sensing range of each other and there are no obstacles in the sensing field. This assures that the hidden node problem never arises. In addition, we assume that the time between two consecutive physical events ph_1 and ph_2 is long enough to assure that the execution of the CSMA/CA algorithm, started by sensor nodes to report ph_1, is surely terminated before ph_2 occurs. Hence, in the following, we focus on a single physical event ph. We indicate as $t = 0$ the time at which ph occurs and, hence, the time at which all the N nodes start executing the CSMA/CA algorithm. Finally, we make the following assumptions:

- Each sensor node transmits a single packet to report the detected event ph.
- Data packets transmitted by different sensor nodes have the same size. In particular, we assume that the packet size is such that the corresponding transmission time can assume a value D_{tx} such that $D_{tx} = D_{max} - k \cdot D_{bp}$, for $k \geq 0$, where D_{max} is the time required to transmit a maximum-size packet (133 bytes) and D_{bp} is the duration of the backoff period.
- The communication channel is ideal, i.e., data/acknowledgement packets are never corrupted, or lost, due to transmission errors.

3.3 Event Chains Computation

The CSMA/CA algorithm is a random access algorithm whose goal is to minimize the probability of collision between packets transmitted by different sensor nodes. Due to its random nature, different executions of the algorithm can yield completely different outcomes. For instance, if we consider N nodes simultaneously transmitting a single data packet to their coordinator node, a run of the algorithm can result in a transmission schedule such that all the N data packets are successfully transmitted to the coordinator (and, hence, 100% reliability is achieved) while another run can result in no successful transmissions at all (e.g. due to repeated collisions). Obviously, different outcomes have, in general, different probabilities to occur.

The ECC algorithm is able to generate all the possible *outcomes* an execution of CSMA/CA algorithm can yield, and the corresponding probabilities. However, to reduce the complexity of the analysis, it is possible to instruct the ECC algorithm to only generate the outcomes having a probability to occur greater than or equal to a certain threshold θ ($0 \leq \theta < 1$). The set of possible outcomes produced by the algorithm is then used to calculate the performance metrics of interest such as delivery ratio, latency and energy consumption.

The ECC algorithm is based on the observation that an outcome of the CSMA/CA execution can always be represented as a series (*chain*) of successful/failure transmissions (*events*) occurring subsequently in the network. Figure 3.1 shows a possible outcome o_x of CSMA/CA algorithm representing the case when a transmission failure (time $t = t_1$)

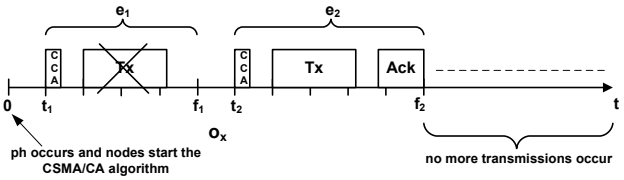


FIGURE 3.1: o_x, a possible outcome of the CSMA/CA.

followed by a successful transmission (time $t = t_2$) occur. Please note that time $t = 0$ is the instant at which all the nodes start the CSMA/CA execution and that, in this specific example, no more transmissions occur in the network after those shown in the figure.

Let us indicate as e_1 and e_2, respectively, the transmission failure and the successful transmission depicted in Figure 3.1. Then, the probability that outcome o_x occurs can be calculated as follows:

$$\mathbb{P}\{o_x\} = \mathbb{P}\{e_1 \wedge e_2 \wedge no\_txs\} \tag{3.1}$$

where no_txs indicates that no more events (successful/failure transmissions) occur in the network after e_2.

By recursively applying the Bayes' theorem, eq. 3.1 can be rewritten as:

$$\begin{aligned} \mathbb{P}\{o_x\} &= \mathbb{P}\{e_1 \wedge e_2 \wedge no\_txs\} \\ &= \mathbb{P}\{e_1 \wedge e_2\}\mathbb{P}\{no\_txs \mid e_1 \wedge e_2\} \\ &= \mathbb{P}\{e_1\}\mathbb{P}\{e_2 \mid e_1\}\mathbb{P}\{no\_txs \mid e_1 \wedge e_2\} \end{aligned} \tag{3.2}$$

More generally, the probability of an outcome o_i, representing the series of events $e_1,\ e_2,\ ...,\ e_n$, can be calculated as follows:

$$\begin{aligned} \mathbb{P}\{o_i\} &= \mathbb{P}\{e_1 \wedge e_2 \wedge ... \wedge e_n \wedge no\_txs\} \\ &= \mathbb{P}\{e_1 \wedge\ ... \wedge e_n\}\mathbb{P}\{no\_txs \mid e_1 \wedge ... \wedge e_n\} \\ &= \mathbb{P}\{e_1\}\mathbb{P}\{e_2 \mid e_1\}\mathbb{P}\{e_3 \mid e_1 \wedge e_2\} \cdot ... \\ &\cdot\ \mathbb{P}\{no\_txs \mid e_1 \wedge ... \wedge e_n\} \end{aligned} \tag{3.3}$$

Eq. 3.3 suggests that, in order to calculate the probability of outcome o_i, $n+1$ different steps have to be performed. First, $\mathbb{P}\{e_1\}$ has to be computed, i.e. the probability that e_1 is the first event occurring in the network. Then, at each subsequent step k, $2 \leq k \leq n$, the probability $\mathbb{P}\{e_k \mid e_1 \wedge ... \wedge e_{k-1}\}$ that event e_k occurs, given that all the previous

$k-1$ events have occurred, has to be derived. Finally, $\mathbb{P}\{no\_txs \mid e_1 \wedge ... \wedge e_n\}$ has to be calculated, i.e. the probability that no other events occur in the network after e_n. The ECC algorithm follows exactly these $n+1$ steps to compute the probability of an outcome.

Now, by means of a simple example, we give an overview of the actions performed by ECC to generate all the possible outcomes of a CSMA/CA execution and the corresponding probabilities.

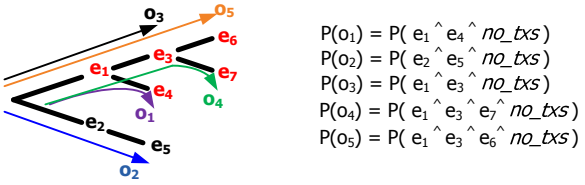


FIGURE 3.2: Possible outcomes of the CSMA/CA.

Figure 3.2 represents a case where a CSMA/CA execution can produce five possible outcomes, namely o_1, o_2, o_3, o_4, o_5. Initially (i.e. just after time $t=0$), two possible events can occur, namely e_1 and e_2. The ECC algorithm calculates that, with probability $\mathbb{P}\{e_1\}$, e_1 is the first event to occur in the network while, with probability $\mathbb{P}\{e_2\}$, e_2 occurs. Then, the algorithm proceeds by checking if there are cases in which no other events occur in the network after e_1 or e_2 have occurred, i.e. if there are outcomes of the algorithm composed only of event e_1 or e_2. To this end, both $\mathbb{P}\{no\_txs \mid e_1\}$ and $\mathbb{P}\{no\_txs \mid e_2\}$ are calculated. Since there are no outcomes terminating with e_1 or e_2, both $\mathbb{P}\{no\_txs \mid e_1\}$ and $\mathbb{P}\{no\_txs \mid e_2\}$ are equal to 0.

ECC then derives the events that can occur after e_1 or e_2. It discovers that, with probability $\mathbb{P}\{e_3 \mid e_1\}$, e_3 will follow e_1, while with probability $\mathbb{P}\{e_4 \mid e_1\}$, e_4 will follow e_1. Also, event e_5 is the only event that can occur in the netwok after e_2, i.e. $\mathbb{P}\{e_5 \mid e_2\} = 1$. As before, ECC checks if there are cases in which no other events occur in the network after e_3, e_4 or e_5 by computing $\mathbb{P}\{no\_txs \mid e_1 \wedge e_3\}$, $\mathbb{P}\{no\_txs \mid e_1 \wedge e_4\}$, $\mathbb{P}\{no\_txs \mid e_2 \wedge e_5\}$. It discovers that both $\mathbb{P}\{no\_txs \mid e_1 \wedge e_4\}$ and $\mathbb{P}\{no\_txs \mid e_2 \wedge e_5\}$ are equal to 1, since no events can occur after e_4 and e_5, while $0 < \mathbb{P}\{no\_txs \mid e_1 \wedge e_3\} < 1$, i.e. there are cases in which no other events occur in the network after e_1 and e_3. Thus, the algorithm stores three different outcomes namely o_1, o_2, o_3 and the corresponding probabilities $\mathbb{P}\{o_1\}$, $\mathbb{P}\{o_2\}$, $\mathbb{P}\{o_3\}$ given by eq. 3.3.

Since $\mathbb{P}\{no\_txs \mid e_1 \wedge e_3\}$ is not equal to 1, there are cases in which other events may occur in the network after e_1 and e_3. Specifically, e_6 occurs with probability $\mathbb{P}\{e_6 \mid e_1 \wedge e_3\}$ while e_7 occurs with probability $\mathbb{P}\{e_7 \mid e_1 \wedge e_3\}$. Also, since no other events can occur after e_6 and e_7 both $\mathbb{P}\{no\_txs \mid e_1 \wedge e_3 \wedge e_6\}$ and $\mathbb{P}\{no\_txs \mid e_1 \wedge e_3 \wedge e_7\}$ are equal to 1. Hence, the algorithm stores outcomes o_4 and o_5 with $\mathbb{P}\{o_4\}$ and $\mathbb{P}\{o_5\}$ calculated according to eq. 3.3. Then, it terminates.

As mentioned above, to reduce the complexity of the analysis, the ECC algorithm can generate only outcomes with probability greater than, or equal to, a certain threshold θ ($0 \leq \theta < 1$). In this case, the algorithm stops to analyze a certain sequence of events as soon as it discovers that its probability to occur is lower than θ. For instance, let us assume that $\mathbb{P}\{e_1\} = 0.95$ while $\mathbb{P}\{e_2\} = 0.05$. In case $\theta = 0.1$, the algorithm analyzes only the sequences composed of events highlighted in red in figure 3.2, i.e. the outcomes starting with event e_1. This is because, since $\mathbb{P}\{e_2\} = 0.05$, all the outcomes starting with event e_2 will have a probability to occur lower than or equal to $0.05 < \theta$.

The actions performed by the ECC algorithm are summarized by the flowchart depicted in Figure 3.4. Before describing it in detail we need to define the concept of *event* and *chain of events*.

Definition 1. An **event** $e_i = \{T_i,\ t_i\}$ represents a transmission occurring in the network, where T_i indicates the *type* of the event and t_i denotes its *starting time*. The event type can be either a success ($T_i = S$) or a failure ($T_i = F$). A success occurs whenever a node successfully transmits its packet, while a failure happens when two or more nodes transmit their packets simultaneously and, therefore, a collision occurs. The *starting time* t_i of an event e_i is defined as the time instant at which the node(s) causing e_i start their CCA. Each event e_i is also associated with a *finish time* f_i, defined as the first time instant, following e_i, at which a new event can occur, i.e. as the time $t^* > t_i$ such that a (new) successful CCA can be performed.

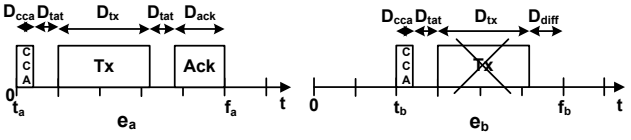


FIGURE 3.3: Events e_a and e_b.

Figure 3.3 depicts two possible events, i.e., a successful transmission e_a and a transmission failure e_b (time is divided in slots of duration equal to the backoff period, D_{bp}).

In Figure 3.3, D_{cca} is the duration of CCA, D_{tat} is the turnaround time (i.e., the time needed to switch the radio from receive to transmission mode, or vice versa), D_{ack} is the time to transmit/receive an acknowledgement and $D_{diff} = D_{bp} - (D_{tx} \bmod D_{bp})$ is the time between the end of a packet transmission and the beginning of the next slot. In this example, the starting time of event e_a is 0, while the starting time of event e_b is $t_b = 2 \cdot D_{bp}$. Hence, we can refer to e_a as $\{S,\ 0\}$ and to e_b as $\{F,\ 2D_{bp}\}$.

Definition 2. A **chain of events** $c = \{s_c,\ p_c,\ en_c\}$ represents a sequence of events that occur subsequently in the network. It is characterized by:

- the event sequence $s_c = \{e_1, ..., e_m\}$, where m denotes the total number of events occurred;
- the aggregate probability $p_c = \mathbb{P}\{e_1 \wedge e_2 \ ... \ \wedge e_m\} = \mathbb{P}\{e_1\}\mathbb{P}\{e_2 \mid e_1\} \ ... \ \mathbb{P}\{e_m \mid e_1 \wedge e_2 \ ... \wedge e_{m-1}\}$ that the event sequence s_c occurs;
- the average total energy en_c spent by all the nodes in the network during the time interval $[0,\ f_m]$, where f_m is the finish time of the last event in sequence s_c. □

Please note that a chain $c: \ s_c = \{e_1, ..., e_m\}$ represents a possible *outcome* of the CSMA/CA execution iff $\mathbb{P}\{no\_txs \mid e_1 \wedge ... \wedge e_m\} > 0$.

Hereafter, for brevity, we indicate as $\mathbb{P}\{e_x \mid c\} = \mathbb{P}\{e_x \mid e_1 \wedge ... \wedge e_m\}$ the probability that event e_x occurs in the network, given that the sequence of events $s_c = \{e_1, ..., e_m\}$ has occurred. Also, we denote by $\mathbb{P}\{no\_txs \mid c\} = \mathbb{P}\{no\_txs \mid e_1 \wedge ... \wedge e_m\}$ the probability that no other events occur in the network after the sequence represented by chain c.

To derive all the outcomes of the CSMA/CA execution, and the related probability, ECC follows an iterative approach summarized in Figure 3.4. Initially, ECC creates two empty sets, namely L_c and F_c. At a given point in time, L_c contains the chains still to be analyzed by the algorithm, while F_c contains chains representing possible outcomes of the CSMA/CA execution. ECC starts analyzing the network at time $t = 0$ and derives all the possible events e_i that can occur just after $t = 0$. For each such event e_i, the chain $c: s_c = \{e_i\}$ is added to set L_c, iff $p_c = \mathbb{P}\{e_i\} \geq \theta$. Then, the ECC algorithm enters a loop that ends when there are no more chains to be analyzed, i.e., $L_c = \{\emptyset\}$. At each iteration, a chain $c: s_c = \{e_1,\ e_2, ...,\ e_m\}$ is extracted from set L_c to be analyzed. First, the algorithm checks if c can be a possible outcome of the CSMA/CA execution,

i.e. if $\mathbb{P}\{no\_txs \mid c\} > 0$. If so, the following operations are performed. First, a copy c_x of chain c is created. Second, since no more events have to occur in the network to consider c_x an outcome, the probability of chain c_x is updated as $p_{c_x} = p_c \cdot \mathbb{P}\{no\_txs \mid c\}$. Finally, if $p_{c_x} \geq \theta$, the average energy en_{c_x} spent by the nodes in the network when the events reported by c_x occur is calculated and c_x is added to set F_c since it represents a possible outcome of the algorithm having a probability to occur $\geq \theta$.

If $\mathbb{P}\{no\_txs \mid c\} \neq 1$, it means that other events can occur in the network after those in c. In this case, the algorithm derives all the events e_x (and their probability $\mathbb{P}\{e_x \mid c\}$) that can occur *after* the last event e_m in chain c. For each such event e_x, the chain $c_x : s_{c_x} = \{e_1, e_2, ..., e_m, e_x\}$ is added to set L_c, provided that the corresponding probability $p_{c_x} = \mathbb{P}\{e_x \mid c\} \cdot p_c$ is greater than, or equal to, θ. When the set L_c becomes empty it means that ECC has generated all the possible outcomes having a probability

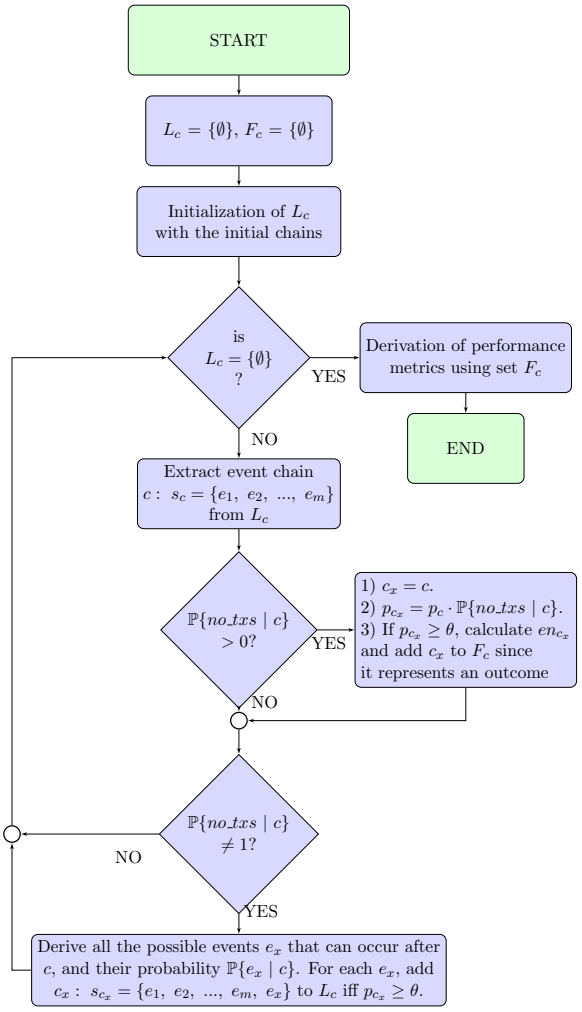


FIGURE 3.4: Steps performed by the ECC algorithm.

to occur greater than, or equal to, θ. Hence, the ECC algorithm proceeds with deriving the performance metrics of interest, using the chains in set F_c. Then, it terminates its execution.

3.4 Model derivation

In this section we detail each single step of the ECC algorithm. After performing a preliminary analysis in section 3.4.1, in Section 3.4.2 we focus on the ECC initialization phase and derive all the possible events that can occur in the network just after time $t = 0$ and the corresponding probabilities. Then, in Section 3.4.3, we focus on the actions performed by the ECC algorithm inside the loop (chains examination phase). Finally, in Section 3.4.4 we show how to parallelize ECC while in Section 3.4.5 we derive performance metrics of interest using chains in set F_c.

3.4.1 Preliminaries

Before proceedings into the ECC algorithm in details, we derive a general formula for the probability that a sensor node performs a CCA (Clear Channel Assessment) at a given time t. To this end, we first derive the possible time instants at which a sensor node could start a CCA. Next, we compute the probability that it actually performs a CCA in one of these time instants.

As a preliminary step, we need to consider all the actions that may lead a node to start a CCA at a given time t. Let $B_{max} = macMaxCSMABackoffs + 1$ denote the maximum number of consecutive CCAs allowed for each transmission attempt, and $T_{max} = macMaxFrameRetries + 1$ be the maximum number of transmission attempts allowed per data packet. In addition, let us indicate by W_i, $1 \leq i \leq B_{max}$, the backoff window size at the i-th backoff stage. For simplicity, hereafter we will use the expression "the sensor node is in state B_{ij}" to indicate that a sensor node is performing a CCA during the i-th backoff stage of the j-th transmission attempt. Now we derive the set Λ_{ij} of all the possible instants at which a sensor node could start a CCA while in state B_{ij}.

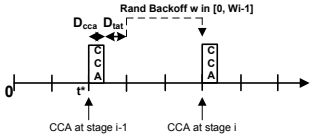


FIGURE 3.5: CCA due to a previous failed CCA

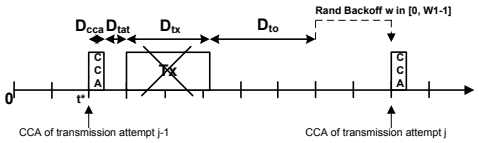


FIGURE 3.6: CCA due to a failed tx attempt

According to the CSMA/CA algorithm, at $t = 0$, each sensor node waits for a random number $w \in [0,\ W_1 - 1]$ of backoff periods and, then, it performs a CCA. Hence, $\Lambda_{11} = \{0, D_{bp}, \cdots, (W_1 - 1) \cdot D_{bp}\}$. Then, a sensor node can start a CCA after one of the two following events: **(i)** a previous CCA during which the channel was found busy (see Figure 3.5), or **(ii)** an unsuccessful transmission attempt (see Figure 3.6). In the former case, the sets $\Lambda_{ij},\ 2 \leq i \leq B_{max}, 1 \leq j \leq T_{max}$, can be recursively derived from Λ_{i-1j} as follows:

$$\begin{aligned}\Lambda_{ij} =& \{t \mid \exists w \in [0, W_i - 1] \wedge \exists t^* \in \Lambda_{i-1j} \\ & \wedge t = t^* + D_{cca} + D_{tat} + w \cdot D_{bp}\}\end{aligned} \tag{3.4}$$

Equation 3.4 derives CCA instants in set Λ_{ij} by considering all the CCA instants $t^* \in \Lambda_{i-1j}$ and all the possible random backoff values $w \in [0, W_i - 1]$ a node can generate when it is in the i-th backoff stage and has found the channel busy.

In case **(ii)** the sensor node performs a CCA due to a previous unsuccessful transmission attempt and, hence, it is in one of the states $B_{1j}, 2 \leq j \leq T_{max}$. Let us denote by $D_{rtx} \triangleq D_{cca} + D_{tat} + D_{tx} + D_{to}$ the total time needed to perform a CCA (D_{cca}), turn the radio in TX mode (D_{tat}), transmit a data packet (D_{tx}), and wait for the timeout (D_{to}). We indicate as R_{ij-1} the set of instants at which a node could perform a CCA after an unsuccessful transmission started at any $t^* \in \Lambda_{ij-1}, 1 \leq i \leq B_{max}$. It can be expressed as follows.

$$\begin{aligned}R_{ij-1} =& \{t \mid \exists w \in [0, W_1 - 1] \wedge \exists t^* \in \Lambda_{ij-1} \\ & \wedge t = t^* + D_{rtx} + w \cdot D_{bp}\}\end{aligned} \tag{3.5}$$

Equation 3.5 calculates R_{ij-1} by taking into account all possible time instants $t^* \in \Lambda_{ij-1}$ and all possible backoff values $w \in [0, W_1 - 1]$ the node can generate at the first backoff stage. Since a retransmission can occur during every backoff stage, the set $\Lambda_{1j}, 2 \leq j \leq T_{max}$, is computed as the union of all the sets R_{ij-1}, i.e., $\Lambda_{1j} = \bigcup_{i=1}^{B_{max}} R_{ij-1}$.

Let Ω_{ij}^t denote the set of all time instants t^* at which a node can perform a CCA before performing a CCA at time t during state B_{ij}. The following claim holds.

Claim 1. The set Ω_{ij}^t can be derived as

$$\Omega_{ij}^t = \begin{cases} \{\emptyset\}, & \text{if } i = 1,\ j = 1 \\ \{t^* \in \Lambda_{i-1j} \mid \exists w \in [0, W_i - 1] \wedge \\ \qquad t = t^* + D_{cca} + D_{tat} + w \cdot D_{bp}\}, & \\ & \text{if } 2 \leq i \leq B_{max},\ 1 \leq j \leq T_{max} \\ \{t^* \in \Lambda_{i'j-1},\ 1 \leq i' \leq B_{max} \mid \\ \exists w \in [0, W_1 - 1] : t = t^* + D_{rtx} + w \cdot D_{bp}\}, & \\ & \text{if } i = 1,\ 2 \leq j \leq T_{max} \end{cases} \tag{3.6}$$

Proof. The set Ω_{11}^t is empty since no CCA can be performed before those occurring at time instants in set Λ_{11}. If $2 \leq i \leq B_{max}$, it means that the node performs a CCA at time t due to a previous failed CCA. In this case, all the CCA instants $t^* \in \Lambda_{i-1j}$ that can result in a CCA at t are selected (second term of Equation 3.6). In the last case, a sensing at time t is due to a previous unsuccessful transmission. Therefore, the set $\Omega_{1j}^t, 2 \leq j \leq T_{max}$, is composed by all the $t^* \in \Lambda_{i'j-1}, 1 \leq i' \leq B_{max}$, which could cause the node to perform a CCA at t due to an unsuccessful transmission (third term of Equation 3.6). □

Let us now derive the probability $\mathbb{P}\{\text{CCA}^t\}$ that a sensor node performs a CCA at time t. To this end, we calculate the probability $\mathbb{P}\{\text{CCA}_{ij}^t\}$ that a node performs a CCA at time t while in state B_{ij} and, then, we compute $\mathbb{P}\{\text{CCA}^t\}$ based on $\mathbb{P}\{\text{CCA}_{ij}^t\}$. Let us denote by $\mathbb{P}\{\text{CB}^t\}$ the probability to find the channel busy during a CCA started at time t, and by $\mathbb{P}\{\text{F}^t\}$ the probability that a transmission whose CCA started at time t fails. The following claims hold.

Claim 2. The probability $\mathbb{P}\{\mathrm{CCA}_{ij}^{t}\}$ that a sensor node performs a CCA at t while in state B_{ij} is

$$\mathbb{P}\{\mathrm{CCA}_{ij}^{t}\} = \begin{cases} 0 & \text{if } t \notin \Lambda_{ij} \\ \frac{1}{W_1}, & \text{if } i = 1,\ j = 1 \\ \sum_{t^* \in \Omega_{ij}^{t}} \mathbb{P}\{\mathrm{CCA}_{i-1j}^{t^*}\} \cdot \mathbb{P}\{\mathrm{CB}^{t^*}\} \cdot \frac{1}{W_i}, & \\ \qquad \text{if } 2 \leq i \leq B_{max}, 1 \leq j \leq T_{max} & \\ \sum_{t^* \in \Omega_{ij}^{t}} \sum_{i'=1}^{B_{max}} \mathbb{P}\{\mathrm{CCA}_{i'j-1}^{t^*}\} \cdot & \\ \qquad \cdot (1 - \mathbb{P}\{\mathrm{CB}^{t^*}\}) \cdot \mathbb{P}\{\mathrm{F}^{t^*}\} \cdot \frac{1}{W_1}, & \\ \qquad \text{if } i = 1,\ 2 \leq j \leq T_{max} & \end{cases} \tag{3.7}$$

Proof. In case $t \notin \Lambda_{ij}$, it is not possible for a node to perform a CCA at time t during state B_{ij}. Therefore, in this case, $\mathbb{P}\{\mathrm{CCA}_{ij}^{t}\} = 0,\ \forall(i,\ j)$. Let us consider now the cases when $t \in \Lambda_{ij}$. If $i = j = 1$, it means that $t \in \Lambda_{11}$. Since there are W_1 time instants in set Λ_{11} and, while in state B_{11}, each node can start a CCA randomly at one of these instants, $\mathbb{P}\{\mathrm{CS}_{ij}^{t}\} = 1/W_1$ in this case.

The third term of Equation calculates the probability that a node performs a CCA at time t while in state $B_{ij},\ 2 \leq i \leq B_{max}, 1 \leq j \leq T_{max}$, i.e. after an unsuccessful CCA performed during state B_{i-1j}. To this end, the term computes $\mathbb{P}\{\mathrm{CCA}_{ij}^{t}\}$ taking into consideration all the possible CCA instants $t^* \in \Omega_{ij}^{t}$ (of the backoff stage $i-1$), which could lead the node to perform a CCA at time t. Specifically, for each $t^* \in \Omega_{ij}^{t}$ the term performs the product between **(i)** the probability $\mathbb{P}\{\mathrm{CCA}_{i-1j}^{t^*}\}$ to start a CCA operation at time t^* while in state B_{i-1j}, **(ii)** the probability to find the channel busy at time t^* ($\mathbb{P}\{\mathrm{CB}^{t^*}\}$), **(iii)** the probability to extract a backoff time such that a new CCA is performed at time t ($1/W_i$).

Finally, the fourth term in Equation computes the probability that a node performs a CCA at time t while in state $B_{1j}, 2 \leq j \leq T_{max}$, i.e. after an unsuccessful transmission attempt. In particular, the outer sum considers all the CCA instants $t^* \in \Omega_{1j}^{t}$ which may lead to a CCA at time t, due to an unsuccessful transmission attempt. Instead, the inner sum considers all the backoff stages $1 \leq i' \leq B_{max}$ of transmission attempt $j-1$. Then, for each couple $(t^*,\ i')$, the formula performs the product between **(i)** the probability $\mathbb{P}\{\mathrm{CCA}_{i'j-1}^{t^*}\}$ that the node starts a CCA at time t^* while in state $B_{i'j-1}$, **(ii)**

the probability to find the channel free during the CCA at time t^* ($(1-\mathbb{P}\{\text{CB}^{t^*}\})$ **(iii)** the probability that the transmission results into a failure, i.e. $\mathbb{P}\{\text{F}^{t^*}\}$, **(iv)** the probability to extract a backoff time such that a new CCA is performed at time t ($1/W_1$). □

Claim 3. The probability that a sensor node performs a CCA at a certain time t can be calculated as:

$$\mathbb{P}\{\text{CCA}^t\} = \sum_{i=1}^{B_{max}} \sum_{j=1}^{T_{max}} \mathbb{P}\{\text{CCA}_{ij}^t\} \tag{3.8}$$

Proof. $\mathbb{P}\{\text{CCA}^t\}$ is equal to the probability that a sensor node performs a CCA at time t in any state B_{ij}. Therefore, Equation 3.8 calculates $\mathbb{P}\{\text{CCA}^t\}$ as the sum of $\mathbb{P}\{\text{CCA}_{ij}^t\}, 1 \leq i \leq B_{max}, 1 \leq j \leq T_{max}$. Since events "performing a CCA at time t while in state B_{ab}" and "performing a CCA at time t while in state B_{cd}", $a \neq c \mid b \neq d$, are always mutually exclusive, it is possible to sum probabilities $\mathbb{P}\{\text{CCA}_{ij}^t\}$. □

3.4.2 ECC Initialization

As shown in Figure 3.4, the first step of the ECC algorithm consists in initializing the set L_c with chains derived from events occurring immediately after $t = 0$. In this section, we will refer to e_{s_i} (e_{f_i}) as the success (failure) event starting at time $i \cdot D_{bp},\ i \in \mathbb{N}$. Also, we denote by $\mathbb{P}\{e_{s_i}\}$ ($\mathbb{P}\{e_{f_i}\}$) the probability that event e_{s_i} (e_{f_i}) occurs.

According to the CSMA/CA algorithm, at $t = 0$ each sensor node waits for a random number $w \in [0,\ W_1 - 1]$ of backoff periods and, then, it performs a CCA. Therefore, the first event occurring in the network can be either a success or a failure with starting time in the set $\{0,\ D_{bp},\ 2D_{bp}, \ldots,\ (W_1 - 1) \cdot D_{bp}\}$.

A successful transmission occurs at time $i \cdot D_{bp}$ ($i = 0, .., W_1 - 1$) when one node generates a backoff time equal to $i \cdot D_{bp}$, and all the other $N - 1$ nodes extract a backoff time larger than $i \cdot D_{bp}$. Therefore,

$$\mathbb{P}\{e_{s_i}\} = N \cdot \frac{1}{W_1} \cdot \left(\frac{W_1 - i - 1}{W_1}\right)^{N-1} \tag{3.9}$$

In Equation (3.9), the term $1/W_1$ is the probability that one node picks up a backoff time equal to $i \cdot D_{bp}$, while the third term gives the probability that all the remaining $N - 1$ nodes generate a backoff time larger than $i \cdot D_{bp}$. Conversely, a failure occurs at

time $i \cdot D_{bp}$ when two or more nodes generate the same backoff time $i \cdot D_{bp}$ and, thus, experience a collision. Hence,

$$\mathbb{P}\{e_{f_i}\} = \sum_{k=2}^{N} \binom{N}{k} \left(\frac{1}{W_1}\right)^k \cdot \left(\frac{W_1 - i - 1}{W_1}\right)^{N-k} \tag{3.10}$$

The sum in Equation (3.10) takes into account that more than two nodes may collide. The term inside the sum gives the probability that exactly k nodes randomly pick up a backoff time of $i \cdot D_{bp}$, while $N - k$ nodes choose a backoff value larger than $i \cdot D_{bp}$.

Using Equations (3.9) and (3.10), ECC initializes L_c by adding chains c: $s_c = \{e_{s_i}\}$ ($s_c = \{e_{f_i}\}$) and $p_c = \mathbb{P}\{e_{s_i}\}$ ($p_c = \mathbb{P}\{e_{f_i}\}$). Note that a chain is added to L_c iff $p_c \geq \theta$. Then, ECC enters the *chains examination* phase.

3.4.3 Chains examination

In the chains examination phase, ECC executes a loop during which, at each step, a chain $c \in L_c$ with $s_c : \{e_1, ..., e_m\}$ is examined. The goal of the examination is twofold. First, the algorithm checks if c represents a possible outcome of the CSMA/CA execution by computing $\mathbb{P}\{no\_txs \mid c\}$ and, if so (i.e. $\mathbb{P}\{no\_txs \mid c\} > 0$), it adds c to F_c. If $\mathbb{P}\{no\_txs \mid c\} \neq 1$ it means that new events can occur after c. Hence, as a second step, all the events that may occur after e_m are derived (i.e. after the last event in c). Hereafter, for simplicity, we will indicate as e_{s_i} (e_{f_i}) a success (failure) event occurring at time $t_i = f_m + i \cdot D_{bp},\ i \in \mathbb{N}$, where f_m is the finish time of event e_m and as $\mathbb{P}\{e_{s_i} \mid c\}$ ($\mathbb{P}\{e_{f_i} \mid c\}$) its corresponding probability. For any e_{s_i} (e_{f_i}) a new chain $c_x : s_{c_x} = \{e_1, ..., e_m, e_{s_i}\}$ ($s_{c_x} = \{e_1, ..., e_m, e_{f_i}\}$) is added to L_c by the ECC algorithm iff $p_{c_x} = p_c \cdot \mathbb{P}\{e_{s_i} \mid c\} (p_c \cdot \mathbb{P}\{e_{f_i} \mid c\}) \geq \theta$.

In the following we show the computation of both $\mathbb{P}\{no\_txs \mid c\}$ and $\mathbb{P}\{e_{s_i} \mid c\}$ ($\mathbb{P}\{e_{f_i} \mid c\}$). However, before deriving them, we perform two preliminary steps. Specifically, we first derive $\mathbb{P}\{CCA^t \mid c\}$, i.e. the probability that a node, that has not experienced a success until f_m, will perform (has performed) a CCA at a certain time $t \geq f_m$ ($t < f_m$), given that the events in c occurred. Second, we compute the exact number of nodes that are still active in the network at time $t = f_m$ (i.e. that have not yet terminated the CSMA/CA execution).

Derivation of $\mathbb{P}\{CCA^t \mid c\}$

Hereafter, we take into consideration a generic chain $c: \ s_c = \{e_1, \ ..., \ e_m\}$ and denote by N_s the number of successful transmissions occurred in chain c ($N_s = |\{e_i \in s_c: \ T_i = S\}|$). Since each sensor node has to transmit just one data packet, at most $N_r = N - N_s$ nodes may be still active in the network after time f_m. Now our goal is to derive the probability $\mathbb{P}\{CCA^t \mid c\}$, that any of the N_r nodes will perform (has performed) a CCA at time t, given that the sequence of events in c occurred.

First of all, we denote by $N_{P_{ij}}, 1 \leq i \leq B_{max}, 1 \leq j \leq T_{max}$, the set of all time instants $t < f_m$ at which it is not possible, for any of the N_r sensor nodes, to have performed a CCA while in state B_{ij}, if the events in c occurred. The derivation of $N_{P_{ij}}$ is shown in the Appendix.

As a second step, we derive, for time instants $t < f_m$, [1], the probability $\mathbb{P}\{CB^t \mid c\}$ that any of the N_r sensor nodes has found the channel busy during a CCA started at time t. We also derive $\mathbb{P}\{F^t \mid c\}$, i.e. the probability that any of the same nodes has experienced a failure for a transmission whose CCA started at time t. The following equations hold.

$$\mathbb{P}\{CB^t \mid c\} = \begin{cases} 1 & \text{if } \exists e_i \in s_c : t \in [t_i + D_{bp}, \ f_i) \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

$$\mathbb{P}\{F^t \mid c\} = \begin{cases} 1 & \text{if } \exists e_i \in s_c : t_i = t \wedge T_i = F \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

The probability that a generic node has found the channel busy during a CCA started at time t only depends on the specific events in chain c. Specifically, $\mathbb{P}\{CB^t \mid c\} = 1$ if a success or failure event has occurred at time t, and zero otherwise. Similarly, $\mathbb{P}\{F^t \mid c\} = 1$ if a failure occurred at time $t_i = t$, and zero otherwise.

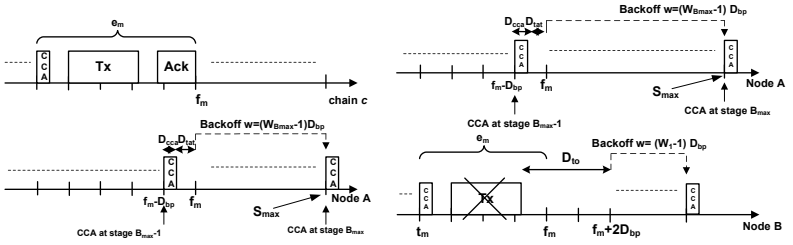
Finally, we indicate as S_{max} ($S_{max} > f_m$), the largest time instant at which any of the N_r nodes can perform a CCA, given that all the events in c occurred. S_{max} represents the largest instant at which a new event can occur after e_m. The following claim holds.

[1] We assume that both $\mathbb{P}\{CB^t \mid c\}$ and $\mathbb{P}\{F^t \mid c\}$ are equal to 0 $\forall t \geq f_m$. This allows to calculate the probability that a node will *directly* perform a CCA at a time $t \geq f_m$.

Claim 4. $S_{max} = f_m + M_w \cdot D_{bp}$, where

$$M_w = \begin{cases} W_{B_{max}} - 1 & T_m = S \\ \max\{W_{B_{max}} - 1,\ 2 + (W_1 - 1)\} & T_m = F \end{cases} \tag{3.13}$$

Proof. If $T_m = S$, S_{max} is derived by considering the worst case shown in fig. 7(a) where a node (node A) performs a CCA at time $t = f_m - D_{bp}$ during its $(B_{max} - 1)$-th backoff stage, finds the channel busy, and extracts a value for the backoff time equal to $(W_{B_{max}} - 1) \cdot D_{bp}$. Hence, S_{max} is given by $f_m + (W_{B_{max}} - 1) \cdot D_{bp}$. Conversely, if $T_m = F$, two cases must be considered. A node (node A in fig. 7(b)) may perform a CCA at time $t = f_m - D_{bp}$ during the $(B_{max} - 1)$-th backoff stage, and extract a backoff time value equal to $(W_{B_{max}} - 1) \cdot D_{bp}$. At the same time, another node (node B in fig. 7(b)), which has experienced the collision represented by e_m, waits for the retransmission timeout D_{to} and extracts a backoff time equal to $(W_1 - 1) \cdot D_{bp}$. Hence, we need to consider the largest value for S_{max} in the two cases, i.e. $S_{max} = f_m + max((W_{B_{max}} - 1),\ 2 + (W_1 - 1)) \cdot D_{bp}$. □



(a) $T_m = S$ (b) $T_m = F$

FIGURE 3.7: Derivation of S_{max}

Now, we can show the computation of $\mathbb{P}\{CCA^t \mid c\}$ for time instants $t \in [0,\ S_{max}]$. The following claim holds.

Claim 5. The probability $\mathbb{P}\{CCA_{ij}^t \mid c\}$ that any of the N_r nodes has performed a CCA at time $t \in [0,\ f_m]$ or will perform a CCA at a time $t \in [f_m,\ S_{max}]$, while in state B_{ij},

provided that all the events in chain c have occurred, is

$$\begin{cases} 0 & if\ t \notin \Lambda_{ij} \vee t \in N_{P_{ij}} \\ \frac{1}{|\Lambda_{11} \setminus N_{P_{11}}|}, & if\ i = 1,\ j = 1 \\ \sum_{t^* \in (\Omega_{ij}^t \setminus N_{P_{i-1j}})} \mathbb{P}\{\mathrm{CCA}_{i-1j}^{t^*} \mid c\} \cdot \mathbb{P}\{\mathrm{CB}^{t^*} \mid c\} \cdot \frac{1}{|h_{ij}^{t^*}|}, \\ \qquad if\ 1 < i \leq B_{max}, 1 \leq j \leq T_{max} \\ \sum_{i'=1}^{B_{max}} \sum_{t^* \in (\Omega_{ij}^t \setminus N_{P_{i'j-1}})} \mathbb{P}\{\mathrm{CCA}_{i'j-1}^{t^*} \mid c\} \cdot (1 - \mathbb{P}\{\mathrm{CB}^{t^*} \mid c\}) \cdot \\ \qquad \cdot\ \mathbb{P}\{\mathrm{F}^{t^*} \mid c\} \cdot \frac{1}{|h_{1j}^{t^*}|}, \\ \qquad if\ i = 1,\ 2 \leq j \leq T_{max} \end{cases} \tag{3.14}$$

Proof. If $t \notin \Lambda_{ij}$ or $t \in N_{P_{ij}}$, then $\mathbb{P}\{CCA_{ij}^t \mid c\} = 0$ by definition of Λ_{ij} and $N_{P_{ij}}$.

Now, we consider cases where $t \in \Lambda_{ij} \wedge t \notin N_{P_{ij}}$. In case $i = j = 1$, we are considering CCAs performed in state B_{11}. At the beginning, each node randomly chooses a backoff time in set $\{0,\ D_{bp},\ 2 \cdot D_{bp},\ ...,\ (W_1 - 1)D_{bp}\}$ and, then, performs a CCA. Since we are assuming that the events in chain c occurred, we have to exclude CCA instants contained in set $N_{P_{11}}$. Specifically, the number of possible CCA instants during state B_{11} can be calculated as $|\Lambda_{11} \setminus N_{P_{11}}|$, where $\setminus$ is the set difference operator. Hence, the second term of the formula computes probability $\mathbb{P}\{CCA_{11}^t \mid c\}$ as $\frac{1}{|\Lambda_{11} \setminus N_{P_{11}}|}$.

Let us consider now states $B_{ij},\ 1 < i \leq B_{max},\ 1 \leq j \leq T_{max}$. In this case the node performs a CCA at time t after a previous unsuccessful CCA operation during state B_{i-1j}. The third term of the formula takes into consideration all the possible CCA instants t^* which could lead the node to perform a CCA at time t. However, we need to exclude instants when a CCA cannot have occurred while in state B_{i-1j}. Hence, only instants in the set $(\Omega_{ij} \setminus N_{P_{i-1j}})$ are considered in the third term. Then, inside the sum, the product between 1) the probability that a node has performed a CCA at time t^* while state B_{i-1j}, given that events in c have occurred ($\mathbb{P}\{\mathrm{CCA}_{i-1j}^{t^*} \mid c\ \}$), 2) the probability that the same node found the channel busy at time t^* ($\mathbb{P}\{\mathrm{CB}^{t^*} \mid c\}$), 3) the probability that the same node has extracted a backoff time such that a new CCA is performed at time t ($\frac{1}{|h_{ij}^{t^*}|}$) is performed. The set $h_{ij}^{t^*}$ contains all possible CCA instants at which a node may have performed a CCA, while in state B_{ij}, after an unsuccessful CCA at time t^* during state B_{i-1j}. The set $h_{ij}^{t^*}$ is calculated by subtracting set $N_{P_{ij}}$ to

set $\{t : \exists w \in [0,\ W_i - 1] \wedge t = t^* + D_{cca} + D_{tat} + w \cdot D_{bp}\}$, i.e. the set of all the possible CCA instants following a failed CCA at time t^* during state B_{i-1j}.

Finally, the last case refer to the case when $i = 1,\ 2 \leq j \leq T_{max}$, i.e. when a node performs a CCA after a transmission failure. The term is composed of two sums. The first one considers all the possible backoff stages i', $1 \leq i' \leq B_{max}$ of the $(j-1)$-th transmission attempt. Instead, the second one considers all the possible CCA instants t^* that may generate a CCA at time t due to a previous unsuccessful transmission. Specifically, the inner sum only considers instants $t^* \in (\Omega_{ij}^t \setminus N_{P_{i'j}})$, i.e., it does not consider all instants when a CCA cannot occur while in state $B_{i'j}$. Then, for each couple $(i',\ t^*)$ the product between 1) the probability $\mathbb{P}\{\mathrm{CCA}_{i'j-1}^{t^*} \mid c\}$ that the node performs a CCA at time t^* during state $B_{i'j}$ given the events in c occurred, 2) the probability that the same node has found the channel idle at time t^*, i.e. $(1 - \mathbb{P}\{\mathrm{CB}^{t^*} \mid c\})$, 3) the probability that the transmission results into a collision, i.e. $\mathbb{P}\{\mathrm{F}^{t^*} \mid c\}$ and 4) the probability $\frac{1}{|h_{1j}^{t^*}|}$ to extract a backoff time such that the node performs a new CCA at time t, is performed. In the formula, $h_{1j}^{t^*} = \{t : \exists w \in [0,\ W_1 - 1] \wedge t = t^* + D_{rtx} + w \cdot D_{bp}\} \setminus N_{P_{ij}}$ represents the set of all possible instants when a node may have performed a CCA during state B_{1j} after experiencing a collision for a transmission whose CCA started at t^*.

□

Finally, we can derive $\mathbb{P}\{CCA^t \mid c\}$ as follows

$$\mathbb{P}\{\mathrm{CCA}^t \mid c\} = \sum_{i=1}^{B_{max}} \sum_{j=1}^{T_{max}} \mathbb{P}\{\mathrm{CCA}_{ij}^t \mid c\} \tag{3.15}$$

Estimating the number of active nodes at time $t = f_m$

Since N_s nodes experienced a success during the chain c, at most $N_r = N - N_s$ nodes can be potentiallly active at time $t = f_m$. Each of these N_r nodes can be in one of the following states at time $t = f_m$: **(i)** the node is really active, i.e. it has not yet finished the CSMA/CA execution; **(ii)** the node has reached the maximum number B_{max} of consecutive CCAs for a data packet transmission; or **(iii)** the node has reached the maximum number T_{max} of retransmissions for a data packet. Indeed, in the last two cases the sensor node drops its data packet, according to the CSMA/CA algorithm and, thus, it is no longer active at time $t = f_m$. To derive the number of sensor nodes that

are really active at time $t = f_m$, we need to calculate the probability, for each of the N_r nodes, to be in state **(i)**, **(ii)** or **(iii)**, respectively. The following claims hold.

Claim 6. Let $\mathbb{P}\{F_{CCA} \mid c\}$ denote the probability that any of the N_r sensor nodes has exceeded the maximum number B_{max} of consecutive CCAs allowed for the transmission of a data packet before time $t = f_m$. It is

$$\mathbb{P}\{F_{CCA} \mid c\} = \sum_{t=0}^{f_m - D_{bp}} \sum_{j=1}^{T_{max}} \mathbb{P}\{CCA^t_{B_{max}j} \mid c\} \cdot \mathbb{P}\{CB^t \mid c\} \tag{3.16}$$

Proof. A data packet is dropped by a sensor node due to exceeded number of CCAs when it performs a CCA during one of the states $B_{B_{max}j}, 1 \leq j \leq T_{max}$, and finds the channel busy. Hence, Equation 3.16 considers all the possible CCA instants $t \in [0,\ f_m - D_{bp}]$ (outer sum) and the last backoff stage of each transmission attempt j (inner sum). Then, $\mathbb{P}\{F_{CCA} \mid c\}$ is calculated by summing up, for each time instant t, the probability that the sensor node: (i) performed a CCA at time t during the last backoff stage, and (ii) found the channel busy, i.e. $\mathbb{P}\{CCA^t_{B_{max}j} \mid c\} \cdot \mathbb{P}\{CB^t \mid c\}$. □

Claim 7. Let $\mathbb{P}\{F_{Rtx} \mid c\}$ denote the probability that any of the N_r sensor nodes has exceeded the number of retransmissions allowed for a data packet before time $t = f_m$. It is

$$\mathbb{P}\{F_{Rtx} \mid c\} = \sum_{t=0}^{f_m - D_{bp}} \sum_{i=1}^{B_{max}} \mathbb{P}\{CCA^t_{iT_{max}} \mid c\} \cdot \mathbb{P}\{F^t \mid c\} \tag{3.17}$$

Proof. A data packet is dropped by a sensor node, due to exceeded number of retransmissions, if it experiences a collision during one of the states $B_{iT_{max}}, 1 \leq i \leq B_{max}$. Hence, the equation considers all the possible CCA instants $t \in [0,\ f_m - D_{bp}]$ (outer sum) and any backoff stage $i,\ 1 \leq i \leq B_{max}$, of the last transmission attempt (inner sum). Then, $\mathbb{P}\{F_{Rtx} \mid c\}$ is calculated by summing up, for each time instant t, the probability that the sensor node: (i) performed a CCA at time t during the last transmission attempt (during any backoff stage i), and (ii) it experienced a collision, i.e. $\mathbb{P}\{CCA^t_{iT_{max}} \mid c\} \cdot \mathbb{P}\{F^t \mid c\}$.

□

Claim 8. Let $\mathbb{P}\{A_p \mid c\}$ denote the probability that any of the N_r nodes: (i) is still active at $t = f_m$ and (ii) it has participated to the last event e_m in chain c, i.e. it has

performed a CCA at time $t = t_m$. It is

$$\mathbb{P}\{A_p \mid c\} = \begin{cases} 0 & T_m = S \\ \sum_{i=1}^{B_{max}} \sum_{j=1}^{T_{max}-1} \mathbb{P}\{\mathrm{CCA}_{ij}^{t_m} \mid c\} & T_m = F \end{cases} \tag{3.18}$$

Proof. Equation 3.18 considers two different cases. The first one regards the case when the last event e_m in the chain is a successful event, i.e. $T_m = S$. Since the considered nodes have not successfully transmitted their packet, it is not possible for any of them to have performed a CCA at time $t = t_m$. Hence, $\mathbb{P}\{A_p \mid c\}$ is equal to 0 in this case. Let us consider now the case when $T_m = F$, i.e. e_m is a failure event. In this case, the probability for a node to be still active in the network and to have participated to event e_m can be derived by calculating the probability that the node has performed a CCA at time t_m during a transmission attempt different from the last one. Hence, in this case, $\mathbb{P}\{A_p \mid c\}$ is calculated as the sum of probabilities that the node has performed a CCA at time t_m during any backoff stage i and transmission attempt j, $j \leq T_{max} - 1$. □

Claim 9. Let $\mathbb{P}\{A_{np} \mid c\}$ denote the probability that any of the N_r sensor nodes: (i) is still active at time $t = f_m$, and (ii) it has not participated to the last event e_m in chain c. Hence,

$$\mathbb{P}\{A_{np} \mid c\} = \sum_{t=f_m}^{S_{max}} \mathbb{P}\{CCA^t \mid c\} - \mathbb{P}\{A_p \mid c\} \tag{3.19}$$

Proof. The probability that any of the considered sensor nodes is still active at time $t = f_m$ is equal to the probability that the same node can perform a CCA after time f_m, i.e., at any time $t \in [f_m,\ S_{max}]$. Hence, in the equation, the sum of probabilities to perform a CCA at any time $t \in [f_m,\ S_{max}]$, during any state B_{ij}, is considered (first term). However, to compute correctly $\mathbb{P}\{A_{np} \mid c\}$, we need to exclude those cases in which the sensor node is active after experiencing a collision at time $t = t_m$. Thus, we must subtract $\mathbb{P}\{A_p \mid c\}$, calculated through Equation (3.18), to $\sum_{t=f_m}^{S_{max}} \mathbb{P}\{CCA^t \mid c\}$

□

Below, we denote by $\overline{N} = [N_p,\ N_{np},\ N_d]$ a composition of sensor nodes, where **(i)** N_p indicates the number of nodes that are still *active* at f_m and *have* participated to event e_m, **(ii)** N_{np} is the number of nodes that are still *active* at f_m but *have not* participated to e_m, and **(iii)** N_d is the number of sensor nodes that have *dropped* their packet and, hence, are no more active at f_m. By definition, $\forall \overline{N},\ \forall c,\ N_p + N_{np} + N_d = N_r$. We now

compute the probability of $\overline{N}$, $\mathbb{P}\{\overline{N} \mid c\}$, both when e_m is a success and when it is a failure.

If e_m is a success $T_m = S$, then $\mathbb{P}\{A_p \mid c\} = 0$. This is because it is not possible for a node to be still active after experiencing a success. Hence, $N_p = 0$, $\forall \overline{N}$ and the probability $\mathbb{P}\{\overline{N} \mid c\}$ of a composition $\overline{N}$ is equal to:

$$\mathbb{P}\{\overline{N} \mid c\} = \binom{N_r}{N_{np}} \cdot \mathbb{P}\{A_{np} \mid c\}^{N_{np}} \cdot \mathbb{P}\{D\}^{N_d} \tag{3.20}$$

In Equation 3.20 $\mathbb{P}\{D\} = \mathbb{P}\{F_{CCA} \mid c\} + \mathbb{P}\{F_{Rtx} \mid c\}$ is the probability that a sensor node has dropped its data packet due to either exceeded number of backoff stages or exceeded number of retransmissions, before f_m. In addition, the second and third terms provide the probability that exactly N_{np} nodes are still active in the network, and the probability that N_d nodes are no more active, respectively. Obviously, all possible combinations are taken into consideration.

The calculation of $\overline{N}$ for the case $T_m = F$ follows the same line of reasoning and is shown in the Appendix.

Claim 10. The probability $\mathbb{P}\{no\_txs \mid c\}$ that no other events occur in the network after e_m is:

$$\mathbb{P}\{no\_txs \mid c\} = \mathbb{P}\{\overline{N} = [0,\ 0,\ N_r] \mid c\} \tag{3.21}$$

Proof. The probability $\mathbb{P}\{no\_txs \mid c\}$ that no events occur after e_m, is equal to the probability that all nodes have finished their CSMA/CA execution before f_m, i.e. $\mathbb{P}\{\overline{N} = [0,\ 0,\ N_r] \mid c\}$. □

Derivation of new events after e_m

When $\mathbb{P}\{no\_txs \mid c\} \neq 1$, it means that there are cases in which at least one sensor node is still active at time $t = f_m$ and, hence, other events may occur in the network. Below, we consider all the events that may occur in the network after e_m and, for each of them, we calculate the corresponding probability. To this end, we first derive the probability for an active sensor node to perform a CCA at a given time $t \in [f_m,\ S_{max}]$. We need to discriminate between active nodes that *have* participated to event e_m, and active nodes that *have not* participated.

In the former case, after participating to e_m, sensor nodes are in one of the states $B_{1j}, 2 \leq j \leq T_{max}$, i.e. the first backoff stage of a retransmission attempt. Hence, according to CSMA/CA, they will perform a CCA after waiting for both the retransmission timeout D_{to} and a random number $w \in [0,\ W_1 - 1]$ of backoff periods. Since w is uniformly distributed in $[0,\ W_1 - 1]$, the probability $\mathbb{P}\{CCA_p^t \mid c\}$ that any of the considered sensor nodes will perform a CCA at time $t \in [f_m,\ S_{max}]$ is equal to $\frac{1}{W_1}$ for $t \in [f_m + 2D_{bp},\ f_m + 2D_{bp} + (W_1 - 1)D_{bp}]$, and zero otherwise (see also figure 3.7(b)).

In the second case we consider sensor nodes that have not participated to event e_m. First, we need to correct the computation of $\mathbb{P}\{CCA_{1j}^t \mid c\},\ 2 \leq j \leq T_{max}$, for time instants $t \in [f_m + 2 \cdot D_{bp},\ f_m + 2 \cdot D_{bp} + (W_1 - 1) \cdot D_{bp}]$, shown in equation 3.14 as follows.

$$\mathbb{P}\{CCA_{1j}^t \mid c\} = \mathbb{P}\{CCA_{1j}^t \mid c\} - \sum_{i=1}^{B_{max}} \mathbb{P}\{CCA_{ij-1}^{t_m} \mid c\} \cdot \frac{1}{W_1} \tag{3.22}$$

This is to exclude those cases that lead the sensor node to perform a CCA at time t after performing a CCA at time t_m. Therefore, the probability $\mathbb{P}\{CCA_{np}^t \mid c\}$, for time instants $t \in [f_m, S_{max}]$, can be calculated as:

$$\mathbb{P}\{CCA_{np}^t \mid c\} = \frac{\mathbb{P}\{CCA^t \mid c\}}{\mathbb{P}\{A_{np}\}} \tag{3.23}$$

Equation 3.23 can be explained as follows. Since we are considering an active sensor node, it will surely perform a CCA at a time $t \geq f_m$. Hence, for such a node, the probability to perform a CCA at a specific time $t \in [f_m,\ S_{max}]$ can be calculated by normalizing $\mathbb{P}\{CCA^t \mid c\}$ with probability $\mathbb{P}\{A_{np}\}$, i.e., the probability that the node will perform a CCA at any instant $t \geq f_m$.

Now, we derive both $\mathbb{P}\{e_{s_i} \mid \overline{N}\}$ and $\mathbb{P}\{e_{f_i} \mid \overline{N}\}$, $i \in \{0,\ 1,\ ...,\ M_w\}$ where $\mathbb{P}\{e_{s_i} \mid \overline{N}\}$ ($\mathbb{P}\{e_{f_i} \mid \overline{N}\}$) denotes the probability that the first event that occurs after e_m, given the composition of nodes $\overline{N}$, is a success (failure) occuring at time $t_i = f_m + i \cdot D_{bp}$. Claim 11 holds.

Claim 11.

$$\begin{aligned}\mathbb{P}\{e_{s_i} \mid \overline{N}\} = N_{np} \cdot \mathbb{P}\{CCA_{np}^{f_m+i\cdot D_{bp}} \mid c\} \\ \cdot \left(\sum_{j=i+1}^{M_w} \mathbb{P}\{CCA_{np}^{f_m+j\cdot D_{bp}} \mid c\}\right)^{N_{np}-1} \\ \cdot \left(\sum_{j=i+1}^{M_w} \mathbb{P}\{CCA_{p}^{f_m+j\cdot D_{bp}} \mid c\}\right)^{N_{p}} \\ + N_p \cdot \mathbb{P}\{CCA_{p}^{f_m+i\cdot D_{bp}} \mid c\} \cdot \\ \cdot \left(\sum_{j=i+1}^{M_w} \mathbb{P}\{CCA_{p}^{f_m+j\cdot D_{bp}} \mid c\}\right)^{N_{p}-1} \\ \cdot \left(\sum_{j=i+1}^{M_w} \mathbb{P}\{CCA_{np}^{f_m+j\cdot D_{bp}} \mid c\}\right)^{N_{np}}\end{aligned}$$

Proof. In order to have a successful transmission at time $t = f_m + i \cdot D_{bp}$ only one sensor node, among the $N_{np} + N_p$ active nodes, must start the CCA at time t, while all the other nodes have to perform a CCA after time $f_m + i \cdot D_{bp}$. We need to consider two different cases, depending on whether the sensor node that successfully transmits its data packet has, or has not, participated to e_m. The equation provides $\mathbb{P}\{e_{s_i} \mid \overline{N}\}$ as the sum of two components, referring to the first and second case, respectively. Let us examine the first term of the equation. $\mathbb{P}\{CCA_{np}^{f_m+i\cdot D_{bp}} \mid c\}$ indicates the probability, for a sensor node that has not participated to event e_m, to perform a CCA operation at time $t = f_m + i \cdot D_{bp}$ while $\left(\sum_{j=i+1}^{M_w} \mathbb{P}\{CCA_{np}^{f_m+j\cdot D_{bp}} \mid c\}\right)^{N_{np}-1}$ is the probability that all the other $N_{np} - 1$ sensor nodes perform their CCA operation at a time instant $t > f_m + i \cdot D_{bp}$. Naturally, all possible N_{np} combinations are taken into consideration. Finally, the term $\left(\sum_{j=i+1}^{M_w} \mathbb{P}\{CCA_{p}^{f_m+j\cdot D_{bp}} \mid c\}\right)^{N_p}$ indicates the probability that all the N_p sensor nodes that have participated to event e_m perform a CCA operation at a time $t > f_m + i \cdot D_{bp}$. The second component of the equation has the same structure of the first one. Hence, we omit its explanation. □

Let us focus now on the probability $\mathbb{P}\{e_{f_i} \mid \overline{N}\}$. We need to point out that a failure can occur only if $N_{np} + N_p \geq 2$. Hence, $\mathbb{P}\{e_{f_i} \mid \overline{N}\} = 0$, $\forall i \in [0,\ M_w]$ when $N_p + N_{np} < 2$. Below, we will focus on cases where $N_p + N_{np} \geq 2$. We denote by $comp(m)$ the set of

possible compositions $(m_1,\ m_2)$ of an integer m in two distinct parts m_1 and m_2 with $m_1 + m_2 = m$.

Claim 12.

$$\mathbb{P}\{e_{f_i} \mid \overline{N}\} = \sum_{m=2}^{N_p+N_{np}} \sum_{(m_1,\ m_2)\in comp(m):m_1\leq N_p \wedge m_2 \leq N_{np}} \binom{N_p}{m_1} \cdot \mathbb{P}\{CCA_p^{f_m+i\cdot D_{bp}} \mid c\}^{m_1} \left(\sum_{j=i+1}^{M_w} \mathbb{P}\{CCA_p^{f_m+j\cdot D_{bp}} \mid c\} \right)^{N_p-m_1} \cdot \binom{N_{np}}{m_2} \mathbb{P}\{CCA_{np}^{f_m+i\cdot D_{bp}} \mid c\}^{m_2} \left(\sum_{j=i+1}^{M_w} \mathbb{P}\{CCA_{np}^{f_m+j\cdot D_{bp}} \mid c\} \right)^{N_{np}-m_2}$$

Proof. In the equation, the outer sum accounts for all possible numbers m, $2 \leq m \leq N_p + N_{np}$, of colliding sensor nodes, while the inner sum considers all the possible compositions $(m_1,\ m_2) \in comp(m)$ of colliding nodes. Specifically, m_1 denotes the number of colliding nodes that have taken part to event e_m while m_2 is the number of colliding nodes that have not. Inside the inner sum, the product between two terms is performed. The first one indicates the probability that exactly m_1 sensor nodes, out of N_p, perform a CCA at time $t = f_m + i \cdot D_{bp}$, while the remaining ones $(N_p - m_1)$ perform a CCA at a time $t > f_m + i \cdot D_{bp}$. The second term is derived in a similar way.

□

Finally, using the law of total probability, we derive both $\mathbb{P}\{e_{s_i} \mid c\}$ and $\mathbb{P}\{e_{f_i} \mid c\}$, $i \in [0,\ M_w]$ as follows.

$$\mathbb{P}\{e_{s_i} \mid c\} = \sum_{\overline{N}} \mathbb{P}\{\overline{N} \mid c\} \cdot \mathbb{P}\{e_{s_i} \mid \overline{N}\} \tag{3.24}$$

$$\mathbb{P}\{e_{f_i} \mid c\} = \sum_{\overline{N}} \mathbb{P}\{\overline{N} \mid c\} \cdot \mathbb{P}\{e_{f_i} \mid \overline{N}\} \tag{3.25}$$

3.4.4 Parallelization of ECC algorithm

As shown in Figure 3.4, ECC continuously performs the following steps:

1. extracts a chain $c:\ s_c = \{e_1,\ e_2,\ ...,\ e_m\}$ from L_c;
2. generates all the events e_x that can occur after e_m.
3. $\forall e_x$, adds $c_x:\ s_{c_x} = \{e_1,\ e_2,\ ...,\ e_m,\ e_x\}$ to L_c.

During the execution of the three steps above only information associated with chain c is used by the algorithm. Hence, it is possible to speed-up the execution by allowing more threads to manage different chains in parallel. In section 3.5 we show, through experimental measurements, that the execution time of the ECC algorithm decreases almost *linearly* with the number of used threads. This drastically reduces the computation time needed to generate all the possible outcomes and makes the analysis of large networks computationally feasible.

3.4.5 Derivation of performance metrics

When L_c becomes empty, F_c contains all chains c representing possible outcomes of the CSMA/CA execution with a probability to occur greater than, or equal to, θ. Now, by using chains $c \in F_c$, we derive the following metrics:

- *Coverage* (C): portion of the event space covered by chains in set F_c; it characterizes the accuracy of the analysis.
- *Packet delivery ratio* (R): fraction of data packet successfully transmitted to the coordinator node; it measures the reliability provided by CSMA/CA.
- *Packet latency* (L): delay experienced by a sensor node to successfully transmit a data packet to the coordinator node; it indicates the timeliness allowed by CSMA/CA in reporting an event.
- *Energy consumption* (E): average total energy consumed by all sensor nodes in the network to report an event to the coordinator node; it measures the energy efficiency of CSMA/CA.

Let $c_i : \ s_{c_i} = \{e_1, \ e_2, \ ..., \ e_m\}$ be a specific chain in set F_c, and p_{c_i} its associated probability. Then, the coverage C can be calculated as follows:

$$C = \sum_{c_i \in F_c} p_{c_i} \tag{3.26}$$

To derive the delivery ratio R we compute, for each chain c_i, the fraction of successful transmissions R_i occurred in c_i. Since there are N nodes in the network and each node transmits one data packet, R_i can be easily calculated as $R_i = {N_s}^i/N$ where ${N_s}^i$ is the number of successful transmissions in c_i. Since each chain c_i has probability p_{c_i} to occur, the delivery ratio R is given by

$$R = \sum_{c_i \in F_c} \frac{p_{c_i}}{C} \cdot R_i \tag{3.27}$$

Similarly, to calculate the average latency, we first derive the average latency L_i experienced by data packets successfully transmitted in chain c_i, for any c_i containing at least one successful transmission (i.e. $c_i : \exists e_j \in s_{c_i} \wedge T_j = S$). L_i can be expressed as follows.

$$L_i = \frac{\sum_{e_j \in s_{c_i} : T_j = S} f_j}{N_s^i} \tag{3.28}$$

In Equation (3.28), the sum of latencies f_j for all packets successfully transmitted in chain c_i is first calculated. Then, the obtained sum is divided by the number $N_s^i = |\{e_j \in s_{c_i} : \ T_j = S\}|)$ of successful transmissions in c_i. Therefore, the average latency L is derived as

$$L = \sum_{c_i \in F_c : \ N_s^i > 0} \frac{p_{c_i}}{\sum_{c_i \in F_c : \ N_s^i > 0} p_{c_i}} \cdot L_i \tag{3.29}$$

We now show the computation of the probability density function (PDF) of packet latency. The probabilty $P(t)$ to receive one data packet with a delay equal to t can be calculated as follows:

$$P(t) = \sum_{c_i \in F_c : \exists e_j \in s_{c_i} | T_j = S \wedge f_j = t} p_{c_i} \tag{3.30}$$

Equation 3.30 calculates $P(t)$ by performing the sum of the probabilities of all chains c_i containing a successful transmission at time t.

Finally, we calculate the average total energy consumption E. Let en_{c_i} denote the total energy consumed by all sensor nodes to manage the events occurred in a specific chain c_i. Hence,

$$E = \sum_{c_i \in F_c} \frac{p_{c_i}}{C} \cdot en_{c_i} \tag{3.31}$$

For the derivation of en_{c_i} see Appendix.

3.5 Results

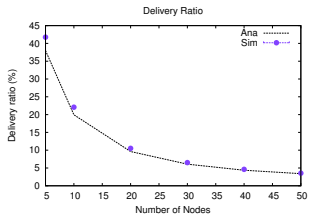


FIGURE 3.8: Delivery Ratio.

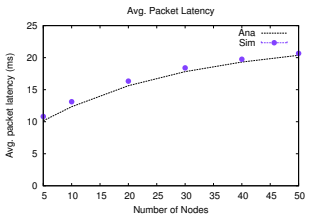


FIGURE 3.9: Average Packet Latency.

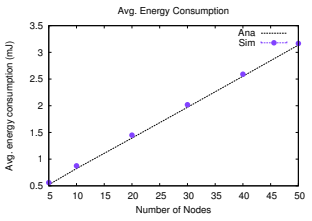


FIGURE 3.10: Average Energy Consumption.

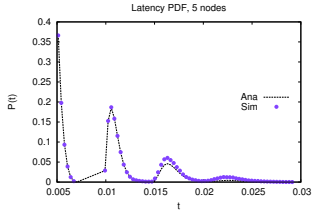


FIGURE 3.11: Latency PDF, 5 nodes.

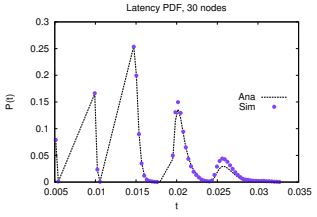


FIGURE 3.12: Latency PDF, 30 nodes.

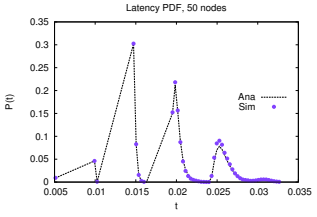


FIGURE 3.13: Latency PDF, 50 nodes.

In this section we evaluate the accuracy and tractability of our analytical model. We also use the model to investigate the performance of CSMA/CA in the considered event-driven scenario.

3.5.1 Model validation

As a preliminary step, we validate our analytical model through simulation experiments. To this end, we use the ns2 simulation tool [54], which includes the 802.15.4

module. In our simulations we consider a star network topology with sensor nodes located in a circle of radius $10m$ centered at the coordinator node. The transmission range is set to $15m$, while the carrier sensing range is set to $30m$. In all the experiments we assume that the 802.15.4 MAC protocol is operating in NBE mode with a 250 Kbps bit rate. Power consumption values are derived from the Chipcon CC2420 radio transceiver [55]. Unless stated otherwise, we use the following set of CSMA/CA parameter values: $macMinBE = 3$, $macMaxBE = 4$, $macMaxBackoffs = 2$, $macMaxFrameRetries = 1$.

For each simulation experiment, we perform 10 independent replications, each of which consists of 10000 cycles. In each cycle, it is assumed that all sensor nodes have to transmit one packet. We derive confidence intervals by using the independent replications method and a 95% confidence level. However, they are typically very small and cannot be appreciated in the plots below.

Figures 3.8-3.10 compare analytical and simulation results, obtained in the same operating conditions, in terms of delivery ratio, average latency and average energy consumption, respectively, for different network sizes (i.e., number of sensor nodes). In addition, Figures 3.11-3.13 compare the probability density function (PDF) of packet latency for three different network sizes (i.e, with 5, 30, and 50 sensor nodes). The analytical results have been obtained using $\theta = 0$ in the ECC algorithm. As a general remark, we observe that analytical and simulation results are almost overlapped, in all the considered scenarios. We also investigated additional scenarios, with different CSMA/CA parameter values, and observed a similar agreement. Some of these results are presented in section 3.5.3, where we investigate the impact of CSMA/CA parameters on the netwok performance.

Figure 3.8 shows that the delivery ratio drastically reduces with the number of sensor nodes and it is very low even with a limited number of sensor nodes. This is mainly due to the 802.15.4 CSMA/CA algorithm that is not able to manage contentions efficiently even when the number of contending nodes is relatively low, due to its random nature. In the considered $event - driven$ scenario this limitation is more apparent as *all* sensor nodes in the network start reporting data *simultaneously*, upon detecting an event.

Figures 3.9 and 3.10 show that both the average latency and energy consumption increase with the network size, as expected. This is because when the number of sensor

nodes contending for channel access increases, the collision probability increases as well. Hence, sensor nodes consume more energy. In addition, they take more time to transmit their packets. Specifically, successful transmissions tend to occur some time after the beginning, when the level of contention is lower because some nodes have already given up, due to exceeded number of backoff stages or retransmissions (see Section 3.1). This behavior is confirmed by the results in Figures 3.11-3.13, where the highest spikes in the PDF gradually shift to the right side of the graphs as the number of sensor nodes increases.

3.5.2 Impact of θ and multithreading

TABLE 3.1: Analytical results with different values of θ.

N	$\theta = 0$					
	C	Chains	R (%)	L (ms)	E (mJ)	Time (s)
10	1	1842355	19.94	12.33	0.822	14407
30	1	1842355	6.07	17.81	1.970	24768
50	1	1842355	3.40	20.36	3.130	33264
N	$\theta = 10^{-7}$					
	C	Chains	R (%)	L (ms)	E (mJ)	Time (s)
10	0.999	27590	19.94	12.32	0.822	398
30	0.999	19536	6.07	17.81	1.970	922
50	0.999	11509	3.40	20.36	3.130	875
N	$\theta = 10^{-5}$					
	C	Chains	R (%)	L (ms)	E (mJ)	Time (s)
10	0.970	3814	19.88	12.17	0.811	93
30	0.978	3224	6.04	17.75	1.966	267
50	0.987	2253	3.39	20.34	3.136	316

In order to evaluate the trade-off between *tractability* and *accuracy* of our model, Table 3.1 summarizes the analytical results obtained with different values of θ (the network parameter values are the same as in Section 3.5.1). Specifically, Table 3.1 shows the number of chains generated by the ECC algorithm, the coverage of the event space (C), the performance metrics defined in Section 6.5 (i.e., R, L and E) and, finally, the computation time (in seconds) required to compute the same performance metrics. When using $\theta = 0$, a coverage of 100% and, hence, the maximum accuracy of results is obtained. However, the number of generated chains is very large (more than 1800000), which requires a high computation time (33264s in the case of 50 sensor nodes, i.e. more than 9 hours).

As expected, the coverage of the event space decreases as the value of θ increases. With $\theta = 10^{-5}$, it reduces to $97 - 98\%$, however the model still obtains nearly the same

accuracy as with $\theta = 0$. Moreover, the computation time reduces by a factor of more than 100, and the number of generated chains reduces by a factor of more than 500, with respect to $\theta = 0$, when $N = 50$. These results can be explained as follows. As mentioned in Section 3.4, ECC analyzes only events that are most likely to occur in the network, which are the events that will influence most the performance metrics. This way, ECC achieves approximately the same accuracy as when $\theta = 0$ while analyzing a much lower number of events. Therefore, it saves a tremendous amount of computation time.

Surprisingly, Table 3.1 shows that, when $\theta > 0$, the number of chains generated by ECC *decreases* as the network size increases. In addition, the coverage *increases* although the number of chains decreases. This apparently counterintuitive behavior can be explained as follows. When the number of sensor nodes increases, some events become significantly more likely to occur than others. For instance, if $N = 50$, collisions are more likely to occur than successful transmissions. Therefore, when $N = 50$, ECC will select, at each step, fewer (yet very likely) events with respect to the case $N = 5$. Hence, the number of considered chains decreases (and the coverage increases), as the number of nodes increases. This property of ECC dramatically reduces the computation effort needed to calculate the metrics of interest when the network size increases, thus making the analysis of large networks easier.

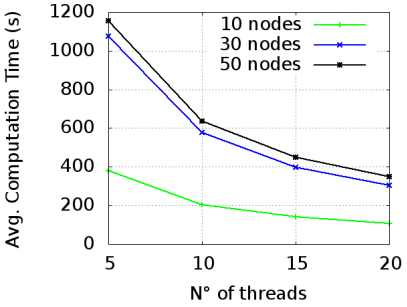


FIGURE 3.14: Avg. computation time vs. number of threads.

Finally, to further assess the model tractability, we measured the average computation time of the analytical model as a function of the number of threads that are activated. Figure 3.14 shows that the computation time decreases almost *linearly* with the number of threads. This is because the algorithm is implemented in such a way to assign the computation of the various chains to different threads. Since threads synchronize only

to modify a global variable (i.e. the list of chains L_c), each thread is almost independent from the others. Hence, the almost linear decrease of computation time.

3.5.3 Impact of CSMA/CA parameters

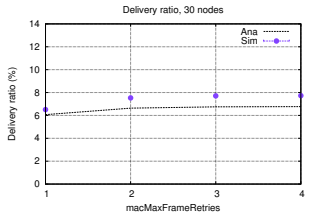


FIGURE 3.15: Delivery ratio vs. $macMaxFrameRetries$

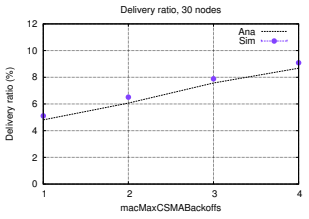


FIGURE 3.16: Delivery ratio vs. $macMaxCSMABackoffs$

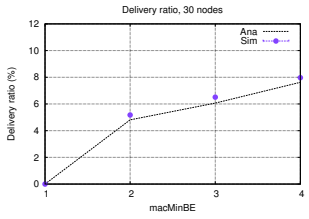


FIGURE 3.17: Delivery ratio vs. $macMinBE$

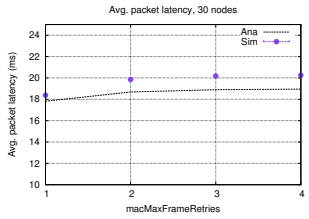


FIGURE 3.18: Avg. latency vs. $macMaxFrameRetries$

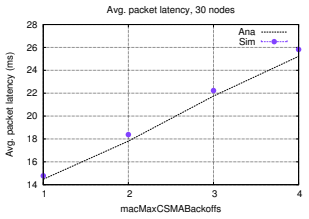


FIGURE 3.19: Avg. latency vs. $macMaxCSMABackoffs$

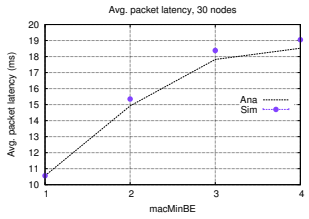


FIGURE 3.20: Avg. latency vs. $macMinBE$

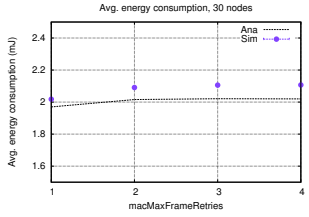


FIGURE 3.21: Avg. energy consumption vs. $macMaxFrameRetries$

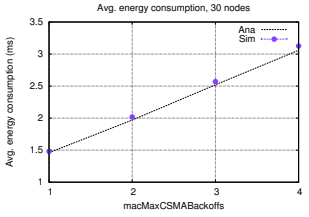


FIGURE 3.22: Avg. energy consumption vs. $macMaxCSMABackoffs$

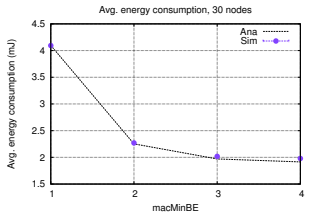


FIGURE 3.23: Avg. energy consumption vs. $macMinBE$

In this section we evaluate the impact of each single CSMA/CA parameter on the overall performance. To this end, we focus on a network with 30 sensor nodes and show the impact of parameters on the delivery ratio, average packet latency and energy consumption of the network.

Figure 3.15 shows that increasing the maximum allowed number of retransmissions (i.e. $macMaxFrameRetries$) does not provide any significant effect on the delivery ratio, for values larger than one. This is because the majority of packets are dropped by the

MAC protocol because of exceeded number of backof stages (i.e., consecutive CCAs). Indeed, increasing the maximum number of backoff stages (i.e. the $macMaxBackoffs$) results in an increase of the delivery ratio, as shown in Figure 3.16. Furthermore, we observe an increase in both the average latency and energy consumption (Figure 3.19 and 3.22). This is because a larger number of packets is successfully transmitted, which takes more time and more energy. Finally, Figure 3.17 shows that increasing the minimum backoff window size (i.e. $macMinBE$) also causes an increase of the delivery ratio. The motivation is that a larger initial backoff window size reduces the collision probability at the first backoff stage, thus increasing the probability of successful transmission. Again, we observe an increase in the average latency (Figure 3.20). Instead, the energy consumption decreases since less collisions occur (Figure 3.23).

The above results shows that, increasing the CSMA/CA parameter values is surely beneficial in terms of increased communication reliability. However, it also increases packet latency and/or the energy consumption, which may not be good for time-critical applications. Hence, the most appropriate parameter setting depends on the specific application scenario.

3.6 Conclusions

In this chapter we have presented an analytical model of the unslotted CSMA/CA algorithm used in 802.15.4 WSNs operating in NBE mode. The proposed model is both accurate and efficient. It leverages an approach called *Event Chains Computation* (ECC), that reduces complexity, with a limited loss of accuracy, by removing event sequences whose probability to occur is below a given threshold. We have shown that the computation time required for deriving the performance metrics of interest can be reduced by a factor of more than 100, and even more, with a negligible impact on the accuracy of the obtained results. Also, our model can exploit a multi-threading approach, thus taking advantage of a parallel execution. Since we have introduced no oversimplifying assumptions, our model allows an accurate analysis of 802.15.4 WSNs in NBE mode, even when there is a large number of sensor nodes. Our results highlight that the unslotted CSMA/CA algorithm has severe limitations in terms of reliability and scalability. Specifically, the delivery ratio provided by the algorithm is very low even when the number of contending nodes is not so high. In addition, it sharply

decreases when the number of contending nodes increases. The reliability provided by the algorithm can be significantly improved by increasing the initial backoff-window size and/or the maximum number of backoff stages allowed for each packet. However, this also increases packet latency and/or energy consumption. As we reported in section 2.4.2, similar considerations also apply to the slotted 802.15.4 CSMA/CA algorithm used in BE mode and a number of approaches [22, 29, 30] (described in section 2.4.2.1) have been proposed in the literature to provide the *optimal* setting of CSMA/CA parameter values, i.e. the set of values for CSMA/CA parameters through which it is possible to satisfy the application requirements (in terms of reliability) with the minimum energy consumption for sensor nodes. In the next chapter a comparison of the performance of the different approaches is reported.

Chapter 4

Comparison of strategies for 802.15.4 CSMA/CA algorithm parameters setting

4.1 Introduction

In section 2.4.2.1 we described the different approaches (i.e. *model-based offline computation* [22], *model-based adaptation* [29], and *measurement-based adaptation* [30]), that have been proposed in the literature for the tuning of 802.15.4 CSMA/CA algorithm parameters. In order to better understand pros and cons of the various approaches, and help network designers in choosing the most effective solution, in this chapter we compare their performance. We investigate, through simulation, their capability to satisfy application requirements, in terms of reliability. However, we also consider their energy efficiency and ability to adapt to time-varying operating conditions. Our simulation results highlight the effectiveness and flexibility of the *ADaptive Access Parameter Tuning (ADAPT)* algorithm proposed in [30]. Motivated by these results, we implemented *ADAPT* in a real sensor network and carried out an experimental analysis to confirm its suitability to operate in a real environment. The experimental results confirmed the simulation results.

The rest of this chapter is organized as follows. Next section compares the considered algorithms both in stationary and dynamic scenarios. Section 4.3 reports an experimental evaluation of the performance of ADAPT. Finally, conclusions are drawn in section 4.4.

4.2 Performance Comparison

To compare the performance of the three algorithms described in section 2.4.2.1 we used the ns2 simulation tool [54], which includes an 802.15.4 module. To implement the analytical model leveraged by the offline algorithm we also used MATLAB [56]. In our analysis we considered a star network scenario where the sink node acts as the network coordinator, and sensor nodes are placed in a circle centered at the sink node, $10m$ far from it. The transmission range was set to $15m$ while the carrier sensing range was set to $30m$. We considered a periodic reporting application where sensed data have to be reported to the sink periodically, i.e., at each Beacon Interval.

To evaluate the performance of the three considered algorithms we derived the following performance indexes.

1. *Packet delivery ratio*, defined as the ratio between the number of packets correctly received by the sink, and the total number of packets generated by all sensor nodes. It measures the *long-term reliability*.

2. *Miss ratio*, defined as the fraction of times the packet delivery probability – calculated over the current Beacon Interval – drops *below* the threshold required by the application. It measures the inability to provide *short-term reliability* (ideally should be zero).

3. *Average energy per packet*, defined as the total energy consumed by each sensor node divided by the total number of generated packets. It measures the *energy efficiency*.

4. *Average latency*, defined as the average time from when the packet transmission starts at the source node to when the packet is correctly received by the sink. It characterizes the *timeliness*.

5. *Convergence time,* defined as the time when the packet delivery probability – calculated over the current Beacon Interval – reaches the threshold required by the application for the first time (from the network startup time). It measures the *ability to adapt* to changing conditions.

The energy consumed by a sensor node was calculated using the model presented in [57], which is based on the Chipcon CC2420 radio transceiver [55]. The model assumes four radio states, namely *transmit*, *receive*, *idle* and *sleep*. In addition, the model also accounts for the energy spent during state transitions.

In our analysis we assumed that the application is critical in terms of reliability and, hence, it requires a packet delivery ratio >= 80%. In addition, to avoid concentrated packet losses, the application also requires a miss ratio lower than 10%. This means that the application can temporarily tolerate a delivery probability lower than 80%. However, this should occur in a limited number of cases (less than 10% of the beacon intervals). Hence, the optimal CSMA/CA parameter setting must guarantee the requested reliability (in terms of delivery ratio and miss ratio), while minimizing the average energy per packet. We need to point out that the considered thresholds for delivery ratio and miss ratio (i.e., 80% and 10%) are somewhat arbitrary as, in practice, they depend on the specific application. However, we performed other experiments with different thresholds and we obtained results similar to those presented below.

Table 4.1 summarizes the 802.15.4 MAC protocol parameters used for all the algorithms, while Table 4.2 shows specific parameters of individual algorithms. For ADAPT, the considered values of d, σ, and γ are taken from [30]. For the model-based adaptive algorithm, a window size m=16 was used in [29] to estimate α, β, and τ. We used a lower value (i.e., m=4), as it still provides accurate estimates and reduces the convergence time of the algorithm (see below). As anticipated, we considered a periodic reporting application, which is typical for beacon-enabled sensor networks. We also assumed that each sensor node generates 10 data packets at every Beacon Interval. This means that all sensor nodes have a packet to transmit for some time. Hence, the operating conditions are very close to saturated traffic conditions. Accordingly, we set q_0=0 in the model used by both model-based algorithms.

In our experiments, for each simulated scenario, we performed 10 independent replications, where each replication consists of 1000 Beacon Periods. For each replication we

TABLE 4.1: 802.15.4 MAC Protocol parameters

Parameter	**Value**
Bit Rate	250 Kbps
Data Frame (Payload) Size	109 (100) bytes
ACK Frame Size	11 bytes
Beacon Order (BO), Superframe Order (SO)	11, 8
$macMinBE^{min}$, $macMinBE^{max}$	1, 7
$macMaxCSMABackoffs^{min}$, $macMaxCSMABackoffs^{max}$	1, 10
$macMaxFrameRetries^{min}$, $macMaxFrameRetries^{max}$	0, 9
Power Consumption in RX mode (P_{rx})	35.46 mW
Power Consumption in TX mode (P_{tx})	31.32 mW
Power Consumption in Idle mode (P_{idle})	0.77 mW
Power Consumption in Sleep mode (P_{sleep})	0.036 μW

TABLE 4.2: Specific parameters

Parameter	**Value**
d(ADAPT)	0.6
σ (ADAPT)	0.03
γ (ADAPT)	0.03
m (Model-based Adaptation)	4

discarded the initial transient interval (10% of the overall duration) during which nodes associate to the coordinator node and start generating packets. The results shown below are averaged over all the different replications. We also derived confidence intervals by using the independent replication method.

4.2.1 Analysis in stationary conditions

In this section we analyze the performance of the three algorithms in stationary conditions. Specifically, we assume that, for each experiment, the number of sensor nodes is fixed. Figure 4.1(a) and Figure 4.1(b) show the packet delivery ratio and miss ratio of the considered algorithms, for an increasing number of sensor nodes. We can

see that all the three algorithms are able to fulfill the application requirements. The two adaptive algorithms have similar performance in terms of delivery ratio, but the measurements-based algorithm typically exhibits a larger miss ratio. This is because it tends to oscillate between adjacent parameter sets, while the model-based adaptive algorithm is more stable. We can also observe that both the adaptive algorithms provide a delivery ratio slightly higher than the threshold. Instead, the delivery ratio provided by the offline algorithm is significantly above the threshold. This is because the adaptive algorithms can switch between different parameter sets over time, while the offline algorithm always uses the same set during the whole experiment. This also reflects into an high energy consumption and packet latency for the offline algorithm.

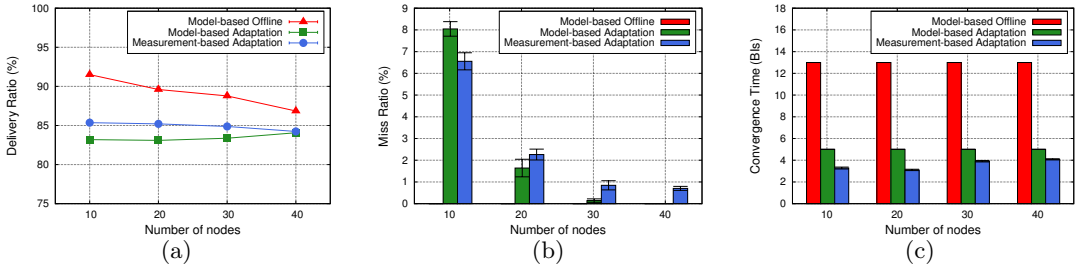


FIGURE 4.1: Delivery ratio (a), Miss ratio (b) and Convergence time (c) for the different algorithms

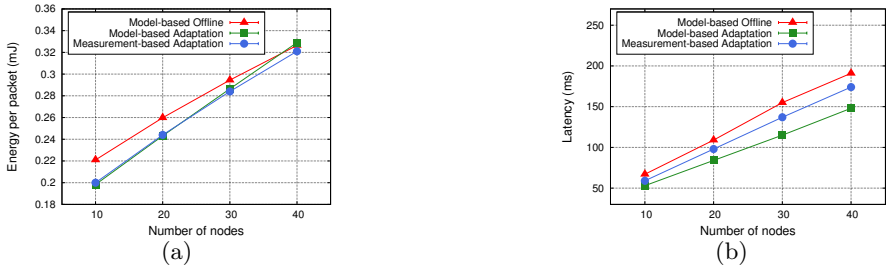


FIGURE 4.2: Average Energy per Packet (a), and Average Latency (b) for the different algorithms.

In terms of energy efficiency 4.2(a) the measurement-based algorithm has the best performance, while the model-based adaptive algorithm is more energy consuming. This is mainly because the latter algorithm tends to increase the delivery probability by increasing the number of retransmissions (*macMaxFrameRetries*), while the measurement-based algorithm achieves the same result by increasing the number of backoff trials (*macMaxCSMABackoffs*), which is less expensive in terms of energy consumption (as highlighted in [30]). On the other side, the strategy used by the model-based adaptive algorithm results in a lower average latency experienced by packets, as highlighted by

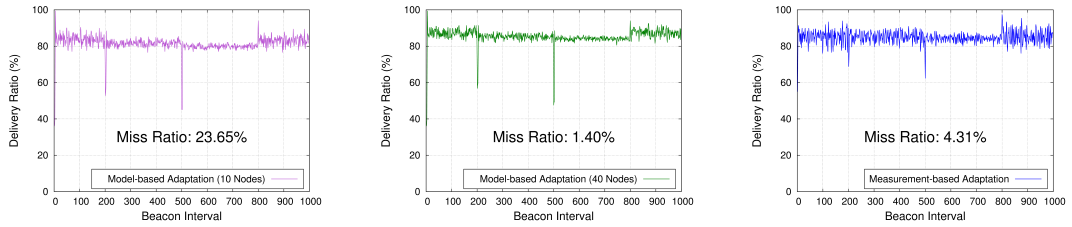
Figure 4.2(b). This is because, when a packet is re-transmitted the contention window size is re-initialized to its minimum value. Instead, when a new backoff trial is started, the contention window size is doubled (unless it has reached its maximum value).

Finally, Figure 4.1(c) shows the *convergence time* of the three algorithms. For the offline algorithm we assume as convergence time the time needed to solve the analytical model at the coordinator/sink node, derive the optimal parameter setting, and communicate it to the sensor nodes. Obviously, this time strongly depends on the machine used to solve the model. We found that, with a quad-core PC, this time is approximately 400 s (i.e., around 7 minutes). Assuming a Beacon Interval of about 30 s (31.45s with BO=11), this results in a convergence time of 13 Beacon Intervals, as shown in Figure 4.1(c) (obviously it depends on the size of the Beacon Interval). The two adaptive algorithms have significantly lower convergence times. The model-based algorithms converges in about 5 Beacon Intervals as the optimal setting is derived using the samples measured in the previous m Beacon Intervals (and we used m=4). The measurement-based algorithm is the fastest one as, in the considered scenarios, it converges in 3-5 Beacon Intervals. Unlike the model-based algorithms, in ADAPT the convergence time is not constant and, generally, tends to increase with the number of sensor nodes. This is because it starts using the default parameter set and, then, increases the parameter values until an appropriate parameter configuration is reached. Obviously, the number of steps increases with the number of nodes as a higher number of contending nodes increases congestion and, thus, requires higher parameter values to guarantee the same reliability level.

4.2.2 Analysis in dynamic conditions

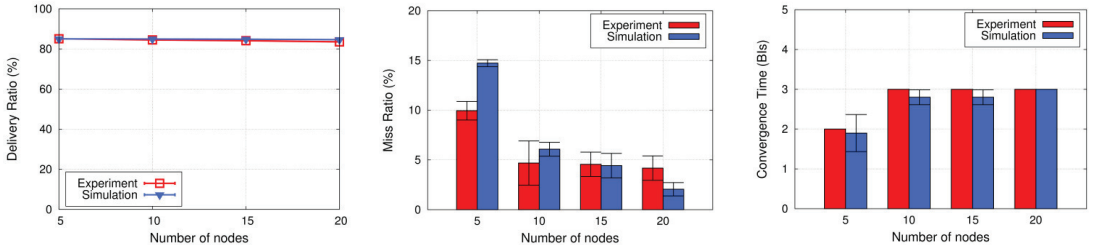
In this section we analyze the behavior of the three algorithms in dynamic conditions. To this end, we vary over time the number of active sensor nodes. We assume that, initially, there are 10 nodes. Then, this number increases to 20 at Beacon Interval 200, and to 40 at Beacon Interval 500. Finally, it reduces again to 10 at Beacon Interval 800. All the sensor nodes, when active, generate 10 packets per period and, hence, the offered load increases accordingly with the number of nodes. We restricted the following analysis to adaptive algorithms only, as the offline algorithm is not able to adapt itself to changing conditions. Also, we need to point out that the model-based adaptive algorithm requires the number of nodes as an input parameter. This may be problematic when the number

of active nodes cannot be predicted in advance. In our analysis we ran two different set of experiments for the model-based algorithm, with two different input values for the number of nodes. In the first case the algorithm uses the initial number of nodes (i.e., 10), while in the second case it uses the maximum number of nodes (i.e., 40). Figure 4.3(a) and Figure 4.3(b) show the delivery ratio, as a function of the Beacon Interval, experienced in the two cases (we show here a single run for each case, however we observed similar results in all the runs we performed). As expected, in the first case the algorithm exhibits very bad performance when the number of sensor nodes is higher than 10. The miss ratio is 15.6% and 53.3% with 20 and 40 nodes, respectively. In the second case the algorithm performs well in all the considered conditions and the observed miss ratio is always less than 2%.



(a) (b) (c)

FIGURE 4.3: Delivery ratio, as a function of time, provided by the model-based adaptation algorithm with input equal to 10 nodes (a) and 40 nodes (b). Delivery ratio, as a function of time, provided by the measurement-based adaptation algorithm (c).



(a) (b) (c)

FIGURE 4.4: Delivery ratio, Miss ratio, and Convergence Time for the measurement-based adaptation algorithm: Experimental vs. Simulation Results.

The measurement based algorithm does not require to know in advance the number of sensor nodes. The results for it are shown in Figure 4.3(c). Apart from spikes due to the sudden changes in the number of sensor nodes, the algorithm performs well in all the considered conditions. The miss ratio is always lower than 10% and, on average, equal to 4.3%.

4.2.3 Learned Lesson

A number of considerations follow from the previous analysis. First, the model-based offline algorithm is not able to adapt to changing operating conditions and, hence, it is not suitable for dynamic scenarios. Also, since the related analytical model makes some assumptions (e.g., on the packet generation process), it may not always provide the optimal setting in practical scenarios (e.g., periodic reporting applications). The latter consideration also applies to the model-based adaptive algorithm.

Both the adaptive algorithms perform similarly in terms of delivery ratio. The measurement based algorithm tends to oscillate more, but it takes a lower time to converge. The model-based algorithm provides a lower average latency, while the measurement-based algorithm is more energy efficient. This is an important point as sensor nodes are typically energy constrained. Finally, it should be emphasized that the model-based adaptive algorithm requires to know in advance the number of sensor nodes in the network and assumes that the wireless channel is ideal. This makes it unsuitable for practical scenarios where operating conditions (including the number of active nodes) may vary over time and transmission errors cannot be neglected. The measurement-based algorithm does not suffer from these limitations.

4.3 Experimental Analysis

Motivated by the previous remarks, we implemented the measurement-based adaptive algorithm (ADAPT) in real sensor nodes and performed an experimental analysis aimed at (i) showing that ADAPT is a viable solution for a real environment, and (ii) validating the previous simulation results for it. Our testbed consists of Tmote Sky sensor nodes [58] with TinyOS 2.x operating system [59]. Tmote Sky sensor nodes use the Chipcon CC2420 radio transceiver [55] that is compliant to the 802.15.4 physical layer. Instead of using the default MAC protocol shipped with the TinyOS software, we used TKN15.4 [60], an implementation of the 802.15.4 MAC protocol for TinyOS 2.x developed at TU Berlin, and implemented ADAPT on top of it. In our experimental analysis, we referred to the same star network scenario and parameters setting considered in the simulation analysis. We measured all the performance indexes considered in the simulation analysis but energy consumption. The results obtained are shown in Figure 4.4. It can be

observed that delivery ratio, miss ratio and convergence time obtained in simulations and experiments are quite similar. We also measured packet latency and we observed an experimental latency slightly higher than that measured in simulations. This is due to additional re-transmissions triggered by collisions caused by non-perfectly synchronized clocks and possible transmission errors (which are unavoidable in a real environment).

4.4 Conclusions

In this chapter we have compared three different algorithms for deriving the optimal settings of 802.15.4 sensor networks so as to guarantee the reliability requirements of the application with minimum energy consumption. The considered algorithms take different approaches, namely *offline computation*, *model-based adaptation* and *measurement-based adaptation*. We have found that both the adaptive algorithms perform well, however the model-based approach has some limitations that make it unsuitable for practical scenarios, where operating conditions typically vary over time and transmission errors cannot be neglected. Instead, the heuristic measurement-based algorithm does not suffer from such limitations. Hence, we have implemented it in a real sensor network and validated the simulation results for it through experimental measurements.

We want to highlight that the measurement-based algorithm still has some limitations. First, due to its nature, it typically tends to oscillate between two or more parameter sets and never stabilizes, thus consuming more energy than necessary. Second, it is memory-less and does not take advantage of previously derived optimal settings. If the same operating conditions repeat over time, the algorithm re-executes the adaptation procedure. This motivated us to design a new adaptive algorithm, called JIT-LEAP, that overcomes all these limitations. We present it in the next chapter.

Chapter 5

A Just-in-Time Adaptive Algorithm for Optimal Parameters Setting in 802.15.4 WSNs

5.1 Introduction

In the previous chapter we compared the performance of different solutions, namely *model-based* strategies [22] [29], and *measurement-based* strategies [30], that have been proposed in the literature to identify the optimal CSMA/CA setting in 802.15.4 WSNs. Our analysis highlighted that model-based approaches have a number of limitations. First, their effectiveness in providing the optimal setting depends on the accuracy of the underlying model. Typically, simplifying assumptions are introduced to make the model tractable. Furthermore, the model requires some input parameters, which may not be available in a real environment. For instance, the model used in [22] and [29] assumes ideal channel conditions and requires knowing in advance the number of network nodes. Conversely, measurement-based approaches, like ADAPT [30], do not suffer from these problems. However, they still have some limitations. ADAPT, due to its nature, typically tends to oscillate between two or more parameter sets and never stabilizes, thus consuming more energy than necessary. Furthermore, it is memory-less and does not

take advantage of previously derived optimal settings. If the same operating conditions repeat over time, the algorithm re-executes the adaptation procedure.

To overcome these limitations, in this chapter we propose a *Just-in-Time LEarning-based Adaptive Parameter tuning* (JIT-LEAP) algorithm. JIT-LEAP follows a measurement-based approach, does not make any assumption on the channel conditions, and does not require any a-priori information about the WSN (e.g., number of nodes). This makes it suitable for real-life scenarios. JIT-LEAP allows nodes to derive the optimal setting *autonomously*, i.e., only basing on local measurements. Furthermore, it avoids unnecessary energy wastes and, learning from the past history, is able to speed up the selection of the optimal setting. JIT-LEAP belongs to the class of *active* adaptive algorithms [61] since it relies on a trigger mechanism to activate the adaptation phase only when needed. Differently from *passive* algorithms (e.g., [29]) – where the adaptation mechanism is always on – active algorithms are generally prompter in adapting to new conditions, hence providing better performance and lower energy consumption. Formally, also ADAPT is an active algorithm as the adaptation mechanism is activated only when the locally-estimated packet delivery probability is below/over a predefined threshold. In practice, ADAPT tends to change the parameters setting (almost) at each step, thus behaving similarly to passive algorithms.

The detection/adaptation mechanism proper of active algorithms is inspired by recent studies on human brain that model brain mechanisms as a hierarchy of subsystems. Active adaptive algorithms inspired by this detection/adaptation mechanism have been successfully applied to classification problems [62], cognitive fault detection/diagnosis systems [63], and adaptive sampling in WSNs [64].

JIT-LEAP relies on a theoretically-grounded mechanism to detect changes in the operating conditions, based on a statistical Change Detection Test. This allows reducing significantly the number of false positive detections and identifying even small variations in the operating conditions. Hence, the parameters setting provided by JIT-LEAP is stable and accurate, resulting in low energy consumption. We show, by simulations, that JIT-LEAP outperforms all the previous algorithms meant to identify the optimal CSMA/CA setting in 802.15.4 WSNs.

The rest of this chapter is organized as follows. Next section formulates the problem addressed in this chapter in a formal way. Section 5.3 presents the JIT-LEAP algorithm.

Section 5.4 describes the simulation setup, while Section 5.5 presents the simulation results. Conclusions are drawn in Section 5.6.

5.2 Problem Formulation

In the following we refer to a WSN with a star topology, including a sink node (acting as the PAN coordinator) and a number of sensor nodes. We consider a periodic reporting application where data gathered by a sensor node are reported to the sink at each Beacon Interval. We assume that data/acknowledgement frames transmitted by nodes may be corrupted or lost. Despite that, the application requires a certain reliability level (expressed as percentage of data packets correctly delivered to the sink), that must be guaranteed with minimum energy consumption. To formulate the problem in a more formal way, we define the following indexes:

1. *Packet delivery ratio* (R_D): ratio between the number of data packets correctly delivered to the sink by a sensor node, and the total number of packets generated by that node. It measures the *long-term reliability* experienced by a sensor node, and is requested to be higher than a minimum value, R_D^{min}.

2. *Miss ratio* (R_M): fraction of times the packet delivery ratio – calculated by a sensor node over the current Beacon Interval – drops below R_D^{min}. It measures the inability to achieve *short-term reliability* and should not exceed a pre-defined threshold R_M^{max}.

3. *Average energy consumption per packet* (E_P): total energy consumed by a sensor node divided by the total number of packets generated by that node. It measures *energy efficiency.*

Let $R_D(\mathbf{par})$, $R_M(\mathbf{par})$, and $E_P(\mathbf{par})$ denote the delivery ratio, miss ratio, and average energy consumption, respectively, experienced by a sensor node when using a set of CSMA/CA parameter values denoted by $\boldsymbol{par}$. Hence, the problem of optimal parameters

setting can be formulated as

$$\begin{cases} minimize\ E_P\left(\mathbf{par}\right) \\ R_D\left(\mathbf{par}\right) \geq R_D^{min} \\ R_M\left(\mathbf{par}\right) \leq R_M^{max} \end{cases} . \tag{5.1}$$

A possible approach for solving this problem is deriving an analytical model of the WSN and calculating the CSMA/CA parameter values that satisfy (5.1). This is very close to the approach used in [22], where the authors consider delivery ratio and latency (instead of miss ratio). As emphasized in the previous chapter, the use of a model-based approach has some limitations that make it unsuitable for real-life scenarios. In this chapter we use a heuristic solution, following a measurement-based approach, that leverages a change detection test and a learning algorithm to identify the optimal setting adaptively.

5.3 JIT-LEAP Algorithm Description

In this section we describe the proposed JIT-LEAP algorithm. We start with a high-level description (Section 5.3.1). Then, we detail the different phases of the algorithm (Sections 5.3.2-5.3.4). Finally, we describe some optimizations for improving its energy efficiency (Section 5.3.5).

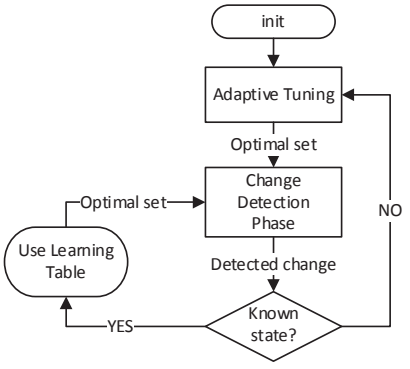


FIGURE 5.1: JIT-LEAP operating flowchart

5.3.1 Basic Ideas

JIT-LEAP aims to dynamically select the optimal CSMA/CA setting (depending on the current operating conditions) that guarantees the reliability required by the application

with minimum energy consumption. To this end, it also exploits the knowledge learned in the past history (if any). Figure 5.1 summarizes the main phases of JIT-LEAP. At each Beacon Interval, a sensor node measures some quantities that characterize the current network congestion and channel unreliability. Then, the following two phases can be distinguished:

1. *Adaptive Tuning.* When no information about the current operating conditions is available (e.g., at startup), the sensor node executes an Adaptive Tuning similar to ADAPT [30]. CSMA/CA parameter values are increased when the reliability experienced by the sensor node (in terms of R_D and R_M) does not satisfy the application requirements, and decreased otherwise. After a number of steps, the Adaptive Tuning algorithm starts oscillating between two parameter sets. This means that the optimal setting, for the current conditions, has been reached. Hence, this information is inserted into an appropriate data structure called *Learning Table* (see below for details). Then, the algorithm moves to the *Change Detection Phase.*

2. *Change Detection Phase.* This phase is aimed at detecting possible changes in the operating conditions, so as to adapt to the new conditions. To this end, a *Change Detection Test (CDT)*, i.e., an on-line statistical test, is considered to detect possible variations in the state variables measuring network congestion and channel unreliability. The statistical test is executed at each Beacon Interval, until a change is detected.

When no change is detected, the current CSMA/CA parameter values used by the sensor node are not updated and no further actions are performed. Whenever a change is detected, it is necessary to determine the optimal setting for the new conditions. If similar conditions have been already experienced in the past, the learning mechanism allows to immediately re-activate the optimal setting previously used. Specifically, upon detecting a change, the Learning Table is checked and the following two outcomes can occur.

1. Similar operating conditions have been already experienced in the past and, hence, the Learning Table contains an entry with the corresponding optimal set. Therefore, the node sets up the optimal parameter values suggested by the table (just in one step, or *leap*). Then, the Change Detection phase restarts.

2. There is no entry in the Learning Table for the new operating conditions. Therefore, the Adaptive Tuning phase is performed again. As soon as the Adaptive Tuning algorithm stabilizes and the optimal set is detected, a new entry is inserted in the Learning Table for the current operating conditions. Afterwards, the Change Detection phase is activated.

In the next subsections we will detail the actions carried out by JIT-LEAP during the Adaptive Tuning and Change Detection phases, and the data structures it uses.

5.3.2 Status Assessment and Data Structures

We denote by $\boldsymbol{s}(t)=[p_{busy}(t),\ p_{fail}(t),\ \boldsymbol{par}(t)]$ the state of the sensor network, as perceived by a generic sensor node, at a given Beacon Interval t. Specifically, $\boldsymbol{par}(t)$ is the used set of CSMA/CA parameter values, $p_{busy}(t)$ denotes the probability to find the channel busy during a channel access, and $p_{fail}(t)$ gives the probability that a packet transmission fails (i.e., the sensor node does not receive the related acknowledgment). In particular, p_{busy} is a measure of the congestion experienced by the sensor node and depends on some factors such as number of sensor nodes and offered load. It also depends on the CSMA/CA parameter setting used by the sensor node, i.e., $\boldsymbol{par}$. On the contrary, p_{fail} measures the communication unreliability and mainly depends on the wireless medium unreliability (i.e., PER). However, it also depends on the network congestion since a transmission can fail also due to a collision.

The state vector $\boldsymbol{s}(t)$ is derived by the sensor node as follows. The CSMA/CA parameter values (i.e., $\boldsymbol{par}$) are known. p_{busy} can be measured locally as $p_{busy} = p^1_{busy} + \left(1 - p^1_{busy}\right) \times p^2_{busy}$, where p^1_{busy} (p^2_{busy}) is the probability to find the channel busy during the first (second) CCA operation. In practice, p^1_{busy} and p^2_{busy} are estimated by calculating the fraction of CCA operations resulted in a busy channel in the current Beacon Interval. Similarly, p_{fail} is computed as the fraction of the transmissions for which the acknowledgment was missed during the current Beacon Interval.

At each sensor node JIT-LEAP uses the following data structures to store information.

1. *Data Cluster.* A table with an entry for each CSMA/CA parameters setting used since the last change in the operating conditions (or network startup). The cluster

is cleared whenever a new change is detected. Each entry has the following format: <***par***, R_D, R_M, R_F, *count*>, where R_D (R_M) represents the delivery ratio (miss ratio) experienced by the sensor node with the ***par*** parameters set. R_F denotes the *Transmission Failure Ratio*, defined as the ratio between the number of transmissions for which the acknowledgment was missed, and the total number of transmissions performed by the sensor node using the ***par*** parameters set. Finally, *count* indicates the number of times the corresponding parameter set has been used so far.

2. *Training Buffer.* A data structure containing a buffer with the last W states experienced during the Adaptive Tuning phase, or after the optimal set has been obtained using the Learning Table. It is needed for the CDT training, both at network startup and after a change detection.

3. *State Sample.* Whenever the CDT detects a change in the operating conditions, it calculates the mean value and standard deviation of p_{busy} and p_{fail} over the Beacon Intervals immediately following the change. These values characterize the new operating conditions. Thus, they are inserted into a proper data structure, called *State Sample*, that will be used to build the *Learning Table* at the end of the Adaptive Tuning phase.

4. *Learning Table.* This data structure is created at the end of the first Adaptive Tuning phase, and updated after each Adaptive Tuning phase, on the basis of the State Sample. The Learning Table contains information about each operating condition experienced during the past history, and the corresponding optimal setting, according to the learned knowledge. Each entry in the table has the following format: <***par***, $elem_1$, $elem_2$, ..., $elem_i$, ...>, where $elem_i = <\left[p^i_{busy\_min}, p^i_{busy\_max}\right]$, $\left[p^i_{fail\_min}, p^i_{fail\_max}\right]$, ***new_set***> for any i. Basically, each operating condition is represented by an element *elem*, where the two intervals $[p_{busy\_min}, p_{busy\_max}]$ and $[p_{fail\_min}, p_{fail\_max}]$ indicate the range of p_{busy} and p_{fail} characterizing that specific operating condition. Therefore, the table suggests that, whenever the estimated values of p_{busy} and p_{fail} fall within the above-mentioned intervals, and the parameters set ***par*** is used, the new optimal setting must be ***new_set***. Examples on how to access and use the Learning Table will be given below.

5.3.3 Adaptive Tuning Phase

5.3.3.1 CSMA/CA parameters change

Initially, JIT-LEAP starts with a simple Adaptive Tuning algorithm to dynamically adjust CSMA/CA parameters, as it has no information about the past history. The Adaptive Tuning increases or decreases one parameter at a time, depending on the experienced reliability, as follows. At each Beacon Interval, the sensor node updates the estimates of delivery ratio R_D and miss ratio R_M with measurements taken in the current Beacon Interval (see below). If at least one of these estimates does not satisfy the application requirements, the value of a CSMA/CA parameter is increased, by considering first *macMinBE* and, then, *macMaxCSMABackoffs* (*macMaxBE* is kept to a fixed value, i.e., $MaxBE^{max}$). The retransmission mechanism is initially disabled. Only when both *macMinBE* and *macMaxCSMABackoffs* have reached their maximum value, *macMaxFrameRetries* is also progressively increased. Conversely, if both R_D and R_M satisfy the application requirements, the set of parameter values is tentatively reduced. The strategy for decreasing parameter is just the opposite. First, *macMaxFrameRetries* is reduced up to when it reaches its minimum value. Then, the same procedure is applied to *macMaxCSMABackoffs* and, afterwards, to *macMinBE*. Without loss in generality, we can assume that CSMA/CA parameter sets are ordered as shown in Table 5.3. Hence, each parameter set can be identified by the corresponding index in the table and the Adaptive Tuning algorithm always moves from a set to an adjacent one.

Table 5.1: Ordered CSMA/CA Parameter Sets

index	**MaxBE**	**MinBE**	**MaxCSMABackoffs**	**MaxFrameRetries**
1	$MaxBE_{MAX}$	$MinBE_{MIN}$	$MaxCSMABackoffs_{MIN}$	$MaxFrameRetries_{MIN}$
2	$MaxBE_{MAX}$	$MinBE_{MIN}+1$	$MaxCSMABackoffs_{MIN}$	$MaxFrameRetries_{MIN}$
3	$MaxBE_{MAX}$	$MinBE_{MIN}+2$	$MaxCSMABackoffs_{MIN}$	$MaxFrameRetries_{MIN}$
...	...	...	...	...
...	$MaxBE_{MAX}$	$MinBE_{MAX}$	$MaxCSMABackoffs_{MIN}$	$MaxFrameRectries_{MIN}$
...	$MaxBE_{MAX}$	$MinBE_{MAX}$	$MaxCSMABackoffs_{MIN}+1$	$MaxFrameRetries_{MIN}$
...	$MaxBE_{MAX}$	$MinBE_{MAX}$	$MaxCSMABackoffs_{MIN}+2$	$MaxFrameRetries_{MIN}$
...	...	...	...	...
...	$MaxBE_{MAX}$	$MinBE_{MAX}$	$MaxCSMABackoffs_{MAX}$	$MaxFrameRetries_{MIN}$
...	$MaxBE_{MAX}$	$MinBE_{MAX}$	$MaxCSMABackoffs_{MAX}$	$MaxFrameRetries_{MIN}+1$
...	$MaxBE_{MAX}$	$MinBE_{MAX}$	$MaxCSMABackoffs_{MAX}$	$MaxFrameRetries_{MIN}+2$
...	...	...	...	...
I_{max}	$MaxBE_{MAX}$	$MinBE_{MAX}$	$MaxCSMABackoffs_{MAX}$	$MaxFrameRetries_{MAX}$

5.3.3.2 Data Cluster update

As anticipated, CSMA/CA parameter values are increased or decreased depending on the current estimates of R_D e R_M. For this reason, for each used set, the algorithm stores, inside the Data Cluster, the value of R_D e R_M experienced with that set. At each Beacon Interval, the entry corresponding to the current set is updated, as follows.

$$R_D = \frac{\overline{R_D} + R_D \cdot count}{count + 1} \qquad R_M = \frac{\overline{R_M} + R_M \cdot count}{count + 1}$$

$\overline{R_D}$ and $\overline{R_M}$ represent the delivery ratio and miss ratio measured in the current Beacon Interval, while *count* tracks the number of times the corresponding set has been used so far (it is increased after each update). This way, the estimates of R_D e R_M get more and more accurate over time.

If the channel is ideal (i.e., $PER = 0$), $\overline{R_D}$ can be obtained as $\overline{R_D} = P_{ACK}/P_{gen}$, where P_{gen} is the number of packets generated by the sensor node and P_{ACK} is the number of acknowledgements received from the sink. Furthermore, if $\overline{R_D} > R_D^{min}$, then $\overline{R_M} = 0$; otherwise $\overline{R_M} = 1$. If the channel is not ideal (i.e.,$PER > 0$), the previous formula for $\overline{R_D}$ underestimates the delivery ratio since a packet may have been delivered correctly to the sink even if the corresponding acknowledgment was missed by the sensor node. To correctly estimate the delivery ratio, when $PER > 0$, we need to take into account also packets dropped due to exceeded number of retransmissions, but correctly received by the sink. Let P_{MFR} denote the total number of packets dropped due to exceeded number of retransmissions and α be the probability that a dropped packet is received correctly by the sink. The delivery ratio can be estimated as $\overline{R_D} = \frac{P_{ACK}+P_{MFR}\cdot\alpha}{P_{gen}}$. The value of P_{MFR} is provided by the MAC layer, while α can be derived using the following claim.

Claim 1. Assuming that *(i)* packet transmission errors are independent from each other, and *(ii)* the PER is the same for both data packets and acknowledgements, then

$$\alpha = 1 - \left(\frac{\mathrm{R_F} - \mathrm{PER}}{(1 - \mathrm{PER})\mathrm{R_F}} \right)^{\mathrm{macMAXFrameRetries}+1}$$

where $\mathrm{R_F}$ denotes the transmission failure ratio, i.e., the probability that a data packet transmission fails for any reason.

Proof. See Appendix.

The previous claim allows to derive α, once PER and R_F are known. PER is estimated by the sensor node by computing the ratio between the number of missed Beacons and the number of expected Beacons. Instead, the transmission failure ratio R_F is estimated by taking an approach similar to that used for estimating R_D. As mentioned above, for each used parameter set, the corresponding entry in the Data Cluster also includes a field R_F. The latter is updated, at each Beacon Interval, as $R_F = \frac{\overline{R_F}+R_F \cdot count}{count+1}$, where $\overline{R_F}$ is the ratio between the number of missed acknowledgements and the total number of transmissions (including retransmissions) performed by the sensor node in the current Beacon Interval.

5.3.3.3 Training Buffer and Learning Table update

In addition to the Data Cluster, the Training Buffer and Learning Table are also updated during the Adaptive Tuning phase. At each Beacon Interval t, after estimating p_{busy} and p_{fail}, the new state $\mathbf{s}(t) = [p_{busy}(t),\ p_{fail}(t),\ \mathbf{par}(t)]$ is added to the Training Buffer. Since the Training Buffer has a limited size, when it is full, the new state overwrites the oldest one, following a FIFO approach. We emphasize that, due to its behavior, after a (short) transient time the Adaptive Tuning algorithm tends to oscillate between two adjacent parameter sets. We assume that the Adaptive Tuning phase ends when all the states stored inside the Training Buffer refer to only two parameter sets. Then, the Training Buffer is ready to be used for training the CDT as described below. The most frequent setting within the Training Buffer is assumed to be the most appropriate set for the current operating conditions, i.e., the "optimal" set according to the Adaptive Tuning algorithm. Throughout, we will refer to this set as $\mathbf{par}_{opt}$.

Now, a new element can be added in the Learning Table, pointing to the optimal set $\mathbf{par}_{opt}$. Let us denote by $\mathbf{par}_{prev}$ the parameter set used before the current Adaptive Tuning phase started, i.e. before the operating conditions changed. Also, let us indicate as μ_{busy} (μ_{fail}) and $\sigma_{busy}(\sigma_{fail})$ the mean value and standard deviation of p_{busy} (p_{fail}) calculated over the Beacon Intervals immediately following the change, inserted by the CDT into the State Sample. These values characterize the current operating conditions. Then, a new entry corresponding to set-index $\mathbf{par}_{prev}$ is inserted in the Learning Table

(if there is no entry for this set, a new entry is created). The new added element is

$$< \left[\mu_{busy} - \sigma_{busy},\ \ \mu_{busy} + \sigma_{busy}\right],\ [\mu_{fail} - \sigma_{fail},\ \ \mu_{fail} + \sigma_{fail}],\ \mathbf{par}_{opt} >$$

In the future, if these operating conditions are encountered again, while the set $\mathbf{par}_{prev}$ is used, the algorithm immediately knows that the set $\mathbf{par}_{opt}$ has to be used. The update of the Learning Table concludes the Adaptive Tuning phase. Then, the sensor node enters the Change Detection phase.

5.3.4 Change Detection Phase

5.3.4.1 Change Detection Test

Detecting changes in the operating conditions is a primary ability of JIT-LEAP. In particular, it detects possible changes by inspecting variations in p_{busy} and p_{fail}. To achieve this goal, among the wide range of CDTs available in the literature [65–69], we focus on the family of CDTs based on the *Intersection-of-Confidence-Interval (ICI)* rule [70], which prove to be particularly effective in several application scenarios [62, 71]. In addition, ICI-based CDTs are theoretically grounded and exhibit a reduced computational complexity, which makes them particularly suitable for WSNs. Finally, this family of CDTs follows the "non-parametric" approach, i.e., they do not require any a-priori information about the measured state variables or changes that might affect the network. This makes ICI-based CDTs particularly suitable for time-varying and a-priori unknown environments (such as WSNs). Among ICI-based CDTs, we focus on the element-wise CDT [72]. This CDT is able to operate in an element-wise manner, thanks to Gaussian transform of measured variables, providing very prompt detections to changes. The considered Gaussian transform is the Manly transform [72], i.e.,

$$\overline{p}_{index}(t) = \begin{cases} \frac{e^{\lambda p_{index}(t)} - 1}{\lambda}; & \lambda \neq 0 \\ p_{index}(t); & \lambda = 0 \end{cases}$$

where $p_{index}(t)$can be either $p_{busy}(t)$ or $p_{fail}(t)$and $\lambda \in \mathbb{R}$ is the transform parameter. The Manly transform is applied both to $p_{busy}(t)$ and $p_{fail}(t)$ to generate the approximately Gaussian variables $\overline{p}_{busy}(t)$ and $\overline{p}_{fail}(t)$. As mentioned above, this CDT requires

an initial training sequence to configure the test parameters and the parameter λ of the Manly transform. In our scenario the CDT is configured on the Training Buffer to estimate the sample mean and variance of $\overline{p}_{busy}(t)$ and $\overline{p}_{fail}(t)$ stored therein. Details about the configuration phase of the CDT can be found in [72]. During the operational life, the ICI-rule is then applied to $\overline{p}_{busy}(t)$ and $\overline{p}_{fail}(t)$ to detect possible variations in their expected values, with respect to what trained during the training phase.

When a change is detected, the Learning Table is looked up to determine the optimal set for the new operating conditions. A key requirement for obtaining correct values from the Learning Table is the ability to correctly characterize the operating conditions after the change, in terms of the new values of p_{busy} and p_{fail}. To achieve this goal, once a change is detected, a Change-Point Method (CPM) is applied to a buffer containing the last W acquired data to identify the time instant at which the operating conditions changed. CPMs are statistical hypothesis tests [73] able to assess whether a change point exists in a given sequence of data and locate it within the sequence. Specifically, let $\hat{T}$ be the Beacon Interval when the change was detected (either in p_{busy} or p_{fail}), and let $\mathcal{X}$ be the sequence of the corresponding variable up to $\hat{T}$, i.e.,

$$\mathcal{X} = \left\{\overline{p}_D\left(\left(\hat{T}-W+1\right)\right),\ldots,\overline{p}_D\left(\hat{T}\right)\right\}$$

where $\overline{p}_D(t)$ is either p_{busy} or p_{fail}. The CPM acts as follows. For each Beacon Interval t, such that $\hat{T}-W+1 \leq t \leq \hat{T}$, the sequence $\mathcal{X}$ is split into two parts

$$A_t = \{\overline{p}_D\left(\hat{T}-W+1\right),\ldots,\overline{p}_D(t)\}$$
$$B_t = \{\overline{p}_D(t+1),\ldots,\overline{p}_D\left(\hat{T}\right)\}$$

and a test statistics $\mathcal{T}_t$=(A_t, B_t) is computed for all the Beacon Intervals t ($\hat{T}-W+1 \leq t \leq \hat{T}$). Let $\mathcal{T}_M$ be its maximum value, i.e.,

$$\mathcal{T}_M = \max_{t=\hat{T}-W+1,\ldots,\widehat{T}} \mathcal{T}_t \ .$$

When $\mathcal{T}_M$ is larger than a predefined threshold $H_{\varepsilon,\hat{T}}$ (that depends on the test statistic, $\hat{T}$ and a given confidence level) there is enough statistical confidence that a change point

exists in $\mathcal{X}$. Let τ be the time instant of this change point, i.e.,

$$\tau = \underset{t=\hat{T}-W+1,\ldots,\widehat{T}}{\operatorname{argmax}} \mathcal{T}_t \ .$$

Since we are interested in detecting change-points affecting the expected value of $\mathcal{X}$, the test statistic we are considering is the Mann-Whitney [74]. Other test statistics able to assess variations in the expected values could be considered as well. We emphasize that τ represents the Beacon Interval at which a change affected the operating conditions. Hence, the new operating conditions can be computed as follows:

$$\mu = \frac{1}{\hat{T}-\tau} \sum_{i=\tau}^{\hat{T}} \overline{p}_D\,(i)$$
$$\sigma = \frac{1}{\hat{T}-\tau-1} \sum_{i=\tau}^{\hat{T}} \left(\overline{p}_D\,(i) - P_D\right)^2$$

where u and σ are respectively the sample mean and sample variance of the state variable $\overline{p}_D(t)$ that detected the change from τ to $\hat{T}$. The values of u and σ, for both p_{busy} and p_{fail} (u_{busy}, σ_{busy}, u_{fail} and σ_{fail}), represent the new operating conditions and are stored inside the State Sample.

5.3.4.2 Learning Table access

Once a change has been detected, the Learning Table is checked to verify whether the new conditions have been already experienced in the past. To this aim, the current set index, along with the values of u_{busy} and u_{fail} contained in the State Sample, are used.

TABLE 5.2: Example of Learning Table

Current Set	**Elem1**			**Elem2**		
	$\mathbf{p_{busy}}$	$\mathbf{p_{fail}}$	**new set**	$\mathbf{p_{busy}}$	$\mathbf{p_{fail}}$	**new set**
par1	[0.30, 0.33]	[0.33, 0.52]	par2	[0.64, 0.70]		par3
par2	[0.43, 0.51]	[0, 0.24]	par3			
par3	[0.70, 0.75]	[0.16, 0.22]	par5			
par4	[0.64, 0.76]	[0.20, 0.43]	par8	[0.25, 0.35]	[0, 0.15]	par1

Table 5.2 shows an example of Learning Table. Let us assume that ***par4*** is the current parameter set and that $\widehat{\mu}_{busy}$ and $\widehat{\mu}_{fail}$ are the values of μ_{busy} and μ_{fail} stored in the State Sample. Based on the past history, the Learning Table suggests to use ***par8*** as

the new set, if $\widehat{\mu}_{busy}$ is in the range [0.64, 0.76] and $\widehat{\mu}_{fail}$ is in the range [0.20, 0.43], or ***par1,*** if $\widehat{\mu}_{busy}$ is in the range [0.25, 0.35] and $\widehat{\mu}_{fail}$ in the range [0, 0.15].

When the Learning Table does not contain any entry for the values in the State Sample, JIT-LEAP infers that the current operating conditions have never been experienced in the past and a new Adaptive Tuning phase is activated to identify the optimal MAC parameters set. When the optimal set is determined, at the end of the Adaptive Tuning phase, a new entry is added to the Learning Table by using the data contained in the State Sample, as previously explained.

5.3.5 Controlled Tuning Algorithm

The Adaptive Tuning phase allows to identify the parameters setting that guarantees the reliability required by the application with minimum energy consumption. However, since CSMA/CA parameters assume discrete values, typically the delivery ratio (miss ratio) experienced with the obtained parameter set is significantly above (below) R_D^{min} (R_M^{max}), thus consuming more energy than necessary. On the other hand, using a lower set might not guarantee the application requirements.

To optimize the energy consumption while still satisfying the reliability constraints, JIT-LEAP uses a *Controlled Tuning* algorithm that finely adjusts the parameters setting on the sensor node, by switching between two adjacent sets in a controlled way. The idea is to have a reliability level just above the required value, so as to minimize the energy consumption. The Controlled Tuning algorithm is detailed in Appendix.

5.4 Simulation Setup

To evaluate the performance of JIT-LEAP we used the *ns2* simulation tool [54]. We considered a star network scenario, where sensor nodes are placed in a circle centered at the sink node (PAN coordinator), 10m far from it. The transmission range was set to 15m, while the carrier sensing range was set to 30m (according to [75]). In our analysis, in addition to the reliability and energy efficiency indexes already introduced in Section 5.2 (i.e., *packet delivery ratio*, *miss ratio* and *average energy consumption per packet*), we also considered the following two performance indexes.

1. *Average latency*, defined as the average time since the packet transmission starts at the source node to when the packet is correctly received by the sink. This index characterizes *timeliness*.

2. *Transient time*, defined as the time instant, after a change in the operating conditions, when the packet delivery probability – calculated over the current Beacon Interval – reaches the steady-state value for the new operating conditions (with a tolerance of 3%). This performance index measures the *ability to adapt* to changing conditions.

The energy consumed by a sensor node is calculated following the model in [57], which is based on the Chipcon CC2420 radio transceiver [55].

5.4.1 Algorithms for Comparison

We compared the performance of JIT-LEAP with that of the *Model-based offline computation* [22], *Model-based online adaptation* [29] algorithm and *ADaptive Access Parameter Tuning (ADAPT)* [30].

5.4.2 Parameter Values and Methodology

Table 5.3 summarizes the operating parameters used in our simulation experiments. Since the other algorithms do not consider miss ratio when deriving the optimal setting, the operating parameters for these algorithms have been chosen in such a way to guarantee both R_D^{min} and R_M^{max} required by the application. This allows a fair comparison of the considered algorithms in terms of both energy consumption and latency.

In our experiments we considered both ideal and noisy channels. In the latter case, we used the Gilbert-Elliot (GE) model to simulate packet errors/losses as it provides a good approximation of fading in real environments [76]. We used the same values derived in [27] and inspired from measurements in [76]. Specifically, the PER in the bad (good) state of the GE model is 100% (0%). Sojourn times in both states are exponentially distributed and, for a PER equal to 10%, their average values are 5.7 (bad state) and 46.2 ms (good state). We also assumed that each sensor node generates 10 data packets

TABLE 5.3: CSMA/CA Parameters and values

Parameter	**Value**
Bit Rate	250 Kbps
Data Frame (Payload) Size	109 (100) bytes
ACK Frame Size	11 bytes
Beacon Order (BO), Superframe Order (SO)	13, 8
$MinBE^{min,max}$	1, 7
$MaxCSMABackoffs^{min,max}$	1, 10
$MaxFrameRetries^{min,max}$	0, 9
Power Consumption in RX, TX, Idle, Sleep mode	35.46 mW, 31.32 mW, 0.77 mW, 0.036 μW
W	15
γ	1.2

at every Beacon Interval. All these packets are passed together to the MAC layer, for transmission, at the beginning of the Beacon Interval.

For each simulated scenario, we performed 10 independent replications, each consisting of 1000 Beacon Intervals. For each replication we discarded the initial transient interval (10% of the overall duration) during which nodes associate to the PAN coordinator node and start generating packets. The results presented below are averaged over all the replications. We also derived confidence intervals using the independent replications method. They are typically so small that cannot be appreciated in the figures below.

5.5 Simulation Results

Our analysis is divided into two parts. In the first part, we compare the considered algorithms in *stationary* scenarios, i.e., we assume that the operating conditions do not change over time. In the second part we consider *dynamic* scenarios with time-varying operating conditions. Since the two model-based algorithms (implicitly) assume ideal channel conditions, in our analysis – both in stationary and dynamic scenarios – we initially assume that packet errors/losses never occur (i.e., PER = 0). Then, we restrict our analysis to ADAPT and JIT-LEAP only, and investigate the impact of PER on their performance.

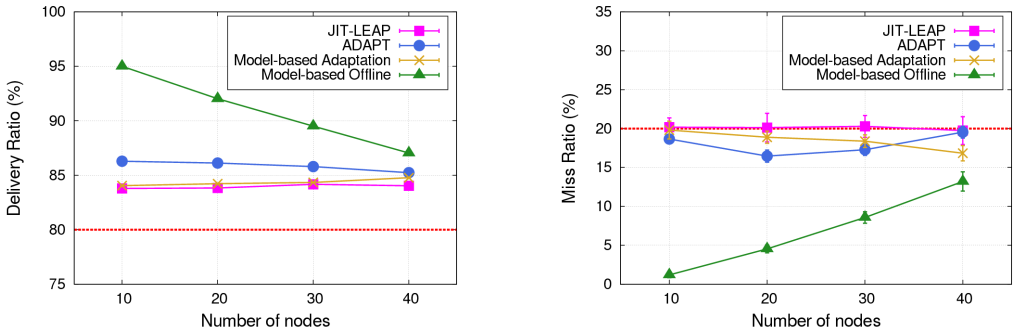
In our experiments we assumed that the application requires a packet delivery ratio $R_D \geq$ 80% and a miss ratio $R_M \leq 20\%$, for any sensor node (i.e., $R_D^{min} = 0.80$ and $R_M^{max} =$

0.20). Obviously, these thresholds are somehow arbitrary, as they strongly depend on the specific application. However, we performed additional experiments with different thresholds, and we achieved results in line with those presented here.

5.5.1 Analysis in stationary conditions

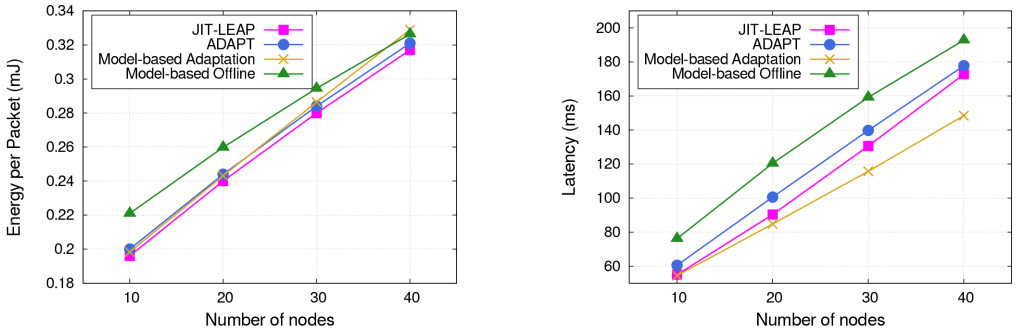
As mentioned above, we start our analysis in stationary conditions assuming ideal channel conditions. Figure 5.2 shows the delivery ratio and miss ratio of a generic sensor node, for an increasing size of the WSN. All the four algorithms satisfy the reliability requirements of the application (both in terms of R_D and R_M). The offline algorithm provides the highest delivery ratio and the lowest miss ratio. However, this results in a large latency and energy consumption, as shown in Figure 5.3. ADAPT provides a delivery ratio between the two thresholds, R_D^{low} and R_D^{high}, that have been determined to satisfy also the miss ratio constraint. The model-based adaptive algorithm and JIT-LEAP have similar performance in terms of delivery ratio and miss ratio. However, thanks to the Controlled Tuning algorithm, JIT-LEAP provides a delivery ratio (miss ratio) very close to R_D^{min} (R_M^{max}). This allows JIT-LEAP to minimize the energy consumption, as shown in Figure 5.3(a). We emphasize that JIT-LEAP is the best solution in terms of energy consumption. In terms of latency (Figure 5.3(b)), the model-based adaptive algorithm has the best performance. This is because the latter algorithm tends to improve the delivery ratio by increasing the number of retransmissions (*macMaxFrameRetries*), whereas the other algorithms achieve the same result by increasing the number of backoff trials (*macMaxCSMABackoffs*). When a packet is re-transmitted the contention window size is re-initialized to its minimum value. Instead, when a new backoff trial is started, the contention window size is doubled (unless it has reached its maximum value). However, the lower latency introduced by the model-based adaptive algorithm is paid in terms of a higher energy consumption. This is because increasing the number of re-transmissions is more energy consuming than increasing the number of backoff trials [30]. Finally, JIT-LEAP performs significantly better than both ADAPT and the model-based offline algorithm, also in terms of latency.

In the second set of experiments we considered non-ideal channel conditions. The corresponding results are in Appendix.



(a) (b)

FIGURE 5.2: Delivery ratio (left) and miss ratio (right) vs. number of sensor nodes

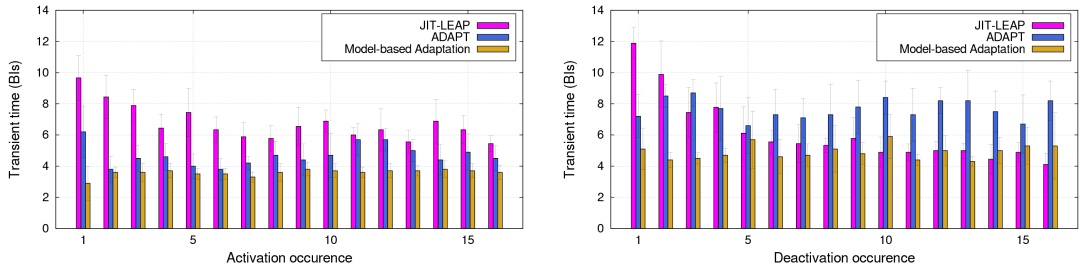


(a) (b)

FIGURE 5.3: Average per-packet energy consumption (left), and average latency (right) vs. number of sensor nodes

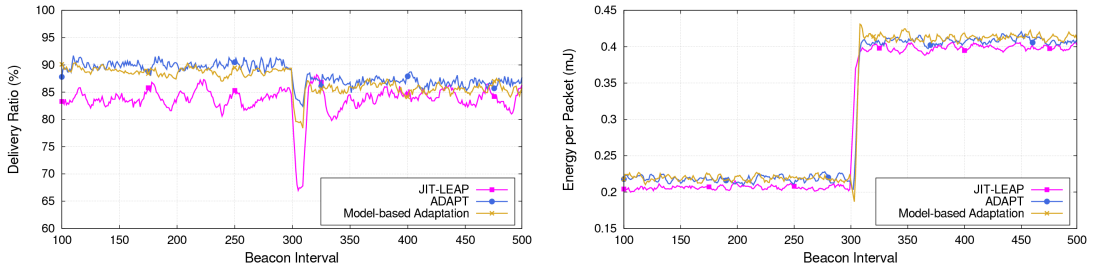
5.5.2 Analysis in dynamic conditions

We now turn our attention to dynamic scenarios, where operating conditions vary over time. We limit our analysis to adaptive algorithms, since the model-based offline algorithm is not suited for dynamic scenarios.



(a) (b)

FIGURE 5.4: Transient time when the number of active nodes increases (left) and decreases (right)



(a) (b)

FIGURE 5.5: Delivery ratio (left) and energy per packet (right) when the number of active nodes increases

In the first set of simulation experiments, we consider a scenario where the number of active sensor nodes changes over time. Specifically, we assume that 10 sensor nodes are always active, while 50 more sensor nodes are activated and deactivated simultaneously and periodically, every 300 Beacon Intervals. Our goal is to investigate how the different algorithms react to such changes. The results presented below refer to nodes that are always active. Figure 5.4 shows the transient time taken by the various algorithms to adapt to the new operating conditions. We analyzed separately the transient originated by an increase and a decrease in the number of active nodes (left and right side in Figure 5.4, respectively). As a general comment, the transient times experienced by JIT-LEAP tend to become shorter and shorter over time, while they remain approximately constant for the other algorithms. This is due to the learning mechanism used by JIT-LEAP. When the WSN passes through similar operating conditions experienced in the past, JIT-LEAP is able to find out the optimal setting by exploiting the information available in the Learning Table. If we look at deactivation events (Figure 5.4(b)), after some time, JIT-LEAP becomes significantly faster than the two other algorithms. In ADAPT the transient time depends on the number of active sensor nodes, as the algorithm converges to the optimal setting step by step, and a larger network generally requires higher CSMA/CA parameter values to guarantee the same reliability. The model-based adaptive algorithm converges in no more than 5 Beacon Intervals as the optimal setting is derived by using samples measured in the previous m Beacon Intervals (and we used m=4 in our experiments). However, both ADAPT and the model-based adaptive algorithm exhibits some drawbacks. The latter assumes to know in advance the number of (active) sensor nodes in the WSN. This is generally difficult to predict and may become a serious issue in dynamic scenarios. In our experiments, we ran the algorithm using the maximum number of sensor nodes (i.e., 60). This means that, when

there are only 10 nodes, the provided setting is not optimal and sensor nodes consume more energy than necessary. Conversely, running the algorithm with the minimum number of nodes (i.e., 10) has negative drawbacks as well. When the number of active nodes is higher, the algorithm may not satisfy the reliability constraints required by the applications. Similarly, ADAPT requires the definition of the thresholds R_D^{low} and R_D^{high} that strictly depends both on the reliability requirements (R_M^{max} and R_D^{min}) and the network congestion. When the number of sensor nodes increases, the two thresholds should take higher values in order to guarantee the same levels of R_D and R_M. As above, in our experiments, we referred to the worst case (60 active nodes) to define the input parameters, so as to satisfy the reliability constraints both with 10 and 60 nodes. JIT-LEAP does not suffer from such limitations, as it does not require any input parameter. This leads to significant benefits, especially in terms of energy consumption. Figure 5.5 compares the delivery ratio (left) and energy consumption (right) of the three algorithms, before and after an increase in the number of sensor nodes has occurred. JIT-LEAP provides a delivery ratio that is the closest to the application requirement (80%), thus allowing the lowest energy consumption.

In the second set of experiments we considered a scenario where the PER changes over time. The corresponding results are shown in Appendix.

5.5.3 Resource Usage

We conclude our analysis looking at the computational resource usage required by the considered algorithms. As a preliminary remark, we observe that the model-based offline algorithm does not require any computational/memory resource, since it is run offline. Thus, we will focus on the other three algorithms.

In terms of computational cost, ADAPT is the lightest one, as it only requires few simple operations to update the estimates. For the model-based adaptive algorithm, the authors suggest two implementations. In the first one (used in our experiments) the optimal setting is obtained by solving the analytical model at the sensor node, thus resulting in a significant computational load. In the second implementation, the optimal parameter values are computed offline and stored on the node in a look-up table. Obviously, the latter approach requires no computational cost but introduces a high memory occupancy. Finally, JIT-LEAP is particularly suitable to be executed on

sensor nodes. Like ADAPT, it has a lightweight Adaptive Tuning phase. The CDT is a light task as well, as it requires few simple calculations over the state variables. Only the CDT training and CPM are a bit more computationally intensive operations, however they are performed only when a change is detected. Hence, they introduce a slight additional computational load just in a small fraction of Beacon Intervals.

JIT-LEAP	ADAPT	MODEL-BASED ADAPTATION		OFFL. COMP.
		Online Computation	Lookup Table	
$\approx$ 1 KByte	$\approx$ 10 Bytes	$\approx$ 100 Bytes	1-10 Bytes	0 Bytes

TABLE 5.4: Memory occupancy

Let us now analyze the memory footprint. Table 5.4 shows the memory occupancy of the considered algorithms. Among the adaptive algorithms, ADAPT exhibits the smallest footprint, since it needs to store only some statistics and estimates. As regards the model-based adaptive algorithm, both the versions require the node to store the measured congestion indexes (which takes about 100 bytes). When the computation of the optimal set is carried out offline, the memory occupancy is much higher, due to the lookup table. The memory occupancy of JIT-LEAP strongly depends on the considered scenario, as the algorithm stores data about each used parameter set and experienced operating condition. As shown in Figure 5.6, the Learning Table is the more consuming data structure and its size increases if the operating conditions change frequently. Let M denote the maximum number of *elem* elements for each entry, and N the maximum number of entries (i.e. the number of possible parameter sets). Hence, the size S of the Learning Table can be derived as $S = N \cdot M \cdot E$, where and $E = sizeof(elem)$. In JIT-LEAP N is constant and equal to 19, and $E = 5$ *bytes*. Assuming $M = 10$ it yields $S = 950$ *bytes*. Figure 5.6 shows the memory space required by each data structure. In our experiments, both in stationary and dynamic scenarios, the observed footprint of JIT-LEAP was well below 1 Kilobyte.

5.6 Conclusions

In this chapter we have proposed a new Just-in-Time learning-based algorithm, called JIT-LEAP, for deriving the optimal CSMA/CA parameters setting in IEEE 802.15.4 sensor networks. The proposed algorithm adapts the CSMA/CA parameters so as to

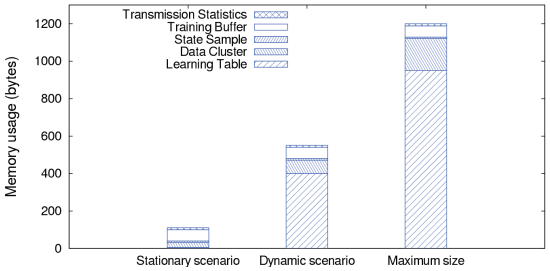


FIGURE 5.6: Memory occupancy in JIT-LEAP

guarantee the reliability constraints required by the application, with minimum energy consumption, on the basis of the reliability experienced by the sensor nodes (measured locally). Unlike previous similar adaptive algorithms, it also exploits a learning mechanism to speed up the transient time when the network operating conditions have been already experienced in the past and, thus, information about the optimal setting is already present in a Learning Table. We have analyzed our algorithm both in stationary and dynamic scenarios. Our simulation results have shown that the proposed JIT-LEAP algorithm outperforms all previous similar algorithms.

Through the use of JIT-LEAP is possible to significantly increase the reliability of 802.15.4 networks, making them suitable for applications having reliability as the main concern. However, JIT-LEAP is not a viable solution for time-critical applications since the increase in reliability comes at the cost of increased packet latency. When applications have stringent requirements in terms of both reliability and timeliness, TDMA is typically used for regulating channel access. Therefore, from the next chapter we focus on analyzing TDMA-based WSNs.

Part II
TDMA-based WSNs

Chapter 6

Background on TDMA-based WSNs

6.1 Introduction

The second part of this thesis is devoted to the analysis of TDMA-based WSNs. In this chapter we first highlight the limitations of TDMA-based WSNs. Then, we review literature on TDMA-based WSNs and motivate the work we present in chapters 7 and 8.

6.2 Limitations of TDMA-based WSNs

Many application domains require very high reliability and, at the same time, low and predictable latencies as well as energy efficiency. In such scenarios a *Time Division Multiple Access* (TDMA) scheme is typically used for regulating channel access. As well known, in TDMA-based systems time is divided into a sequence of periodic *superframes*, each one of which consists of a fixed number of transmission *slots*. Slots are allocated to sensor nodes so that each node needs to be active only during its own slot(s), while it can sleep for the rest of the time. Therefore, TDMA provides guaranteed bandwidth, high energy efficiency, absence of collisions (i.e., high reliability), low and predictable latency. For these reasons a number of TDMA-based MAC protocols for WSNs have been proposed in the literature [77–80]. However, TDMA also suffers from a number of

limitations that limit its suitability for critical application scenerios. First of all, a strict synchronization among sensor nodes is needed [31, 32]. Second, TDMA has a limited *flexibility* since an allocation of slots to sensor nodes has to be performed in order for the network to operate [33] and a different slot allocation pattern (and hence a re-execution of the slot allocation procedure) is usually required if a change in the network topology occurs. Finally, TDMA suffers from *selective jamming* attack [12, 34], a particularly insidious form of Denial-of-Service (DoS) that allows an adversary to completely thwart the communication of a victim node with a very low probability to be detected.

In this thesis we focus on the two latter limitations. Specifically, we present two original proposals to both provide a flexible solution to the problem of slot allocation and to contrast selective jamming attacks in TDMA-based WSNs. In chapter 7 we propose LOCALL, a localized slot allocation algorithm which allows sensor nodes of a TDMA-based WSN to select their own transmission slot(s) in a completely autonomous way, just relying on local information. Thanks to its localized approach LOCALL does not require the exchange of messages to establish a communication schedule. This minimizes energy consumption of sensor nodes and makes LOCALL particularly suitable both for environments where packets can be corrupted or missed and dynamic networks. In chapter 8 we propose JAMMY, a novel distributed and self-adaptive solution against selective jamming attacks in TDMA-based WSNs. Unlike previous approaches, JAMMY is completely *distributed*, i.e. it does not suffer from typical limitations of centralized solutions (e.g. single point of failure). Moreover, JAMMY introduces a negligible computation overhead and no communication overhead at all.

The remainder of this chapter is organized as follows. In section 6.3.1 we review works on link scheduling for TDMA-based WSNs and highlight the novelty of LOCALL. In section 6.3.2 we describe the solutions that have been proposed in the literature to contrast selective jamming attack and we compare them with JAMMY.

6.3 Literature review and our contributions

6.3.1 Link scheduling in TDMA-based WSNs

Due to the broadcast nature of the wireless medium, packets transmitted by a sensor node can be received by any node within its transmission range. Hence, transmission on one link[1] can, in general, interfere with the reception on another link.

In TDMA-based WSNs a link scheduling algorithm is typically used to avoid interferences between links. Specifically, a link scheduling algorithm has the role to assign to each link in the network a number of timeslots in the superframe for data transmissions so that an interference-free communication pattern is established, i.e. every scheduled transmission has not to result in a collision both at the sender and at the receiver. A number of link scheduling algorithms have been proposed in the context of WSNs. In [33] they have been classified in four classes according to their main objective. The four identified objectives are i) *minimizing superframe length* [81–86], ii) *minimizing latency* [87–90], iii) *minimizing energy consumption* [91–93], iv) *maximizing fairness* [94, 95]. Also, algorithms targeting multiple objectives have been proposed [92, 96–98]. Regardless of its target(s) any link scheduling algorithm always falls in one of the two following categories: *centralized* algorithms or *distributed* algorithms.

In centralized algorithms a master node collects information about the entire network topology and on the traffic load at each sensor node. Then, such master node computes an interference-free schedule for the whole network and provides each node in the network with its time schedule. A centralized approach requires a multi-hop signaling phase, which could impair the power efficiency of TDMA. For this reason, centralized algorithms work very well when the network is (quasi) static [99, 100], while they are not suitable for dynamic networks where sensor nodes can join and leave frequently and network topology dynamically changes.

Conversely, distributed algorithms provide that the communication pattern is established in a distributed way by part of all the sensor nodes in the network. In distributed solutions, when a node joins/leaves the network only the nodes in its neighborhood have to adjust their communication schedule and, hence, the overhead due to the scheduling is reduced. This makes distributed algorithms particularly suitable for dynamic

[1] With the term *link* we denote a sender-receiver couple

networks. However, the majority of distributed algorithms are not *localized*, as sensor nodes typically need to exchange special control messages with their neighbors to acquire the right to use one or more slots in a dedicated way. This may be a problem in many real environments where packet loss is relevant. In addition, sensor nodes must remain active during the entire slot negotiation phase, to exchange information with other nodes, thus consuming energy. For these reasons, in the next chapter we propose LOCALL, a localized slot allocation algorithm which allows sensor nodes of a TDMA-based WSN to select their own transmission slot(s) in a completely autonomous way, just relying on local information. Thanks to its localized approach LOCALL does not require the exchange of messages to establish a communication schedule. This minimizes energy consumption of sensor nodes and makes LOCALL particularly suitable both for environments where packets can be corrupted or missed and dynamic networks.

6.3.2 Literature on selective jamming attack in WSNs

The shared and easy-to-access wireless medium makes WSNs particularly vulnerable to jamming attacks. In fact, jamming is considered one of the most common DoS attacks, as well as a severe security issue in wireless communication [101][102][103][104][105].

In the context of WSNs, jamming attacks aim at interfering with network's operational frequencies. Xu *et al.* have classified possible jamming attacks in WSNs as *constant*, *deceptive*, *random*, and *reactive* [105]. The objective of a *constant* jammer is to corrupt *all* network packets, by continually transmitting random signals. However, such an "always-on" jamming strategy is based on the continuous presence of a high interference level, hence it is easy to detect [101][104][105]. On the other hand, a *deceptive* jammer injects a constant stream of bytes into the network, making it look as legitimate traffic. Unlike constant jamming, deceptive jamming is harder to detect using monitoring tools, since legitimate traffic is sent on the medium. The main disadvantage of both the aforementioned jamming strategies is their power inefficiency that limits the attacker ability to be perpetual (i.e. to not depend on an external power source). In this regard, a more efficient strategy is *random* jamming. It consists in alternating sleep phases and jamming phases, thus reducing power consumption, but this is usually less effective than constant and deceptive jamming. Finally, a smarter and more power efficient approach is *reactive* jamming, which performs jamming only when transmissions from other nodes

take place. Reactive jamming is likely to be confused with regular collisions, hence it is much more difficult to detect. Also, reactive jamming is efficient in terms of power consumption.

In [11][12][106], the authors consider *selective jamming*, a particular form of reactive jamming aimed at disturbing communication among sensor nodes according to specific criteria and objectives. With respect to jamming strategies mentioned above, selective jamming is more difficult to detect, due to the reduced adversary exposure, as well as more power efficient. In chapter 8, we focus on a specific type of selective jamming, where the adversary aims at disrupting communication of one particular sensor node. Such an attack is very easy and effective to perform in a TDMA-based WSN.

This is because since a node of a TDMA-based WSN retains an allocated slot for many consecutive superframes, an adversary has to monitor communication, detect such a slot and jam it in order to completely thwart the node communications. Also, the attack is power efficient, as the adversary has to activate her radio only during slots used by the victim node.

Solutions against jamming proposed in the literature can be divided into *physical-layer* solutions and *MAC-layer* countermeasures (our proposed solution JAMMY belongs to the latter class).

Physical-layer solutions try to prevent a jammer from interfering with network operational frequencies. The most relevant proposals in this class have been surveyed in [107] by Mpitziopoulos *et al.*. The authors mainly consider *Frequency-Hopping Spread Spectrum* (*FHSS*) [108], i.e. a spread-spectrum transmission method that switches a carrier among many frequency channels, according to an algorithm shared by transmitter and receiver. Frequency hopping is based on the premise that operating on a channel orthogonal to the jammer's one suppresses the jamming interference. However, since current commercial systems use only a small number of orthogonal bands, and adjacent orthogonal channel interference exists, frequency hopping has been shown to be rather ineffective [109]. Also, *Direct Sequence Spread Spectrum* (*DSSS*) has been considered [108]. It consists in multiplying data to be transmitted (RF carrier) by a Pseudo-Noise (PN) digital signal having a frequency (chip rate) much higher than the original signal's. This replaces the original RF signal with a wide bandwidth one displaying a spectrum equivalent to a noise signal's, so minimizing unauthorized interception and jamming of

radio transmission between nodes. However, Mpitziopoulos *et al.* stress that, although the IEEE 802.15.4 standard [10] relies on DSSS, this does not make it invulnerable to jamming attack. Morever, the network is likely to be taken down by jamming, due to the limited supported chip rate, and the restricted transmission power of sensor nodes. The main drawback of physical layer solutions is that they are not actually able to *neutralize* jamming attack.

MAC-layer countermeasures assume that it is always possible for a jammer to interfere with network's regular transmissions and make use of security schemes to contrast jamming. The majority of them addresses constant jamming [110][111][104][112], while considerable less solutions target selective jamming [11][113][12][114].

In the following, we survey only MAC-layer countermeasures addressing selective jamming. Our proposal also falls in this category.

In [114], Law *et al.* discuss power-efficient jamming attacks operating at the data-link layer. First, they consider some common Medium Access Control (MAC) protocols for WSNs (i.e. S-MAC, L-MAC and B-MAC), and derive power-efficient jamming attacks based on the estimation of the probability distribution function of inter-packet transmission times. They show that proposed attacks are as effective as constant/deceptive/reactive jamming, but more power-efficient. Then, they propose potential countermeasures against jamming for the considered MAC protocols (e.g. using a high duty cycle in S-MAC, and short data packets in L-MAC). While [114] considers contention-based MAC protocols, we focus on TDMA WSNs.

Wood *et al.* proposed DEE-JAM [113], a new MAC protocol that provides defense against jammers using IEEE 802.15.4-based hardware. DEE-JAM relies on techniques such as frequency hopping, redundant encoding and packet fragmentation, and aims at hiding packets from a jammer node, so evading its search, and limiting the impact of packets that are corrupted anyway. DEE-JAM is compatible with existing nodes' hardware. However, it is a solution specifically tailored to 802.15.4 WSNs. Also, it introduces significant computational and energy costs in resource constrained sensor nodes.

Proano *et al.* analyze a specific selective jamming attack, where the adversary thwarts the transmission of particularly important kinds of packets [11]. They also propose some

methods, based on cryptographic primitives, to mitigate the attack effects. Encryption of transmitted packets is an effective solution against packet classification. However, it requires that the entire packet, including the header, is encrypted. Usually, it is a standard practice to leave the header unencrypted, so that receivers can early abort the reception of packets not destined to them and save energy. In their work, Proano *et al.* consider a jammer that continuously senses and classifies packets, in order to perform selective jamming based on their importance. Instead, we consider a different model of jammer, where the attacker does *not* need to continuously monitor the channel to effectively perform jamming attack.

The closest work to JAMMY is [12], which proposes a countermeasure against the GTS-based selective jamming in IEEE 802.15.4 networks. GTS is basically a form of TDMA communication, where up to 7 *reserved* time slots in different superframes are allocated to sensor nodes by a central *Coordinator*. GTS is extremely vulnerable and prone to selective jamming attacks [10]. In [12], the authors propose a centralized solution where the slot allocation pattern is computed, and randomly changed at each superframe, by the Coordinator node. This reduces the attack effectiveness to at most 1/7. However, the Coordinator node represents a single *point of failure*, i.e., if it fails, the entire network goes out of service. In addition, the above-mentioned solution is tailored to IEEE 802.15.4 and, hence, is not general.

In chapter 8, we propose JAMMY, a novel distributed and self-adaptive solution against selective jamming attacks in TDMA-based WSNs. Like in [12] JAMMY randomly permutes the slot allocation pattern on a superframe basis. By doing so, the slot(s) allocated to a sensor node change(s) unpredictably at each superframe. Hence, the selective jammer is forced to jam slots picked at random in the hope to guess the ones allocated to the victim node. It follows that, assuming that a single slot per sensor node is allocated at each superframe, the probability of a successful selective jamming attack becomes 1/N, where N is the number of slots in a superframe. JAMMY is distributed , in that each sensor node computes the slot to use in the next superframe only using local information in a consistent way, i.e. without causing collisions. Therefore, no centralized slot scheduler is necessary. JAMMY is also self-adaptive as it manages dynamic joining and leaving of multiple nodes. Finally, JAMMY is general in that, although we present it in the WSNs context, it can be in principle adopted in any TDMA systems. We show

that JAMMY is also efficient as it introduces a negligible processing overhead for computing the slot to be used in the next superframe, and also no communication overhead. Furthermore, JAMMY allows new sensor nodes to join the network in a limited time, with consequent benefits in terms of energy consumption in the joining phase.

Chapter 7

LOCALL: a localized slot allocation algorithm for TDMA-based WSNs

In this chapter we present a *localized* slot allocation (LOCALL) algorithm which allows sensor nodes of a TDMA-based WSN to select their own transmission slot(s) in a completely *autonomous* way, just relying on *local* information. This reduces energy consumption and makes the algorithm suitable for environments where packets can be easily corrupted or missed. LOCALL takes an approach similar to the *Collision Detection with Memory* (CDM) algorithm presented in [115]. However, our algorithm uses a different and more efficient strategy for slot selection, which results in a shorter time for completing the slot allocation phase. Altough in the following we only refer to a star network topology LOCALL can be in principle extended to also support multi-hop topologies.

To analyze the performance of LOCALL we develop and solve an analytical model based on a Discrete Time Markov Chain (DTMC). We validate our model through simulation, and evaluate LOCALL in terms of *convergence time* (i.e., time required by the algorithm to achieve a complete TDMA schedule with a pre-defined probability) and *energy consumption*. We show that LOCALL has a lower convergence time with respect to CDM. In addition, the energy consumed by LOCALL to establish a collision-free schedule is quite limited.

The rest of this chapter is organized as follows. Section 7.1 presents the LOCALL algorithm. Section 7.2 describes the analytical model, while section 7.3 discusses the obtained results. Finally, Section 7.4 concludes the chapter.

7.1 LOCALL Algorithm

In this section, we provide a description of our LOCALL algorithm. We consider a star sensor network, where each sensor node has to report data periodically to the sink node (which is assumed to be always active). Without losing in generality, we can assume that the reporting period, T, is fixed and common to all sensor nodes in the network. We also assume that the reporting period is divided into N_s slots, of equal duration, and sensor nodes have to report one data packet per period. Hence, each sensor node requires one dedicated slot per reporting period T.

In order to obtain a collision-free schedule, different sensor nodes must select a different slot in the period. This can be easily achieved if sensor nodes can exploit some centralized information (e.g., an allocation pattern sent by the sink node), or exchange information with other nodes. Instead, LOCALL takes a different approach as it provides a *simple*, *localized,* and *self adaptive* mechanism through which sensor nodes can *autonomously* decide their transmission slot, within the period T.

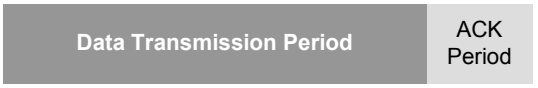


FIGURE 7.1: Slot access pattern during the data reporting phase.

FIGURE 7.2: Slot access pattern during the slot acquisition phase.

As shown in Figure 7.1 the size of a slot is such that it can accommodate the transmission of one data packet and the related acknowledgement (ack). Before using a slot for data transmission, however, a node has to achieve the right to use it. To this end, a preliminary *slot acquisition* phase is carried out where sensor nodes contend for acquiring the *exclusive* use of a slot. At the end of this phase, *each* node has acquired the right to use *one* slot. To implement contention, during the slot acquisition phase slots are

accessed by sensor nodes with a different access pattern, shown in Figure 7.2 (the slot size is the same in both phases). Basically, sensor nodes contending for a generic slot σ, try to transmit a *fake* packet in that slot, using the contention period. If a sensor node wins the contention (i.e., it transmits successfully), it achieves a priority on that slot over all the other sensor nodes, and is authorized to access slot σ in all subsequent periods, *without* contention, to transmit its data packets. A complete collision-free schedule is achieved as soon as all sensor nodes in the network have acquired their own slot.

Ideally, we can assume that a contention is always solved (i.e., there is always one winner) irrespective of the number of competing nodes. Under this assumption, the quickest way to achieve a collision-free schedule is to allow *all* sensor nodes to contend for any slot. As a result of contention for a slot, one node is accommodated in that slot, while the remaining ones will contend for the next slots. Hence, the slot acquisition phase takes just one period.

In practice, the previous (ideal) scheme is unfeasible and we can only approximate it. In our algorithm we use a *random backoff time* to solve contentions, and we assume that it can take a number of discrete values (hence, collisions can still occur). After waiting for the chosen backoff time, a competing sensor node i checks the status of the channel and, if idle, tries to transmit a (fake) packet. Three possible outcomes can occur, namely (**i**) *successful transmission*, (**ii**) *busy channel*, or (**iii**) *collision*. In case of a successful transmission, node i is the winner of the contention and, hence, it acquires the right to use slot σ in all subsequent periods. To get priority over all the other nodes, in the next periods node i will access slot σ with a backoff time equal to 0 (i.e., hereafter, slot σ will be viewed by node i as its own data slot). This also reduces energy consumption and packet latency at node i. If the channel is found busy after the backoff time (case **ii**), it means that one or more sensor nodes have generated a *shorter* backoff time. Thus, node i has to try the next slot (i.e., $\sigma + 1$) in the current period. Finally, when a collision is experienced by node i (case **iii**), it means that one or more other nodes have selected the *same* backoff time for contention in slot σ. In principle, node i could either retry the next slot (i.e., $\sigma + 1$) in the current period, or retry the same slot (i.e., σ in the next period). The rationale behind the latter option is that, if the number of colliding nodes is limited, the contention will be very likely solved at the next period. Another option for a colliding node i, would be re-trying slot $\sigma + 1$ in the current period with probability p_r and defer contention to slot σ in the next period with probability $(1 - p_r)$. Also, this

is the most general case (the previous ones can be derived from it, by using a value of p_r equal to 1 or 0).

Algorithm 1 shows the specific actions performed by a generic sensor node i. Initially, node i selects a random slot σ, within the current period, to try contention. This random choice is aimed at spreading contention trials within the whole period T, thus reducing the number of competitors for each single slot and increasing the success probability. However, this initial randomization very rarely provides a collision-free schedule. Hence, the slot acquisition phase follows, as described above. Specifically, node i contends for slot σ using a random backoff time B, and waits for the corresponding ack message (lines 2-3). Depending on the received notification (line 4), node i either acquires the right to use slot σ (lines 5-6), or realizes that a failure has occurred. In the latter case it behaves in a different way, depending on whether a channel-busy (lines 7-8), or collision (lines 9-11) notification has been received.

1: Choose a slot σ in $[1,\ N_s]$ randomly;
2: Try slot σ (using a random backoff B); // **contention slot**
3: **Wait** (Notification);
4: **Switch** (Notification);
5: **Case** SUCCESS:
6: Use slot σ in all subsequent periods with backoff $B = 0$ // **data slot**
7: **Case** CHANNEL-BUSY:
8: Re-try slot $((\sigma + 1)\%N_s)$ (with random backoff B);
9: **Case** COLLISION:
10: Re-try slot $\sigma + 1$ in the current period (with random backoff B) with probability p_r;
11: Defer contention to slot σ in the next period (with random backoff B) with probability $(1 - p_r)$;

Algorithm 1: LOCALL algorithm

7.1.1 Slot Allocation in IEEE 802.15.4 sensor networks

In the previous description we made no assumption about the sensor platform where LOCALL is supposed to run. Actually, it can be implemented on any sensor platform. However, if the considered sensor platform includes a contention-based MAC protocol (e.g., 802.15.4 MAC [10]), the algorithm can be customized to take advantage of the specific contention mechanisms provided by the underlying MAC layer. Below we briefly describe how LOCALL behaves when it operates on top of the IEEE 802.15.4 MAC layer.

The 802.15.4 MAC protocol [10] implements an un-slotted CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) algorithm, whose behavior can be controlled through a set of parameters. The following parameters are used by LOCALL.

1. *macMaxCSMABackoffs* defines the maximum number of carrier sensing operations (**C**lear **C**hannel **A**ssessment (CCA)), per packet, that can be performed by sensor nodes. It is set to 0, i.e. just one carrier sensing is performed. If the channel is busy a failure notification is received by LOCALL from the underlying MAC.

2. *macMaxFrameRetries* defines the maximum number of packet retransmission attempts allowed for a packet. It is set to 0, so as to force a failure notification if a collision occurs.

3. *macMinBE* defines the initial backoff-window size. Sensor nodes randomly select a backoff time in the range $[0,\ N_B - 1] \cdot BP$, where $N_B = 2^{macMinBE}$, and $BP = 320\,\mu s$. During the slot acquisition phase *macMinBE* is set to a value larger than 0 (e.g., 3). As soon as a success notification is received (i.e., the slot has been acquired), LOCALL sets *macMinBE=0*. This gives priority to that sensor node over all the other sensor nodes in the network.

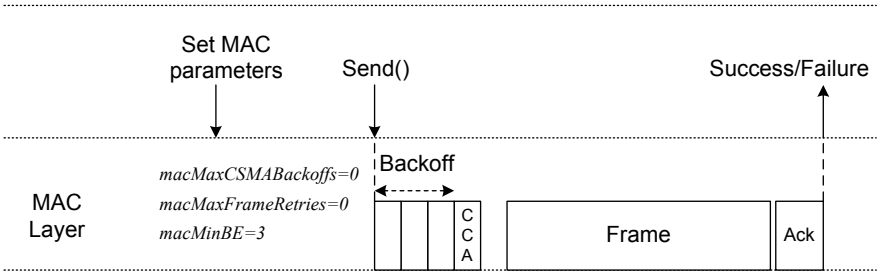


FIGURE 7.3: Interaction between the LOCALL algorithm and the underlying MAC layer.

Figure 7.3 shows the interactions between LOCALL and the underlying 802.15.4 MAC protocol. Protocol parameters are set in advance with respect to packet transmission. At the beginning of a slot, LOCALL makes a *send()* operation, thus activating the underlying MAC protocol. Then, it receives a success/failure notification. A failure may be caused by either a collision or a busy channel condition. The notification message received from the MAC layer contains enough information for LOCALL discriminating between the two events and reacting accordingly (as specified in lines 4-11 of Algorithm 1).

7.2 Analysis

In this section we derive a Discrete-Time Markov-Chain (DTMC) model of the proposed LOCALL algorithm, under the hypothesis that it is operating on top of the 802.15.4 MAC protocol. To simplify the analysis, throughout we will make the following assumptions.

1. *All* sensor nodes start competing for the first slot in a period, i.e., we do not consider the initial randomization (line 1 in Algorithm 1). This maximizes the contention and, hence, we model a "worst-case" condition.
2. The number of contending nodes is equal to the number of available slots, i.e., $N = N_s$ (for the scheduling process to converge it must be $N \leq N_s$).
3. The retry probability p_r (line 10 in Algorithm 1) is assumed equal to 0, i.e., after a collision, *all* colliding sensor nodes defer their transmission to the next period.

TABLE 7.1: Main symbols used in our analysis

Symbol	**Description**
N_B	Number of backoff periods.
BP	Length of a backoff period
$P_{succ}(x)$	Probability of successful transmission, given x contending nodes.
$P_{coll}(c\|x)$	Probability that c nodes collide, given x contending nodes.
$P_{busy}(b\|x)$	Probability that b nodes find the channel busy, given x contending nodes.
$P^A{}_{coll}(c\|x)$	Probability that c nodes collide on a slot already acquired, given x contending nodes.
s_i, n_i	Status of slot i (F/A), Number of nodes that have scheduled a transmission on slot i in the current period
Ω	State space of the Markov chain
$\mathbf{P}$	Transition probability matrix
$\mathbf{v}_k$	State probability vector

Our analysis is split into three parts. In the first part, we consider all the possible events that can occur during the slot acquisition phase and, for each of them, we derive the probability to occur. In the second part, we will use these probabilities to model the slot acquisition process, through a DTMC, and derive the probability distribution of the convergence time. Finally, in the third part, we will derive the average energy consumed by LOCALL during the slot acquisition phase and in steady-state conditions. Table 7.1 summarizes the main symbols used in our analysis.

7.2.1 Event Probabilities

Let us consider a sensor network with N nodes, and assume that, at a given time, M (with $M = \mathrm{N}$) sensor nodes are contending for the same slot. Contention can result in one of the following outcomes.

a Success. One sensor node, out of the M contending ones, successfully transmits in the slot.

b Collision. k $(2 \leq k \leq M)$ sensor nodes experience a collision.

c Busy channel. $h(0 \leq h \leq M-2)$ sensor nodes find the channel busy and must retry at the next slot.

We now analyze the different cases individually and, for each of them, we derive analytically the probability to occur. A *successful* transmission (i.e., case (**a**)) occurs when one sensor node generates a backoff time shorter than that of all the other $M-1$ contending nodes. Let $P_{succ}(M)$ denote the probability that a successful transmission occurs, given that M nodes are contending for the same slot. Since each node can extract a backoff time with a discrete value in the range $[0,\, N_B - 1] \cdot BP$, the following equation holds.

$$P_{succ}\,(M) = M \cdot \sum_{b=0}^{N_B-1} \left[\left(\frac{1}{N_B} \right) \cdot \left(\frac{N_B - 1 - b}{N_B} \right)^{M-1} \right] \tag{7.1}$$

Equation (7.1) can be explained as follows. For each potential backoff time b that can be generated by any sensor node (which can occur with probability $1/N_B$), the second term inside the sum gives the probability that the remaining $M-1$ sensor nodes extract a backoff time larger than b. Finally, all the M possible combinations, corresponding to the different sensor nodes, are considered.

Let us now consider case **(b),** where two or more sensor nodes (i) generate the same backoff time, (ii) start transmitting at the same time and, (iii) experience a collision. Let $P_{coll}(k|M)$ denote the probability that k out of the M contending nodes (with $2 \leq k \leq M$) collide. The following equation holds.

$$P_{coll}(k|M) = \sum_{b=0}^{N_B-1} \left[\begin{pmatrix} M \\ k \end{pmatrix} \cdot \left(\frac{1}{N_B} \right)^{k} \cdot \left(\frac{N_B - 1 - b}{N_B} \right)^{M-k} \right] \tag{7.2}$$

In equation (7.2), for each potential backoff time b, the term inside the sum gives the probability that k (out of M) nodes randomly pick up a value equal to b, and no other node chooses a value shorter than b. Of course, all the $\begin{pmatrix} M \\ k \end{pmatrix}$ possible combinations are considered.

In case **(c)**, h ($0 \leq h \leq M-2$) sensor nodes experience a busy-channel condition. Let $P_{busy}(h|M)$ denote the probability that h nodes find the channel busy during a transmission attempt, given that there are M contending nodes. If h ($0 \leq h \leq M-2$) sensor nodes find the channel busy, it means that the remaining $M-h$ nodes have extracted the same backoff time and have collided. Hence,

$$P_{busy}(h|M) = P_{coll}\,(M-h|M) \tag{7.3}$$

Finally, let us consider another event that may occur after a slot has been assigned. As described above, when a sensor node successfully transmits in a slot, it acquires the right to use that slot in all subsequent periods, and sets *macMinBE*=0. This gives it a priority, for that slot, over all the other nodes. However, collisions can still occur. This is because in the 802.15.4 CSMA algorithm, backoff times range in $[0,\, N_B - 1] \cdot BP$ and, thus, a collision occurs whenever any other node randomly generates a backoff time equal to 0. Let $P^A_{coll}(k|M)$ denote the probability that a collision involving k sensor nodes occurs on a slot already acquired by a node, given that there are M contending nodes, overall. This probability can be expressed as

$$P^A_{coll}(k|M) = \begin{pmatrix} M-1 \\ k-1 \end{pmatrix} \cdot \left(\frac{1}{N_B}\right)^{k-1} \cdot \left(\frac{N_B-1}{N_B}\right)^{M-k} \tag{7.4}$$

In equation (7.4), the second term gives the probability that $k-1$ nodes – in addition to the slot owner – extract a backoff time equal to 0. The third term gives the probability that all the remaining $M-k$ nodes pick up a backoff time larger than 0. Finally, the first term considers all possible combinations.

7.2.2 Convergence Time Distribution

In this section we develop a DTMC model of the LOCALL algorithm and use it to derive the probability distribution of the convergence time, i.e., the time required to achieve a complete schedule. To this end, we will use the event probabilities derived in the previous section.

We observe the system (i.e., sensor network) at the beginning of each period T. Since each period consists of $N_s = N$ slots, the system state at the beginning of period $p(p = 1, 2, ..., n)$ can be represented by a vector $X_p = [(s_1, n_1), (s_2, n_2), ..., (s_N, n_N)]$, whose generic element $X_p[i] = (s_i, n_i)$, $i = 1, 2, ..., N$, refers to the corresponding slot in that period. Specifically, s_i indicates the status of slot i – either *Free* (F) or *Acquired* (A) by a sensor node – and n_i denotes the number of sensor nodes that have scheduled a packet transmission in slot i of period p.

Given the problem constraints, not all vectors in the form defined above represent possible states for the system. The state space Ω can be defined as the set of vectors $X_p = [(s_1, n_1), (s_2, n_2), ..., (s_N, n_N)]$ such that the following conditions hold.

C_1: $\sum_{i=1}^{N} n_i = N$

C_2: $\forall i \in \{1, 2, ..., N\}$, $n_i = 0 \Rightarrow s_i = F$

C_3: $\forall i \in \{1, 2, ..., N\}$, $n_i = 1 \Rightarrow s_i = A$

C_4: $\exists j \in \{1, 2, ..., N\} : n_j > 0 \Rightarrow \forall i < j,\ n_i > 0$

Condition C_1 states that the total number of transmissions scheduled for all the slots, in any period p, cannot exceed the number of sensor nodes. It follows from assuming that each sensor node can transmit, at most, one packet per period. Condition C_2 is almost obvious as well. It states that, if there is no scheduled transmission for slot i $(i = 1, 2, ..., N)$ in period p, then slot i is necessarily free (i.e., $s_i = F$). Instead, Condition C_3 states that, if there is exactly one scheduled transmission for slot i $(i = 1, 2, ..., N)$ in period p, then slot i has been acquired by some sensor node in one of the previous periods (i.e., its state must be $s_i = A$). This statement follows from the definition of *acquired* slot. Condition $n_i = 1$ implies that one sensor node has successfully transmitted on slot i in one of the previous periods. Hence, the slot has been acquired. Finally, condition C_4 states that if there is a slot j $(j = 1, 2, ..., N)$ for

which the number of scheduled transmissions is larger than zero, then all previous slots must have, at least, one scheduled transmission. This follows from assuming that sensor nodes start contending for the first slot and, then, try all the other slots in sequence. As an example, Figure 7.4 shows all possible states of the Markov chain when the number of sensor nodes (and available slots) is equal to three. It can be easily verified that all states satisfy conditions C_1 –C_4.

Now, we need to derive the transition probabilities for the Markov chain, i.e., the probability of passing from a state $\boldsymbol{X}$ to another state $\boldsymbol{Y}$, for any $\boldsymbol{X}$ and $\boldsymbol{Y}$ in Ω. As a preliminary step, let us focus on a generic slot i, with $i = 1, 2, ..., N$. Let $X[i] = (s_i, n_i)$ and $Y[i] = (s'_i, n'_i)$ denote the condition of slot i at period p and $p+1$, respectively. Then, the probability of passing from $X[i] = (s_i, n_i)$ to $Y[i] = (s'_i, n'_i)$ can be expressed as follows.

$$P(Y[i]\,|X[i]) =$$

$$= \begin{cases} 1 & if\ n'_i = n_i = 0 & (T_1) \\ 1 & if\ n'_i = n_i = 1 & (T_2) \\ P_{succ}(n_i) & if\ (s'_i = A) \wedge (n'_i = 1) \wedge (s_i = F) \wedge (n_i > 1) & (T_3) \\ [(N_B - 1)/N_B]^{n_i - 1} & if\ (s'_i = A) \wedge (n'_i = 1) \wedge (s_i = A) \wedge (n_i > 1) & (T_4) \\ P_{busy}(n_i - n'_i | n_i) & if\ (s'_i = F) \wedge (1 < n'_i \leq n_i) \wedge (s_i = F) & (T_5) \\ P^A_{coll}(n'_i | n_i) & if\ (s'_i = A) \wedge (1 < n'_i \leq n_i) \wedge (s_i = A) & (T_6) \\ 0 & otherwise & (T_7) \end{cases}$$

The previous probabilities can be justified as follows.

In transition T_1 we assume $n_i = 0$, i.e., there are no sensor nodes contending for slot i in period p. Hence, the status of that slot at period $p+1$ will be unchanged. Similarly, in transition T_2 it is supposed that $n'_i = n_i = 1$. Since there is just one contending node, $P(Y[i]\,|X[i]) = 1$ in this case. In transition T_3, we suppose to start with $n_i > 1$ contending sensor nodes (and $s_i = F$). For the final state being $(A,\ 1)$ – meaning that the slot has been acquired by a sensor node – a successful transmission must occur during period p. Thus, the corresponding probability is $P_{succ}(n_i)$. Similarly, in transition T_4, we assume that there are $n_i > 1$ contending sensor nodes but, now, $s_i = A$. The final state $(s'_i = A, n'_i = 1)$ can occur only when all nodes – but the one that has already acquired the slot (and, hence, uses a null backoff time) – pick up a random

backoff time larger than zero. The associated probability is thus $[(N_B - 1)/N_B]^{n_i - 1}$. In transition T_5 we suppose to start from state $(s_i = F, n_i > 1)$ but, now, we move to state $(s_i = F, n_i' > 1)$. This can occur when $n_i - n_i'$ (with $n_i - n_i' \geq 0$) find the channel busy during period p. Hence, the corresponding probability is $P_{busy}(n_i - n_i' | n_i)$. Finally, in transition T_6 it is assumed that the initial state is $(s_i = A, n_i > 1)$, while the final state is $(s_i = A, n_i' > 1)$. This transition occurs when n_i' sensor nodes experience a collision during period p. According to the notation introduced in section 7.2.1, the corresponding probability is $P_{coll}^{A}(n_i' | n_i)$.

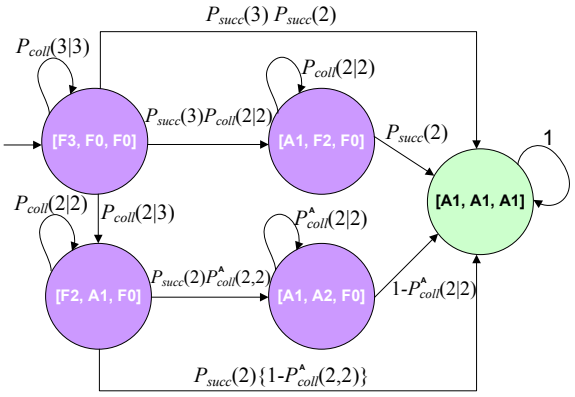


FIGURE 7.4: Markov Chain when $N = N_s = 3$.

We are now in the position to derive the transition probability P_{XY} for any $\boldsymbol{X}$ and $\boldsymbol{Y}$ in Ω, starting from $P(Y[i]|X[i])$, $i = 1, 2, ..., N$.. To this end, let us observe that the transition probability P_{XY} is given by the joint probability that each slot i $(i = 1, 2, ..., N)$ passes from $X[i] = (s_i, n_i)$ to $Y[i] = (s_i', n_i')$. Hence,

$$P_{XY} = P(Y[1]|X[1]) \cdot \prod_{i=2}^{N} P\{Y[i] \,|\, [s_i, n_i + n_i^r]\} \tag{7.5}$$

where $n_i^r = \sum_{j=1}^{i-1}(n_j - n_j')$. In equation (7.5), for slots with index $i \geq 2$, the term inside the product is not $P[Y[i] \,|\, (s_i, n_i)]$, but $P\{Y[i] \,|\, [s_i, n_i + n_i^r]\}$ because we need also to take into consideration possible nodes that experience a busy channel at slot $i - 1$ and, thus, retry at slot i. Figure 7.4 shows the transition probabilities for the simple case when $N = N_s = 3$.

Once we have derived the transition probability P_{XY} for any $\mathbf{X}$ and $\mathbf{Y}$ in Ω, we can obtain the transition probability matrix $\mathbf{P}$ of the Markov chain. To this end, let us sort the states of the Markov chain in such a way that the initial state $I =$

$[(F,N),(F,0),...,(F,0)]$ and the absorbing state $Z = [(A,1),(A,1),...,(A,1)]$ are, respectively, the first and last state in the sequence. Let v_0 denote the initial probability vector, and v_k the vector probability after k periods ($k = 1, 2, ...$). Without losing in generality we can assume $v_0 = [1, 0, 0, ..., 0]$. Hence, $v_k = v_0 P^k$. The probability that the slot allocation process is over after k periods, $P_{schedule}(k)$, corresponds to the probability of being in state Z at step k. Let $|\Omega|$ denote the cardinality of Ω (i.e., the total number of states). Since Z is the last state in the sequence (according to the selected order), it follows

$$P_{schedule}(k) = v_k[|\Omega|] \tag{7.6}$$

7.2.3 Energy Consumption Analysis

In this section we derive formulas for computing the average energy consumed by the network both in a single period and during the entire slot acquisition phase. In the following we will assume that (**i**) the power consumption of sensor nodes in idle state is negligible, i.e., $P_{idle} = 0$; and (**ii**) the power consumption during the turnaround time is $P_{tat} = (P_{rx} + P_{tx})/2$, where P_{rx} (P_{tx}) is the power consumed in receive (transmit) mode. As a preliminary step, we first derive the energy E_{XY} consumed by the network to pass from state **X** to state **Y**.

Let us assume there are M nodes ($M = N$), all contending for the same slot. The total energy consumed by all nodes depends on the specific outcome following the contention. If one of the M nodes succeeds in transmitting its packet, the energy consumption $E_{succ}(M)$ can be calculated as follows

$$E_{succ}(M) = M \cdot P_{rx} D_{cca} + 2 \cdot P_{tat} D_{tat} + P_{tx} D_{tx} + P_{rx} D_{ack} \tag{7.7}$$

where $D_{cca}, D_{tat}, D_{tx}, D_{ack}$ denote the duration of CCA, turnaround time, packet transmission time, and ack reception time, respectively. The first term in Equation (7.7) accounts for the energy consumed by all the M nodes to perform their CCA, while the other terms account for the additional energy consumed by the winner node (all the other nodes find the channel busy and give up). The latter energy is spent for switching the radio from receive to transmit mode ($D_{tat} \cdot P_{tat}$), transmitting the packet ($D_{tx} \cdot P_{tx}$), switching again to receive mode ($D_{tat} \cdot P_{tat}$), and receiving the ACK ($D_{ack} \cdot P_{rx}$). Following the same line of reasoning, the total energy $E_{coll}(k|M)$ consumed when k out of

the M competing nodes experience a collision, with $2 \leq k \leq M$, can be expressed as follows

$$E_{coll}(k|M) = M \cdot P_{rx} D_{cca} + k \cdot [2P_{tat} D_{tat} + P_{tx} D_{tx} + P_{rx} D_{to}]$$

where D_{to} is the timeout interval.

We can now calculate the energy consumed by the network when the state of a certain slot i $(i = 1, 2, ..., N)$ passes from $X[i] = (s_i, n_i)$ to $Y[i] = (s'_i, n'_i)$, i.e., $E(Y[i]|X[i])$.

$$E(Y[i]|X[i]) = \begin{cases} E_{succ}(n_i) & n'_i = 1 \wedge n'_i \leq n_i \\ E_{coll}(k|n_i) & n'_i = k \wedge n'_i \leq n_i \\ 0 & otherwise \end{cases} \tag{7.8}$$

Equation (7.8) can be justified as follows. In the first case, we assume to start with $n_i \geq 1$ competing nodes, one of which will be the winner ($n'_i = 1$). Hence, the energy consumption is $E_{succ}(n_i)$. In the second case, there are $n_i \geq n'_i$ nodes in the initial state and k $(2 \leq k \leq M)$ nodes in the final state. Hence, k out of the n_i nodes experience a collision, and the total energy consumption is given by $E_{coll}(k|n_i)$. In all other cases, since there are no nodes accessing the slot, or the transition from $X[i]$ to $Y[i]$ is not possible, the related energy consumption is 0.

The total energy E_{XY} consumed by the network to pass from state $\mathbf{X}$ to state $\mathbf{Y}$ can be derived by considering the energy spent when the state of slot i passes from $X[i] = (s_i, n_i)$ to $Y[i] = (s'_i, n'_i)$, for any slot $i = 1, 2, ..., N$, i.e.,

$$E_{XY} = E(Y[1]|X[1]) + \sum_{i=2}^{N} E\{Y[i] \,|\, [s_i, n_i + n_i^r]\} \tag{7.9}$$

where $n_i^r = \sum_{j=1}^{i-1}(n_j - n'_j)$. In Equation (7.9), for slots with index $i \geq 2$, the term inside the sum is not $E[Y[i] \,|\, (s_i, n_i)]$, but $E\{Y[i] \,|\, [s_i, n_i + n_i^r]\}$. This accounts for nodes that found the channel busy at slot $i - 1$ and, thus, retry at slot i.

We are now in the position to derive the average energy $\overline{E_k}$ consumed by the network during a period k $(k = 1, 2, ...)$ for transmitting either a contention packet (during the slot acquisition phase) or a data packet (after acquiring a slot). Let P_X^k denote the probability that the system is in state $\mathbf{X}$ at period k for any $X \in \Omega$ (P_X^k is the component

of v_k associated with state $\mathbf{X}$). The following equation holds.

$$\overline{E_k} = \sum_{X \in \Omega} P_X^{k-1} \sum_{Y \in \Omega} E_{XY} P_{XY} \tag{7.10}$$

Equation (7.10) can be justified as follows. To calculate $\overline{E_k}$ we need to consider all the possible state changes that can occur from period $k-1$ to period k. Hence, the outer sum considers any possible state $X \in \Omega$ where the system can be at period $k-1$, which occurs with probability P_X^{k-1}. For each state $X \in \Omega$ the inner sum considers all the possible states $Y \in \Omega$ where the system can transit to – which occurs with probability P_{XY} – and the energy consumption E_{XY} associated with such a transition.

Finally, we derive the average energy spent by the network to achieve a complete schedule, i.e., the total amount of energy consumed by all nodes to contend and acquire slots. This measures the energy overhead due to slot scheduling. A complete schedule is achieved when the system enters the absorbing state $Z = [(A,1),(A,1),...,(A,1)]$. Hence, according to the methodology in [116], the average energy μ_X consumed by the network to reach state Z, starting from any state $X \in \Omega$, can be obtained by solving the following system of linear equations with μ_X as unknowns

$$\mu_X = \sum_{Y \in \Omega} P_{XY}(E'_{XY} + \mu_Y), \forall X \in \Omega \tag{7.11}$$

with $\mu_Z = 0$. E'_{XY} differs fromE_{XY} in that it only considers transitions associated with the transmission of fake packets. Since we always start from the initial state I we only need to calculate μ_I.

7.3 Results

In this section we show the results obtained from the analytical model derived in the previous section. To validate our analytical results, as well as for comparing the performance of LOCALL with that of the CDM algorithm [115], we also used simulation. CDM takes a lightweight vertex-coloring approach. Specifically, sensor nodes are supposed to choose a different color from a set of available colors (in our terminology, colors correspond to slots). Initially, all sensor nodes are in *search* mode. At each round (period) p, a generic node v (**i**) picks up randomly a color from the set of available colors,

and (**ii**) checks whether it has a conflict with any other node. If there is no conflict, node v enters *permanent* mode, selects c as its permanent color, and stops. Otherwise, it waits for a new round and performs the same actions again. The algorithm ends when *all* sensor nodes are in permanent mode.

We implemented both LOCALL and CDM using the ns-2 simulation tool [54]. We considered a single-hop network, where sensor nodes are located at a fixed distance (10 m) from the sink node. The transmission range was set to 15 m, while the carrier sensing range was set to 30 m. Unless stated otherwise, the other parameter values are as shown in Table 7.2 (power consumptions were derived from the CC2420 data sheet [55]). In each experiment, we performed ten independent replications, each of which consisted of 500 different slot acquisition processes. The results shown below are averaged over all the replications. We also derived confidence intervals by using the *independent replication* method and 99% confidence level.

TABLE 7.2: Parameters used in our analysis

Parameter	**Value**
N, N_s	10
Bit Rate	250 Kbps
Data Frame (Payload) Size	127 bytes
ACK Frame Size	11 bytes
N_B(slot acquisition phase)	8
Power Consumption in RX mode (P_{rx})	35.46 mW
Power Consumption in TX mode (P_{tx})	31.32 mW
Power Consumption in Idle mode (P_{idle})	0 mW

Figure 7.5 shows the probability distribution of the convergence time, for different number of sensor nodes, derived both from analysis (i.e., using Equation 7.6) and simulation. In order to validate the analytical model, the initial randomization performed by the algorithm has not been considered in simulations. As it can be observed, analytical and simulation results almost overlap.

We also compared our algorithm with CDM in terms of convergence time. Since an analytical model for CDM is not available, we used simulation for comparison (for LOCALL we also considered the initial randomization). Table 7.3 reports the 95-th percentile of the convergence time distribution, i.e., the number of periods required to provide a complete schedule with a probability of, at least, 0.95. Our algorithm converges in a significantly shorter time due to its more efficient slot (color) selection strategy. In CDM,

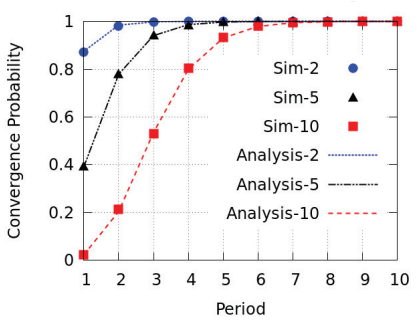


FIGURE 7.5: Convergence Time of LOCALL for different number of nodes

TABLE 7.3: 95% convergence Time (# of Periods).

# of nodes	**LOCALL Simulation**	**CDM Simulation**
2	2.00 (±0.00)	4.8 (±0.34)
5	3.80 (±0.43)	16.3 (±0.77)
10	5.10 (±0.32)	34.3 (±1.59)
20	8.00 (±0.41)	71.1 (±2.53)
30	10.50 (±0.54)	113.1 (±5.92)
40	12.70 (±0.50)	150.4 (±7.51)
50	14.80 (±0.43)	178.1(±9.63)

at each round (period) any sensor node selects a color (slot) at random, and checks for possible conflicts with other nodes. If a conflict is detected, the node tries a new color (slot) at random in the next round. Hence, a complete schedule is reached only after a number of rounds, which increases dramatically with the number of sensor nodes. Conversely, in LOCALL the contention is performed sequentially, for each single slot (color). The initial randomization reduces the number of potential competitors for each slot. Then, if a conflict is detected, the node tries the next slot in the same period, or the same slot in the next period (round). Table 7.3 shows that, for a sensor network with 50 nodes, with LOCALL a complete schedule is reached in about 15 periods (with a probability of 0.95). It should be emphasized that this is the time taken to obtain a *complete* schedule. However, individual sensor nodes may achieve their own slot in considerably less time.

We also investigated analytically the impact of the Contention Period (i.e., the N_B value) on the convergence time. We found that, as expected, increasing the Contention Period reduces the convergence time. However, to avoid undetected collisions (during slot allocation phase) the Contention Period must be shorter than the Transmission Period (see Figure 7.2). For 802.15.4, it can be shown that the maximum allowed value

for N_B is 8.

Figure 7.6 shows the average energy consumed by the network in each period (derived from Eq. 7.10), when there are 10 nodes. As above, analytical and simulation results overlap, when there is no initial randomization. The latter reduces the average energy consumption as it drastically reduces the number of competitors per slot. The consumption in steady-state conditions is the same in both cases.

Finally, we computed (through Eq. 7.11), the total average energy consumed by the network due to slot allocation (i.e., without considering the transmission of data packets). The results are shown in Table 7.4 which also includes simulation results. With 10 nodes, the energy overhead due to slot allocation is 3.32 mJ, which corresponds to the total energy consumed by the network during the data transmission phase in 2.2 periods (1.5 periods with initial randomization).

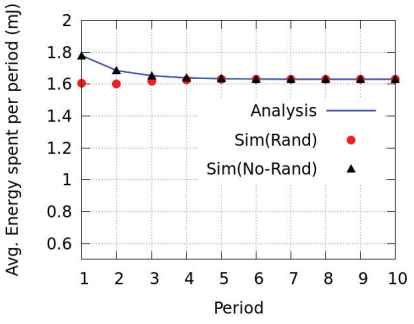


FIGURE 7.6: Average energy consumed by the sensor network when $N = 10$.

TABLE 7.4: Avg. Energy spent for slot scheduling(mJ)

# of nodes	**Model**	**Simulation without Rand.**	**Simulation with Rand.**
2	0.38	0.38 (±0.01)	0.38 (±0.00)
5	1.21	1.21 (±0.01)	1.02 (±0.01)
10	3.32	3.32 (±0.03)	(±0.02)

7.4 Conclusions

In this chapter we have proposed a slot allocation algorithm for WSNs. Unlike many previous decentralized solutions, the proposed algorithm is localized, i.e., sensor nodes select their slot basing on local information only. We have developed an analytical model of the proposed algorithm, based on a Discrete Time Markov Chain, and derived the

probability distribution of the *convergence time* (i.e., the time to obtain a complete schedule) and the average *energy consumption*. Our results show that (**i**) the proposed algorithm is able to converge much faster than another similar algorithm used for comparison, (**ii**) the schedule time increases with the number of nodes with a slope less than linear, (**iii**) the energy overhead due to slot scheduling is very limited.

Chapter 8

JAMMY: a distributed and self-adaptive solution against selective jamming attack in TDMA-based WSNs

8.1 Introduction

TDMA-based WSNs suffer from *selective jamming* attack, a particularly insidious form of Denial-of-Service (DoS) that allows an adversary to completely thwart the communication of a victim node with a very low probability to be detected. In TDMA-based WSNs, a sensor node typically retains its allocated slot for a large number of consecutive superframes. Therefore, an adversary could simply monitor communication, detect the slot allocated to the victim node, and jam only that slot, feigning, for example, a collision. Such an attack is very effective, energy efficient, and much more difficult to be detected than a traditional wide-band jamming. Although several methods are available for jamming detection [105], their application to selective jamming is greatly complicated by the limited exposure time of the adversary and the limited amount of traffic that is affected by the attack.

In this chapter, we propose JAMMY, a novel distributed and self-adaptive solution against selective jamming attacks in TDMA-based WSNs. The proposed methodology is

based on a key idea, i.e. randomly permuting the slot allocation pattern on a superframe basis. By doing so, the slot(s) allocated to a sensor node change(s) *unpredictably* at each superframe. Hence, the selective jammer is forced to jam slots picked at random in the hope to guess the ones allocated to the victim node. It follows that, assuming that a single slot per sensor node is allocated at each superframe, the probability of a successful selective jamming attack becomes $1/N$, where N is the number of slots in a superframe. JAMMY is *distributed*, in that each sensor node computes the slot to use in the next superframe only using *local* information in a *consistent* way, i.e. without causing collisions. Therefore, no centralized slot scheduler is necessary. JAMMY is also *self-adaptive* as it manages dynamic joining and leaving of multiple nodes. Finally, JAMMY is *general* in that, although we present it in the WSNs context, it can be in principle adopted in any TDMA systems.

We carried out a performance analysis of our proposed solution, both in steady-state and dynamic conditions. Our results show that JAMMY is also *efficient* as it introduces a negligible processing overhead for computing the slot to be used in the next superframe, and also no communication overhead. Furthermore, JAMMY allows new sensor nodes to join the network in a limited time, with consequent benefits in terms of energy consumption in the joining phase.

The rest of the chapter is organized as follows. Section 8.2 introduces the system model and the attack model. Sections 8.3 and 8.4 introduce JAMMY in two steps. First, Section 8.3 assumes that the network is in steady state conditions, i.e. no sensor node joins or leaves, and describes how sensor nodes compute the next slot allocation pattern in a distributed way. Then, Section 8.4 describes how JAMMY manages joining and leaving of sensor nodes. Section 8.5 compares the performance of JAMMY with that of a generic centralized solution. Section 8.6 analyzes performance of the join protocol used in JAMMY, by means of a Discrete Time Markov Chain (DTMC). The results of the analysis are presented in Section 8.7. Finally, Section 8.8 draws some conclusions.

8.2 System and adversary model

In this section, we introduce the system and adversary model and describe the selective jamming attack.

We consider a WSN where sensor nodes access the wireless medium using a Time Division Multiple Access (TDMA) method. This means that time is divided into periodic *superframes* of equal duration, each one of which is in turn composed by a number of N equally-sized transmission *slots*. Slots are allocated to sensor nodes for transmitting/receiving their data packets. Specifically, each sensor node remains active only during its allocated slots and sleeps in the remaining time. For the sake of simplicity, and without losing in generality, we assume that a sensor node may have at most one allocated slot in any superframe (*Uniqueness Property*). Also, every slot in the superframe can be allocated to at most one sensor node (*Exclusiveness Property*).

The considered WSN has a dynamic membership, i.e. sensor nodes may join and leave dynamically. Specifically, a sensor node joins the WSN when its mission starts, and leaves it when its mission terminates. Upon joining the WSN, a sensor node runs a *decentralized slot acquisition algorithm* in order to determine a free slot, and acquire it for subsequent communications. We do not commit to any specific slot acquisition algorithm, provided that the adopted one satisfies the Uniqueness and Exclusiveness properties defined above. In a traditional approach, a node retains the allocated slot until it leaves the network. Upon leaving, the sensor node releases the slot, which becomes available to possible joining nodes.

In the rest of this chapter, we consider an external adversary whose objective is to disrupt all communications of a victim node, by performing a *selective jamming* (SJ) attack, i.e. by maliciously transmitting during the victim's transmission slot. We assume that the adversary does not compromise any sensor node, neither physically nor logically, but she is able to eavesdrop and jam any communication within the WSN. In addition, while performing the attack, the adversary is willing to save as much energy as possible and to be as much invisible as possible, to limit the likelihood of being detected.

Specifically, the considered adversary can easily succeeds in playing the SJ attack under the aforementioned constraints as follows. First, she monitors communications for one or more superframes, and identifies the slot allocated to the victim node. The specific approach that the adversary adopts to perform this task is not important here. Just to fix ideas, the adversary may exploit some prior knowledge such as the victim's identifier, its position in the network, or the type of traffic it produces. Then, starting from the next superframe, the adversary systematically jams that slot. Since the victim uses the

same slot in all superframes, the attack is 100% *effective*. Besides, it is also *power-efficient*, as the adversary has to jam only one slot per superframe, while can turn off her radio during all other slots. Finally, the attack is practically *undetectable*, as it exposes the adversary for a very limited amount of time (one slot per superframe).

8.3 The JAMMY algorithm

The SJ attack described above is based on the observation that, in a traditional TDMA approach, the same slot is allocated to a sensor node until the latter leaves the system. Hence, one way to contrast the attack is to change the slot allocation pattern at every superframe making it *unpredictable* for the adversary. This means that the adversary must not be able to predict the slot to be used by the sensor node in the next superframe, even after observing a number of superframes. Thus, the only strategy available to the adversary is to randomly pick a slot and jam it. It follows that the attack effectiveness decreases to $1/N$.

In order to achieve such a goal, at the end of every superframe we can compute the next allocation pattern as a *random permutation* of the current one. However, *unpredictability* is not sufficient. We also require that sensor nodes are able to compute the next allocation pattern *autonomously*, i.e. relying only on local information. Of course, the computation of slot allocation pattern must be *consistent*. That is, all nodes in the system must autonomously compute the *same* permutation or, otherwise, collisions would ensue and the Exclusiveness property would not be guaranteed anymore.

In order to fulfill such requirements, we assume that every node executes a *random permutation algorithm*. At the end of each superframe, every node randomly permutes the current slot allocation pattern, thus producing the next one. Typically, a random permutation algorithm uses a random number generator. In order to prevent collisions, nodes must compute the same permutation, and thus have to produce the *same* sequence of random numbers. It follows that nodes must use *pseudo-random number generators*, which must be maintained in the same internal state. This also implies that, when a new node joins the network, its generator must be initialized into the same internal state as

the one of the nodes already present in the network. Besides, to fulfill the unpredictability requirement, the sequence of psedo-random numbers must also be unpredictable, and therefore the pseudo-random number generator must be secure [117].

In the following, we present our countermeasure against selective jamming in detail. Specifically, in Section 8.3.1, we introduce the random permutation algorithm and the secure pseudo-random number generator (SPRNG) we have used in JAMMY. Then, we make the simplifying assumption that, after system initialization, the WSN membership remains in a steady state condition, i.e. no sensor nodes join or leave (Section 8.3.2). Finally, we remove such an assumption, and show that sensor nodes may join and leave at any time without jeopardizing the solution (Section 8.4).

8.3.1 On implementing a random permutation

In this section, we introduce the two basic components of our countermeasure against SJ, namely a random permutation algorithm and a Secure Pseudo-Random Number Generator (SPRNG). While the literature provides many instances of such algorithms, we need to design two of them which are affordable on resource constrained sensor nodes. As to the random permutation algorithm, we have used the Fisher-Yates algorithm [118], also known as the Knuth shuffle algorithm (Algorithm 2). It takes a vector of n elements as input, and randomly permutes their values. That is, it swaps the value of the element in position i, $0 \leq i \leq n-1$, and the value of the element in position j, where j is picked at random between 0 and $i-1$. The algorithm runs in $\mathcal{O}(n)$ time.

```
1.  /* Returns a random unsigned integer */
2.  unsigned random(unsigned m);
3.
4.  void permute(unsigned perm[], unsigned n)
5.  {
6.    unsigned i;
7.    for (i = n - 1; i >= 0; i--) {
8.      unsigned j = random() % (i + 1);
9.      unsigned aux = perm[j];
10.     perm[j] = perm[i];
11.     perm[i] = aux;
12.   }
13. }
```

Algorithm 2: The Knuth shuffle.

We have implemented the SPRNG by means of a block cipher in the counter mode (Algorithm 3) [117]. Let $E(x, y)$ denote a cipher which encrypts a plaintext y by means of a key x. First, we provide the generator with an encryption key K, and initialize a

counter z to a random seed z_0. Then, we apply the cipher to the sequence of values $z, z+1, z+2, \ldots$ so producing the output random sequence $E(K, z), E(K, z+1), E(K, z+2), \ldots$. In the following, we call counter z the *internal state* of the generator, and K the *permutation key*. Also, we assume that K is kept secret and its length discourages an exhaustive key search.

Although such a method has not been proven to be cryptographically secure, it appears sufficient for most applications [117]. Furthermore, it is affordable on resource-constrained sensor nodes. Actually, commercially-available sensor nodes platforms such as Tmote Sky [58] provide AES-128 encryption in hardware, with negligible overhead in terms of delay, storage, and energy consumption [119]. An alternative optimized version of the *permute()* function is reported in Appendix.

```
1.  unsigned  K; // permutation key
2.  unsigned  z; // counter
3.  unsigned  random()
4.  {
5.    unsigned val = E(K, z);
6.    z = (z + 1);
7.    return val;
8. }
```

Algorithm 3: Secure Pseudo-Random Number Generator.

8.3.2 The Secure Slot Permutation Algorithm

We are now in the position to describe the Secure Slot Permutation (SSP) algorithm used by JAMMY to protect communications against SJ attack. In this section, we assume that, after system initialization, the system membership remains in steady state condition, i.e. no sensor nodes join or leave.

Also, we assume that every sensor node maintains a *permutation vector*, namely a vector of N unsigned elements, which represents the node's view of the current slot allocation pattern. We denote by v_u the permutation vector of node u. Every node maintains its own permutation vector as follows. First of all, $v_u[i] = 1$ iff slot i has been allocated to node u. Otherwise, $v_u[i] = 0$. Also, the Uniqueness property implies that the permutation vector v_u of a node u contains just one element that is equal to 1, whereas all other elements are equal to 0. More formally, $\forall\ 0 \leq i, j < N, v_u[i] = v_u[j] = 1 \Leftrightarrow i = j$. Finally, the Exclusiveness Property implies that a given element i may be equal to 1 in at most one permutation vector. More formally, for any pair of nodes (t, u), then

$v_t[i] = v_u[i] = 1 \Leftrightarrow t = u$. Notice that element i can be zero in every permutation vector iff slot i has not been allocated to any node.

Let us assume that the network initially contains N_A nodes with $N_A < N$. Also, let us assume that nodes have been initialized via off-line methods so that they all secretly share the same permutation key K, and their on-board SPRNGs are all initialized in the same initial state z_0. Finally, an initial slot allocation pattern satisfying the Uniqueness and Exclusiveness properties has been defined and permutation vectors have been initialized accordingly. Without any loss of generality, we may assume that this initial allocation has been defined off-line. Alternatively, it may have been produced by the decentralized slot acquistion algorithm.

1: permute(v, N)
2: Find i s.t. $v[i] = 1$
3: **return** i

Algorithm 4: Secure Slot Permutation.

Since from the initialization, each node protects itself from the selective jamming attack by *periodically* performing the *Secure Slot Permutation* (SSP) algorithm (Algorithm 4) at the end of every superframe. The SSP algorithm takes a permutation vector as input argument, (pseudo-) randomly permutes it (line 1), finds the position i of the unique element with value 1 (line 2), and returns i as the index of the slot to be used in the next superframe (line 3).

To fix ideas, let us focus on the first execution instance of the SSP algorithm at the end of the first superframe. When this superframe ends, every node executes the SSP algorithm passing its permutation vector as input. As all nodes share the same permutation key K and have the SPRNG in the same state (z_0) then they compute the *same* permutation (Algorithm 2), so meeting the requirement of consistency. Since the permutation is based on a SPRNG, then it results unpredictable for an adversary who does not know the permutation key. Finally, the SSP algorithm operates only on local data and, thus, each sensor node can autonomously compute the permutation.

It is worth noting that an adversary could still completely jam the network, i.e. perform a wide-band jamming. Alternatively, she could continuously monitor the network in order to detect the new slot allocated to the victim node, and then jam it. However, by doing so, she would compromise her undetectability and power efficiency. Specifically, a wide-band jamming would make the adversary easily detectable. Furthermore, wide-band

jamming and continuous monitoring would increase the adversary power consumption, making the attack inconvenient from the energy point of view.

The SSP algorithm maintains the Uniqueness Property as it simply permutes the permutation vector elements. It follows that only one element of the resulting permutation vector contains the value 1, while all others have value 0. Also, SSP maintains the Exclusiveness Property, because a random permutation is a bijective function and thus it is injective (or one-to-one). In other words let v_u and v_t be the permutation vectors of any pair of nodes u and t, respectively. Of course, $v_u \neq v_t$ by virtue of the Exclusiveness Property. Furthermore, let Π be the same permutation that u and t compute at the end of the superframe. Let v'_u and v'_t be the resulting permutation vectors after that permutation Π has been applied to v_u and v_t, respectively. That is, $v'_u \leftarrow \Pi(v_u), v'_t \leftarrow \Pi(v_t)$. Then, by virtue of the injective property, it must be $v'_u \neq v'_t$, and hence $v_u[i] = 1 \ \wedge \ v_t[i] = 1 \implies u \equiv t$. A formal proof is reported in Appendix.

Finally, every execution instance of Algorithm 2 causes the counter of the SPRNG to be incremented by N. As all nodes compute the permutation at the end of the superframe, then, at the end of the first superframe, all SPRNGs are still in the same state, namely $z = z_0 + N$. It follows that, in the next execution of the SSP algorithm, which takes place at the end of the second superframe, nodes will compute the same permutation once again, and take their generators into the same next internal state. This reasoning can be iterated for any subsequent superframe, i.e., after r superframes, the internal state of the SPRNG will be $z = z_0 + r \times N$.

As it turns out, the value of the counter of a SPRNG grows at a speed that is equal to the number of slots N in a superframe. It is worthwhile to notice that the size of the counter of the SPRNG establishes an upper bound to the maximum length of the random output sequence the generator is able to produce. Therefore, the counter size must be adequately large to avoid the counter to wrap around during the lifetime of the network (e.g. 64–128 bits). However, one way to deal with the counter wrap around is to refresh the permutation key and re-initialize generators once again. Since the internal states of all SPRNGs remain synchronized over time, the counter wrap-around occurs at the same superframe on all sensor nodes. Hence, at that point in time, all sensor nodes can simultaneously and autonomously generate a new permutation key K^+ as

$K^+ = E(K, K)$. Hereafter, all sensor nodes rely on K^+ until the next counter wrap around occurs.

8.4 Node leave/join

So far we have assumed that the network is in steady state conditions, i.e. the number of sensor nodes is constant over time. In this section, we describe how JAMMY behaves when sensor nodes join and leave the network.

In case one or more sensor nodes leave the network, the behavior of the remaining nodes is not affected at all. Nodes leaving the network just release their own slot, which becomes free, while the remaining sensor nodes continue with their normal operations. However, in order to assure and maintain network security, it is necessary to provide a new permutation key K to the remaining sensor nodes, by excluding and, thus, logically evicting the leaving ones. This problem is commonly known as *rekeying*, and has been widely discussed in literature [120][121][122][123]. Although we included it in our solution, this issue is beyond the scope of this work, and we do not further discuss it.

JAMMY allows sensor nodes to join the network at any time. In addition, multiple sensor nodes can join the network at the same time. Sensor nodes execute a specific *join procedure* (described in Section 8.4.2) in order to correctly join the network. As described later, the *join procedure* assumes that every joining node performs a *Slot Acquisition* algorithm to find and exclusively acquire a free TDMA slot in the superframe. In principle, any decentralized slot scheduling algorithm could be used to this end. For our purposes, we assume that joining sensor nodes use the Slot Acquisition algorithm described in Section 8.4.1.

8.4.1 Slot Acquisition algorithm

We suppose that joining nodes use an approach similar to the one presented in Chapter 7, i.e. a *simple*, *adaptive* and *distributed* strategy through which sensor nodes autonomously find a free slot in the superframe. We consider the general case when N_j sensor nodes try to join the network simultaneously, and assume that $N_j \leq N_F$ where $N_F = N - N_A$ is the number of free slots in the superframe. According to our algorithm,

sensor nodes compete on free slots to acquire the exclusive privilege of using one slot in all superframes. Specifically, joining nodes contending for a generic slot s, try to transmit a *fake* packet during such a slot. In case a joining node wins the contention, i.e. it transmits successfully, it acquires the right to use that slot in all subsequent superframes. Once a joining node has acquired a slot, it stops the Slot Acquisition process, and completes the join procedure (see Section 8.4.2).

The Slot Acquisition algorithm relies on a *random backoff time* to prevent collisions among competing sensor nodes accessing the same slot s. After waiting for the selected backoff time, a joining node u checks the channel status, and, if found idle, tries to transmit a (fake) packet. Then, three possible outcomes can occur, namely **i**) *successful transmission*; **ii**) *busy channel*; or **iii**) *collision*. In case of successful transmission, node u is the winner of the contention, hence it acquires slot s. If the channel is found busy after the backoff time (case **ii**), it means that one or more sensor nodes have selected a *shorter* backoff time. Thus, node u tries again during the next slot (i.e. s+1) in the current superframe. Finally, when a collision is experienced by node u (case **iii**), it means that one or more other sensor nodes have selected the same backoff time for contention during slot s. In principle, node u could either retry at the next slot in the current superframe, namely s+1, or try again with the same slot s in the next superframe. The rationale behind the latter option is that, if the number of colliding sensor nodes is limited, the contention will be solved very likely during the next superframe. Another option for a colliding node u would be re-trying during slot s+1 in the current superframe with probability p_r, and defering contention to slot s in the next superframe with probability $(1 - p_r)$. This is the most general case, as the previous ones can be derived from it using a value of p_r equal to 1 or 0, respectively.

1: Choose a slot s in $[1, N]$ randomly;
2: Contend for s (using a random backoff b);
3: **Case** SUCCESS:
4: Acquire s and terminate the *Slot Acquisition* process;
5: **Case** CHANNEL-BUSY:
6: Re-try s+1 (with random backoff b);
7: **Case** COLLISION:
8: Re-try s+1 in the current superframe (with random backoff b) with probability p_r;
9: Defer contention to s in the next superframe (with random backoff b) with probability $(1 - p_r)$;

Algorithm 5: Slot Acquisition.

Algorithm 5 shows the specific actions performed by a joining node u. Initially, node u selects a random slot s in the current superframe, in order to try contention. This random choice is aimed at spreading contention attempts within the whole superframe, thus reducing the number of competitors for every single slot, and increasing the success probability. Then, node u contends for slot s using a random backoff time b. As it can be observed, the Slot Acquisition process can take more than one superframe to complete. Also, note that the Secure Slot Permutation (described in Section 8.3.2) complicates the research of a free slot, since the slot allocation pattern changes at every superframe.

8.4.2 Join procedure

In this section, we describe the complete set of actions that any joining node u performs in order to correctly join the network, which includes the execution of the Slot Acquisition algorithm described in Section 8.4.1.

We assume that, in order to correctly start the join process at superframe T_j, node u is provided with i) the shared permutation key K; and ii) the value z_j to initialize the generator counter z. Node u can retrieve such security material from an additional entity, namely *Join Manager*, which is responsible for the correct initialization of joining nodes. In principle, the Join Manager can be implemented in both a centralized and distributed fashion. Intuitively, a distributed version of the Join Manager can leverage the fact that each node already in the system holds the current values of both the permutation key and SPRNG state. However, here we consider a centralized version that initializes joining nodes via off-line methods. The Join Manager keeps itself synchronized with super frames in order to maintain an up-to-date value of the SPRNG state. Furthermore, it participates to rekeying in the case of node's leaving (see above).

1: $z \leftarrow z_j$
2: $v_u \leftarrow 0$
3: $s \leftarrow SlotAcquisition$
4: □ **handler upon**(superframe expiration)
5: $z \leftarrow z + N$
6: $v_u[s] \leftarrow 1$

Algorithm 6: Join procedure.

Algorithm 6 describes the specific actions performed by node u during the join procedure. Initially, node u initializes its generator to z_j and its permutation vector to 0, i.e. each

vector element is set to 0 (lines 1-2). Then, u executes the Slot Acquisition algorithm (Section 8.4.1) in order to acquire a free slot s (line 3). The Slot Acquisition process may take one or more superframes to be completed. Then, once initialized, the generator counter has to be kept up-to-date with respect to the one on the other nodes. Therefore, while the Slot Acquisition process is in progress, we activate a handler (line 4) that updates the generator counter whenever a superframe expires (line 5). Finally, once the Slot Acquisition process has been completed, node u accordingly updates its permutation vector to reflect such a slot acquisition (line 6). Hereafter, at the end of every superframe T_m, node u locally determines the slot s_u to be used in the next superframe T_{m+1}, according to the *Secure Slot Permutation* algorithm described in Section 8.3.2.

8.5 Analysis in steady state conditions

In this section, we show that, when the network is in steady state condition, JAMMY displays a practically negligible overhead. Also, we stress the convenience of our distributed approach with respect to a generic centralized solution.

From a computational standpoint, JAMMY performs only simple encryption operations, during the execution of the SSP algorithm (see Section 8.3). Since such operations can be efficiently performed by common hardware platforms [58][55], the computing overhead introduced by JAMMY results to be negligible. Most important, from a communication standpoint, JAMMY does not require sensor nodes to perform any additional transmission or reception (in addition to the initial reception of K and z_j from the *Join Manager*). This results in two main advantages. First, it does not determine any reduction of the available network bandwidth. Second, it does not affect power consumption of sensor nodes, so asserting itself as a solution suitable also to energy constrained devices.

Now we highlight the remarkable convenience of JAMMY with respect to a generic centralized solution against SJ attack. A centralized countermeasure, specifically targeted to IEEE 802.15.4 networks has been proposed in [12]. Such a solution enhances the *Guaranteed Time Slot* (*GTS*) mechanism provided by the IEEE 802.15.4 standard, forcing the adversary to perform the attack at random. However, the 802.15.4 standard limits the usage of GTS to a limited portion of the superframe, i.e. 7 GTS slots. Given this limitation, we believe it is not fair to consider the solution in [12]. Hence, in the

following we extend and generalize it. Specifically, we make reference to a general centralized solution, based on a *Coordinator* node. At each superframe T_m, the Coordinator node performs the following two actions: i) generates a random slot allocation pattern, namely S_m; and ii) broadcasts S_m together with a *Message Authentication Code*, in order to assure S_m authenticity and freshness. Then, every sensor node retrieves S_m, and becomes aware of the specific slot it is supposed to access at T_m. Since a new S_m is randomly created on a per-superframe basis, selective jamming results to be ineffective. However, if the adversary interfered with the transmission of S_m performed by the Coordinator node, she would be able to prevent sensor nodes from receiving the current slot allocation pattern, thus compromising network communication altogether. Conversely, this cannot happen with JAMMY, as the latter is a distributed solution.

To the purpose of our analysis, we assume that the *Coordinator* node represents a generic slot allocation pattern S_m as an array of N elements, where N is the number of slots in the superframe, which is assumed to be also equal to the number of active nodes in the network (i.e. N_A=N). The i-th element of S_m specifies the slot to be used at superframe T_m by sensor node i. Hence, each element of S_m has size equal to $\lceil log_2 N \rceil$ bits and the overall size of S_m is $N \cdot \lceil log_2 N \rceil$ bits. Then, if we denote by C the size, in bits, of the Message Authentication Code, the energy overhead introduced by the centralized solution E_{centr} is the total energy spent by all sensor nodes, at each superframe, to receive the slot allocation pattern, i.e.

$$E_{centr} = \frac{N \cdot P_{RX} \cdot ((N \cdot \lceil log_2 N \rceil) + C)}{R} \tag{8.1}$$

where P_{RX} is the radio power consumption in receive mode, and R denotes the data transmission rate.

Figure 8.1 shows the value of E_{centr} for different numbers of sensor nodes (N) and different sizes of the Message Authentication Code (C). We refer to a power consumption $P_{RX} = 35.46$ mW [55] and a data transmission rate $R = 250$ Kbit/s [10]. As it can be observed, the more sensor nodes in the network, the more energy is consumed. Also, as expected, E_{centr} is higher for larger values of C. It is worth noting that E_{centr} is the constant amount of energy spent by the network at each superframe, hence the impact on energy consumption is constant over time. Instead, as discussed above in this

section, JAMMY does not result in any communication overhead other than the initial provisioning of K and z_j. Thus, its impact on the network lifetime is negligible.

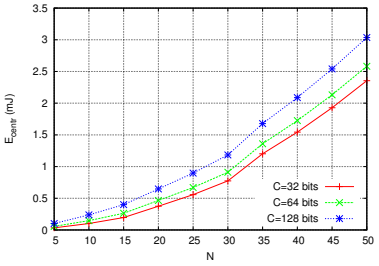


FIGURE 8.1: Energy consumed by a centralized solution.

8.6 Analysis in dynamic conditions

In this section, we develop an analytical model of the JAMMY join procedure, and use it to derive both the duration of the join procedure and the average energy consumed by sensor nodes to join the network.

We assume there are N_A active sensor nodes in the network (i.e. nodes that have already joined the network) and, hence, $N_F = N - N_A$ free slots in the superframe. Also, we assume that $N_F \geq 1$ and that $1 \leq N_j \leq N_F$ sensor nodes start the join procedure (see Section 8.4.2) at the same superframe T_j. Figure 8.2 depicts a possible situation where: (i) the superframe is composed of $N = 3$ slots; (ii) there is a single active node A in the network; and (iii) two more nodes, namely B and C, start their join procedure at superframe T_j. Due to the Secure Slot Permutation algorithm, node A uses the second slot during superframe T_j, and the first one at superframe T_{j+1}.

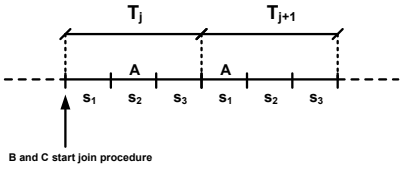


FIGURE 8.2: Nodes B and C join the network.

The analysis is divided into two parts. First, we analyze the slot acquisition phase and consider all the events that can occur during the contention to acquire a slot in the superframe. For each event, we derive the occurrence probability and the energy spent by contending sensor nodes. Then, we use the above-mentioned probabilities to derive a

Discrete Time Markov Chain (DTMC) model of the overall join procedure, and calculate the probability distribution function of the joining time and the average energy spent by sensor nodes to join the network. In order to simplify the analysis, we make the following assumptions.

1. The number of joining nodes N_j is equal to the number of free slots in the superframe, i.e. $N_j = N_F = N - N_A$. Of course, this is a worst case condition.
2. A simple CSMA/CA (Carrier Sense Muliple Access with Collision Avoidance) algorithm, based on random backoff delays, is used to solve contention among sensor nodes trying to acquire the same slot. Specifically, before performing a channel sensing operation and transmitting a packet, each sensor node waits for a random backoff time w in the range $\{0,\ 1,\ ...,\ W_B - 1\} \cdot D_{bo}$, where W_B is the backoff window size and D_{bo} is the backoff unit.
3. All joining nodes start competing at the first slot in the superframe, i.e. we do not consider the initial randomization in the Slot Acquisition algorithm (Algorithm 5, line 1). This maximizes the contention and, hence, models a worst case condition.
4. The retry probability p_r (Algorithm 5, line 8) is equal to 0, i.e., upon a collision, *all* colliding sensor nodes defer their transmission to the next superframe.

8.6.1 Event Probabilities

We start considering all the events that can occur during the contention for a slot and, for each event, we derive the probability to occur, as well as the energy spent by the contending sensor nodes. Let us focus on a generic slot s^* in the superframe and assume that M, out of N_j, sensor nodes are contending for slot s^*.

First, let us consider the case when slot s^* is already used by one of the N_A active nodes. Since active nodes access their slot without any backoff delay, they have priority over joining nodes. Hence all M joining nodes find slot s^* already busy and, according to the Slot Acquisition algorithm (Algorithm 5), wait for the next slot in the same superframe. Now, let us analyze the case when slot s^* is free, i.e. it is not currently used by any active node. Then, the contention can result in one of the following outcomes.

(a) **SUCCESS**. One of the joining sensor nodes successfully transmits during slot s^*.

(b) **COLLISION**. A collision is experienced by k joining nodes, $2 \leq k \leq M$.

(c) **BUSY CHANNEL**. The channel is found busy by $h = M - k$ joining nodes, $0 \leq h \leq M - 2$, that schedule a retry at the next slot.

Now, we derive the probability of each of the above mentioned events to occur. A successful transmission (case (a)) occurs when one joining node generates a backoff time shorter than the one of all the other $M - 1$ contending nodes. Let $P_s^F(M)$ denote the probability that a successful transmission occurs. Given W_B the backoff window size, every node can extract a backoff w, $w \in \{0,\ 1,\ ...,\ W_B - 1\} \cdot D_{bo}$, hence the following equation holds.

$$P_s^F(M) = M \cdot \sum_{w=0}^{W_B-1} \left(\frac{1}{W_B}\right) \cdot \left(\frac{W_B - 1 - w}{W_B}\right)^{M-1} \tag{8.2}$$

Equation 8.2 can be explained as follows. For every possible backoff time w that can be generated by a joining node with probability $\frac{1}{W_B}$, the second term inside the sum is the probability that the remaining $M-1$ sensor nodes extract a backoff time larger than w. Then, all M combinations, corresponding to the different sensor nodes, are considered.

Let us now consider case (b), where two or more joining nodes generate the same backoff time, thus starting their transmission at the same time and experiencing collision. Let $P_c^F(k \mid M)$ be the probability that k out of the M contending nodes, $k \leq M$, experience collision at the free slot s^*. Then, the following equation holds.

$$P_c^F(k \mid M) = \binom{M}{k} \sum_{w=0}^{W_B-1} \left(\frac{1}{W_B}\right)^k \left(\frac{W_B - 1 - w}{W_B}\right)^{M-k} \tag{8.3}$$

In equation 8.3, for every possible backoff time w, the term inside the sum gives the probability that k joining nodes randomly pick up a value equal to w, and $M - k$ sensor nodes choose a value larger than w. Of course, all $\binom{M}{k}$ possible combinations are considered.

Finally, in case (c), $h = M - k$ nodes, $(0 \leq h \leq M - 2)$ find the channel busy. Let $P_b^F(h \mid M)$ be the probability that h joining nodes find the channel busy at the free slot s^*, in the presence of M contending nodes. Since h nodes have found the channel busy, then $M - h$ nodes have extracted the same (minimum) backoff value, and collided.

Hence,

$$P_b^F(h \mid M) = P_c^F(M - h \mid M) \tag{8.4}$$

Now, we derive formulas to compute the energy spent by joining nodes in the different considered cases. Let us denote by P_{RX} (P_{TX}) the power consumed by a sensor node radio in receive (transmit) mode. Also, we assume there are M joining nodes all contending for the same slot. The energy consumed by all sensor nodes depends on the specific outcome following the contention. If one of the M joining nodes wins the contention, the energy $E_s(M)$ is:

$$E_s(M) = M \cdot E_{CS} + P_{TX} \cdot D_{tx} + P_{RX} \cdot D_{ack} \tag{8.5}$$

where D_{tx} and D_{ack} denote the duration of packet transmission time and ack reception time respectively, while $E_{CS} = P_{RX} \cdot D_{CS}$ denotes the energy spent to perform a channel sensing operation, and D_{CS} is the duration of the channel assessment. The first term in Equation 8.5 accounts for the energy consumed by all the M joining nodes to perform their channel sensing operation, while the other terms account for the additional energy consumed by the winner node (all the other sensor nodes find the channel busy and give up). The latter energy is spent for transmitting the packet $P_{TX} \cdot D_{tx}$ and receiving the ack $P_{RX} \cdot D_{ack}$. Following the same line of reasoning, the energy $E_c(k \mid M)$ consumed when k out of the M nodes experience a collision, with $2 \leq k \leq M$, can be expressed as follows, where D_{to} is the timeout interval.

$$E_c(k \mid M) = M \cdot E_{CS} + k \cdot (P_{TX} \cdot D_{tx} + P_{RX} \cdot D_{to}) \tag{8.6}$$

Finally, let us focus on the energy spent by a sensor node to transmit a packet when it has acquired a slot. Hence, the following equation holds.

$$E_u = P_{TX} \cdot D_{tx} + P_{RX} \cdot D_{ack} \tag{8.7}$$

This energy is due to the transmission of packet and reception of the ack.

8.6.2 Markov Chain Derivation

We are now in the position to derive a Discrete Time Markov Chain (DTMC) model of the JAMMY join procedure, that we use later in Section 8.6.3 to derive the probability distribution of the joining time and the average energy consumed by sensor nodes during the join procedure.

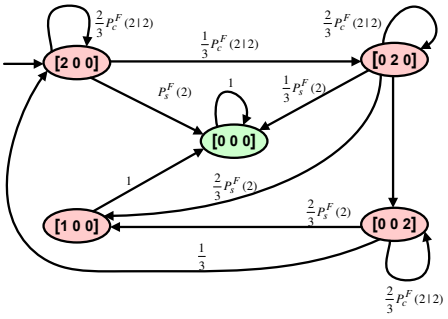


FIGURE 8.3: Markov chain for $N = 3$, $N_A = 1$, $N_j = 2$.

We observe the system at the beginning of every superframe T_m, and represent the system state as a vector $X_m = [n_1,\ n_2, ...,\ n_N]$ where element n_i $(i = 1,\ 2,\ ...,\ N)$ refers to the i-th slot of the superframe T_m. Specifically, n_i indicates the number of joining nodes that try contention at slot i during superframe T_m. We recall that we consider N_j sensor nodes joining the network at the same superframe T_j and assume that $N_j = N - N_A$, where N_A is the number of assigned slots at superframe T_j. Since there are N_j joining nodes, we have $n_i \leq N_j, \forall i$. Also, we denote by $S_m = [\overline{s}_1,\ \overline{s}_2,\ ...,\ \overline{s}_N]$ the slot allocation pattern at the beginning of a superframe T_m. Specifically, $\overline{s}_i$ indicates the state of slot s_i at superframe T_m, i.e. if it is free (F) or if it is acquired (A). Obviously, at superframe T_j, N_A slots must have state equal to A. We start to analyze the system from superframe T_j. In the general case, at the beginning of superframe T_j, the system state is $X_j = [N_j,\ 0, ...,\ 0]$ since all joining nodes contend for the first slot of superframe T_j (Assumption 3). Now, we derive all possible states in which the network could evolve starting from the initial state. For the sake of simplicity, in the following we explain the details of the model focusing on a simple example. Specifically, we consider the case when the superframe is composed of $N = 3$ slots, there is 1 active node (i.e. $N_A = 1$), and $N_j = 2$ nodes start their join procedure at T_j (see Figure 8.2). A procedure to derive a DTMC describing the join procedure with arbitrary values of N, N_A and N_j is reported in Appendix.

With reference to the considered example scenario, the initial system state is $X_j = [2,\ 0,\ 0]$, i.e. the two joining nodes start contending for the first slot of superframe T_j (see Figure 8.3). Then, the system can evolve in different states, depending on both the slot allocation pattern at superframe T_j, i.e. S_j, and the specific events that occur during the contention for each slot in superframe T_j. To properly model the evolution of the system we have to consider all the possible slot allocation patterns S_j at superframe T_j. Since there is a single active node at superframe T_j, the possible slot allocation patterns are $S_j^1 = [A,\ F,\ F]$, $S_j^2 = [F,\ A,\ F]$ and $S_j^3 = [F,\ F,\ A]$. Also, since the different slot allocation patterns are due to the SSP algorithm (Algorithm 4), all patterns have the same probability to occur, i.e. $P(S_j^1) = P(S_j^2) = P(S_j^3) = \frac{1}{3}$.

Let us assume $S_j = S_j^1 = [A,\ F,\ F]$. In this case, the two joining nodes contend for the first slot of the superframe, which is already owned by the active node. Hence, they lose contention and, according to Algorithm 5, they have to retry at the second slot. Then, there are two possible evolutions for the system, depending on the specific event that occurs at the second slot. In case of a collision between the two sensor nodes, which occurs with a probability equal to $P_c^F(2 \mid 2)$, the system at superframe T_{j+1} will be in state $[0,\ 2,\ 0]$, since the two nodes will retry to contend for the second slot (Assumption 4). Instead, if one of the two joining nodes wins the contention at the second slot, which occurs with probability $P_s^F(2)$, it becomes the owner of the slot and ends the join procedure. Hence, starting from superframe T_{j+1}, it has to be considered as an active node. The sensor node which loses the contention at the second slot, retries to contend for the third slot. Since we are assuming $S_j = [A,\ F,\ F]$, the third slot is free. Hence, it surely wins the contention and ends the join procedure. Thus, the system state, at T_{j+1}, will be $[0,\ 0,\ 0]$, i.e. all joining nodes have found their slot and have completed the join procedure.

Let us now consider the case $S_j = S_j^2 = [F,\ A,\ F]$. In this case there are two possible transitions for the system. The state of the system could remain the same, i.e. $X_{j+1} = X_j = [2,\ 0,\ 0]$, or the system could evolve in state $X_{j+1} = [0,\ 0,\ 0]$. Since the first slot is free in this case, the first transition occurs when the two joining nodes experience a collision at the first slot, which happens with probability $P_c^F(2 \mid 2)$. Instead, the second transition happens when a success occurs in the first slot, i.e. with probability $P_s^F(2)$.

Finally, let us analyze the last case, i.e. $S_j = S_j^3 = [F,\ F,\ A]$. The possible transitions

for the system are the same as in the previous case, i.e. the system could transit to state $[2,\ 0,\ 0]$, with probability $P_c^F(2 \mid 2)$, or to state $[0,\ 0,\ 0]$ with probability $P_s^F(2)$. Figure 8.3 reports all transitions of the system starting from state $[2,\ 0,\ 0]$, along with their probability. Note that the probability of each transition considers also the probability of each possible slot allocation pattern to occur.

Following a similar line of reasoning, we can derive all the possible transitions that can occur when the system is initially in state $[0,\ 2,\ 0]$. When the network is in this state, there is only one active node in the network, since all the two joining nodes have not terminated their join procedure yet. Hence, the possible equiprobable slot allocation patterns S_m are $S_m = [A,\ F,\ F]$, $S_m = [F,\ A,\ F]$, and $S_m = [F,\ F,\ A]$. As shown in Figure 8.3, there are four possible transitions for the system in this case, namely $[0,\ 2,\ 0] \rightarrow [0,\ 0,\ 0]$, $[0,\ 2,\ 0] \rightarrow [0,\ 2,\ 0]$, $[0,\ 2,\ 0] \rightarrow [0,\ 0,\ 2]$, and $[0,\ 2,\ 0] \rightarrow [1,\ 0,\ 0]$. First, we analyze transition $[0,\ 2,\ 0] \rightarrow [0,\ 0,\ 0]$. This transition occurs only when the second and third slots are free, i.e. $S_m = [A,\ F,\ F]$, and one of the two joining nodes wins the contention with the other one at the second slot, which happens with probability equal to $P_s^F(2)$. Hence, the probability for transition $[0,\ 2,\ 0] \rightarrow [0,\ 0,\ 0]$ to occur is $\frac{1}{3} \cdot P_s^F(2)$. Transition $[0,\ 2,\ 0] \rightarrow [0,\ 2,\ 0]$ occurs when the second slot is free, i.e. $S_m = [A,\ F,\ F]$ or $S_m = [F,\ F,\ A]$, and a collision between the two joining nodes occurs. Hence, this transition has a probability equal to $\frac{2}{3} \cdot P_c^F(2 \mid 2)$ to occur. Transition $[0,\ 2,\ 0] \rightarrow [0,\ 0,\ 2]$ occurs when the two joining nodes move from the second to the third slot and, then, they experience a collision. This is possible only if the second slot is used by the active node, i.e. $S_m = [F,\ A,\ F]$. Hence the transition has a probability equal to $\frac{1}{3} \cdot P_c^F(2 \mid 2)$. Finally, transition $[0,\ 2,\ 0] \rightarrow [1,\ 0,\ 0]$ can occur in two cases. The first one is when the second slot is acquired by an active node, i.e. $S_m = [F,\ A,\ F]$, the two joining nodes move to the third slot, and one of them wins the contention for the third slot. The sensor node winning the contention terminates the join procedure and becomes an active node. Instead, the sensor node losing the contention retries to contend at the first slot of the subsequent superframe. This events occur with probability $\frac{1}{3} \cdot P_s^F(2)$. The second situation which leds the system to state $[1,\ 0,\ 0]$ is when the second slot is free and the third one is owned by any active node, i.e. $S_m = [F,\ F,\ A]$, and a success occurs at the second slot. In this case, one joining node wins the contention and becomes an active node while the other one tries to contend for the third slot. Since the third slot is already allocated, the joining node finds the

channel busy and schedules a retry at the first slot of the subsequent superframe.

Let us now analyze the transitions originating from state $[0,\ 0,\ 2]$. There are three possible transitions in this case, namely $[0,\ 0,\ 2] \rightarrow [0,\ 0,\ 2]$, $[0,\ 0,\ 2] \rightarrow [2,\ 0,\ 0]$, and $[0,\ 0,\ 2] \rightarrow [1,\ 0,\ 0]$. The first transition occurs when the third slot is free, i.e. $S_m = [A,\ F,\ F]$ or $S_m = [F,\ A,\ F]$ and a collision between the two joining nodes occurs at the third slot. The corresponding probability is $\frac{2}{3} \cdot P_c^F(2 \mid 2)$. Transition $[0,\ 0,\ 2] \rightarrow [2,\ 0,\ 0]$ is possible only when the third slot is allocated. In fact, in this case, the two joining nodes find the channel busy at the third slot and retry to the first slot of the subsequent superframe. This event has a probability equal to $\frac{1}{3}$ to occur. The last transition is $[0,\ 0,\ 2] \rightarrow [1,\ 0,\ 0]$. There are two conditions for this transition to occur. First, the third slot has to be free, i.e. $S_m = [A,\ F,\ F]$ or $S_m = [F,\ A,\ F]$. Second, a success has to occur at the third slot. Hence, the transition probability is equal to $\frac{2}{3} \cdot P_s^F(2)$.

Let us turn our attention to state $[1,\ 0,\ 0]$. In this case one of the two joining nodes has completed the join procedure in a previous superframe. Hence, the possible slot allocation patterns are $S_m = [A,\ A,\ F]$, $S_m = [A,\ F,\ A]$, and $S_m = [F,\ A,\ A]$. There is only one possible transition for the system starting from state $[1,\ 0,\ 0]$. In fact, whatever the slot allocation pattern, the remaining joining node will surely find a free slot in the current superframe. Hence, the system evolves to state $[0,\ 0,\ 0]$ with probability equal to 1. Finally, when the system reaches state $[0,\ 0,\ 0]$, the join procedure is completed. Hence, the system will never change its state, i.e. state $[0,\ 0,\ 0]$ is an absorbing state.

Each transition described above is characterized by an average energy consumed by joining sensor nodes. We report the derivation of the energy associated to each transition of the considered example in Appendix.

So far, we have considered a simple system with only three sensor nodes and three available slots. In the general case, the possible system states, the transition probabilities, and the average energy consumption of each transition can be derived using the same line of reasoning. In Appendix we present an algorithm to derive a DTMC in the general case, i.e., for any value of N, N_A, and N_j.

Let Ω be the set of possible states for the system. Once the states of the Markov Chain are known and the transition probabilities (and associated energy consumptions) have

been computed, we can derive two matrices, namely P and E. Specifically, P is the transition probability matrix of the system, i.e. each element P_{XY}, $X,\ Y \in \Omega$, indicates the probability that the system changes its state from X to Y. Instead, matrix E refers to the energy consumption of joining nodes, i.e. each element E_{XY} represents the average energy consumed by joining nodes when the system changes its state from X to Y. In the following section, we use P and E to derive both the probability distribution of the joining time and the average energy spent by joining nodes during the join procedure. Finally, we point out that the general procedure presented in Appendix takes as input $N,\ N_A$, and N_j, and produces both P and E.

8.6.3 Computation of performance metrics

Now, we derive the following performance metrics.

$P_{join}(k),\ (k = 0,\ 1,\ ...)$: defined as the probability that the join procedure is over at the beginning of superframe T_{j+k}, i.e. the probability that *all* joining nodes complete their join procedure in k superframes. $P_{join}(k)$ provides the probability mass function of joining time.
$\overline{E_k},\ (k = 0,\ 1,\ ...)$: defined as the average energy spent by *all* joining nodes during superframe T_{j+k}.
$\overline{E_{join}}$: defined as the average energy consumed by *all* joining nodes during the entire duration of the join procedure. $\overline{E_{join}}$ represents the total join overhead in terms of energy.

To derive $P_{join}(k)$, we sort the states of the Markov Chain so that the initial state of the system $X_j = [N_j,\ 0, ...,\ 0]$ and the final state $X_f = [0,\ 0, ...,\ 0]$ are the first and last one in the sequence, respectively. Also, let v_0 be the *initial* probability vector, and $v_k,\ k \geq 0$, the probability vector related to superframe T_{j+k}. With no loss of generality, we can assume that $v_0 = [1, 0, \ldots, 0]$, thus $v_k = v_0 \cdot P^k$. Hence, the probability that the join procedure has been completed after k superframes, i.e. $P_{join}(k)$, corresponds to the probability that the system state at step k is X_f. Let us denote by $|\Omega|$ the cardinality of the set Ω, i.e. the total number of system states. Since X_f is the last state in the sequence, it follows that

$$P_{join}(k) = v_k[|\Omega|] \tag{8.8}$$

Let us now derive the average energy $\overline{E_k}$ spent by all joining nodes during superframe $T_{j+k},\ k \geq 0$. Let P_X^k denote the probability that the system is in state X at superframe T_{j+k} for any $X \in \Omega$. Since P_X^k is the component of vector v_k associated to state X, then the following equation holds.

$$\overline{E_k} = \sum_{X \in \Omega} P_X^k \cdot \sum_{Y \in \Omega} E_{XY} P_{XY} \tag{8.9}$$

Equation 8.9 can be justified as follows. In order to calculate $\overline{E_k}$, we must consider all possible state changes that can occur from superframe T_{j+k} to the next superframe T_{j+k+1}. Hence, the outer sum considers any possible system state X, at superfame T_{j+k}, whose occurrence probability is P_X^k. Then, for each state X, the inner sum considers all possible states Y where the system can evolve to. Such a transition has a probability occurrence P_{XY}, and is associated to an average energy consumption E_{XY}.

Finally, we derive the average energy spent by all joining nodes during the whole join procedure, i.e. $\overline{E_{join}}$. We recall that the join procedure is over when the system reaches state $X_f = [0,\ 0,\ ...,\ 0]$. Thus,

$$\overline{E_{join}} = \mu_{X_j} \tag{8.10}$$

where μ_{X_j} indicates the average energy consumed by all joining nodes to reach state X_f starting from state X_j. According to [116], the average energy μ_X consumed by the network to reach state X_f, starting from any state $X \in \Omega$, can be obtained by solving the following linear equation system, with μ_X as unknowns, and $\mu_{X_f} = 0$:

$$\mu_X = \sum_{Y \in \Omega} P_{XY} \cdot (E'_{XY} + \mu_Y),\ \forall X \in \Omega \tag{8.11}$$

where E'_{XY} differs from E_{XY}, since it does not take into account the energy consumed by joining nodes after they have completed the join procedure, i.e. when they become active nodes.

8.7 Results

We now evaluate the overhead introduced by the join procedure, in terms of duration and energy consumption, by using the equations derived in the previous section. To

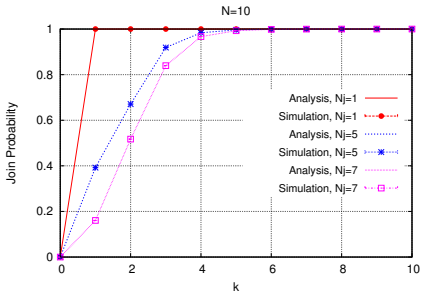


FIGURE 8.4: $P_{join}(k)$ $(N = 10)$.

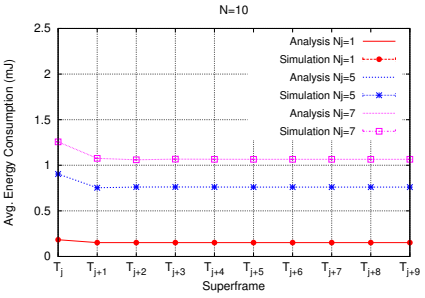


FIGURE 8.5: $\overline{E_k}(N = 10)$.

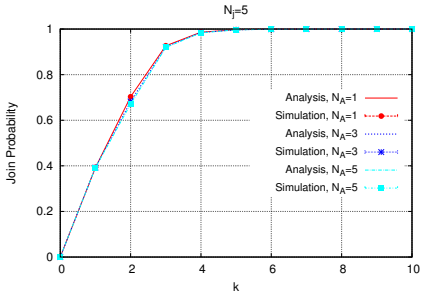


FIGURE 8.6: $P_{join}(k)$ $(N_j = 5)$.

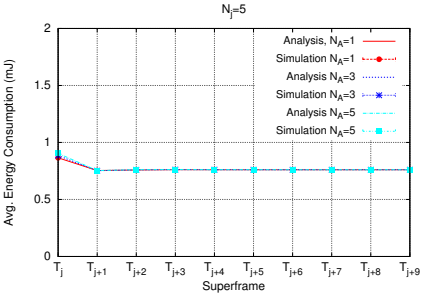


FIGURE 8.7: $\overline{E_k}$ $(N_j = 5)$.

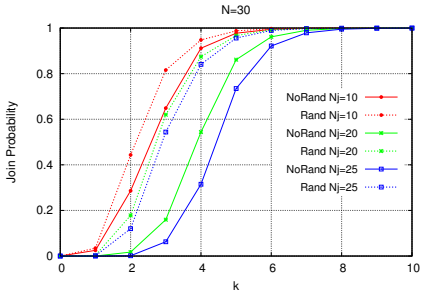


FIGURE 8.8: $P_{join}(k)$ $(N = 30)$.

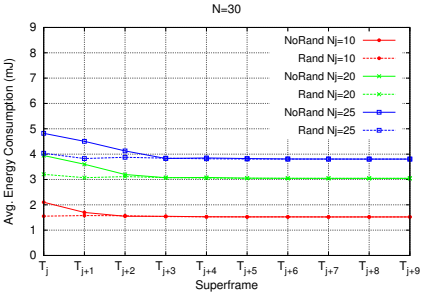


FIGURE 8.9: $\overline{E_k}$ $(N = 30)$.

validate our analytical results, we also rely on simulation. To this end, we implemented JAMMY using the *ns-2* simulation tool [54].

Parameter	Value
Data transmission rate (R)	250 Kbit/s
Power Consumption in TX mode (P_{TX})	31.32 mW
Power Consumption in RX mode (P_{RX})	35.46 mW
Slot duration (D_{slot})	7.4 ms
Duration of channel assessment (D_{CS})	128 μs
Packet transmission time (D_{tx})	4.256 ms
Ack transmission time (D_{ack})	352 μs
Timeout interval (D_{to})	864 μs
Backoff window size (W_B)	8

TABLE 8.1: Parameters used in our analysis.

Our results refer to a single-hop star network topology, where the number of joining nodes N_j is always equal to the number of free slots at superframe T_j. Thus, $N_j =$

$N - N_A$. Unless stated otherwise, the other parameter values are as shown in Table 8.1. The considered values have been inspired by the 802.15.4 standard. For every simulation experiment we performed ten independent replications, each one of which consists of 1,000,000 join procedures. Simulation results shown below are averaged over all replications. We also derived confidence intervals by using the independent replications method and 95% confidence level. However, confidence interval are very small and cannot be appreciated in the presented graphs. The analysis is organized into two parts. In Section 8.7.1, we show our analytical results and validate them through simulation. Then, in Section 8.7.2, we evaluate, through simulation, the impact of the initial randomization provided by the Slot Acquisition algorithm on the join procedure.

8.7.1 Analytical results

Figure 8.4 shows the duration of the join phase calculated through Equation 8.8, for $N = 10$ and different N_j. In all the considered scenarios the number of assigned slots at the beginning of the join procedure is always equal to $N - N_j$. The results are derived, both through analysis and simulations, without considering the initial randomization of the Slot Acquisition algorithm. As it can be observed, analytical and simulation results almost overlap. As expected, the duration of the joining phase significantly increases as N_j increases. This is because, when many sensor nodes try to join the system simultaneously, the probability of collisions in the slot acquisition is very high. However, in all the considered scenarios the join procedure terminates in few superframes. Precisely, the 99-th percentile of the distribution is always less than or equal to 5 superframes.

Figure 8.5 shows the average energy consumed by all N_j joining nodes during each superframe (derived from Equation 8.9), for $N = 10$ and different numbers of joining nodes N_j. As above, analytical and simulation results almost overlap. The average energy consumption exhibits the same trend for all the considered scenarios. At T_j, the energy consumption is higher than during the next superframes as all joining nodes are still contending to acquire a free slot and, hence, there is the maximum level of contention. However, the consumed energy tends to a constant value once the join procedure has been completed. Also, note that energy consumption is higher for greater values of N_j, i.e. in the presence of more joining nodes. From Figure 8.4 it emerges that, even in the worst case, the 99-th percentile of the join procedure is less than

six superframes. Hence, from figure 8.5 we can conclude that, apart from the initial superframe T_j, the energy consumed by joining nodes is only slightly higher than the energy consumed after the join procedure has been completed (e.g. from T_{j+6} onwards). This means that the additional energy consumed by sensor nodes during the join phase is very limited, in comparison with the total energy consumed during the entire network lifetime. Table 8.2 shows the average total energy $\overline{E_{join}}$ (Equation 8.10) consumed by *all* joining nodes during the entire join phase (calculated through Equation 8.9). It characterizes the total overhead of the join procedure in terms of energy consumption. As expected, $\overline{E_{join}}$ increases when N_j increases but remains quite low in all considered scenarios.

N_j	$\overline{E_{join}}$ (mJ)	$\overline{E_{join}}/N_j$
1	0.47	0.47
5	3.65	0.73
7	6.93	0.99

TABLE 8.2: Average total energy consumption at each superframe

Figure 8.6 shows the impact of the number of (already) assigned slots in the superframe, i.e. N_A, on the duration of the join procedure. Specifically, Figure 8.6 shows the duration of the join phase (calculated through Equation 8.8), for $N_j = 5$ joining nodes, and different N_A values. We can see that the number of assigned slots in the superframe does not significantly affect the duration of the join procedure, which is always completed in a low number of superframes. As above, analytical and simulation results overlap.

Finally, Figure 8.7 reports the average energy consumed by *all* joining nodes during each superframe (calculated through Equation 8.9), for $N_j = 5$ and different numbers of assigned slots N_A. As above, analytical and simulation results almost overlap. Again, for all the considered scenarios, the average energy consumption is quite high at superframe T_j, while it reaches a low and steady value after few periods. The number of assigned slots does not affect at all the average energy consumption of joining nodes. Thus, the join procedure results to be very efficient even in the presence of a loaded network.

8.7.2 Simulation Results: Initial Randomization

Now, we evaluate the impact of the initial randomization provided by the Slot Acquisition algorithm on the duration of the join procedure. To this purpose, we rely on

simulation only. Since the possible effects of the initial randomization can be appreciated only for a large number of joining nodes, in this set of experiments we considered a larger superframe, i.e. $N = 30$. Figure 8.8 shows the probability mass function of the join phase duration, for different numbers of joining nodes, namely $N_j = \{10,\ 20,\ 25\}$. The curves labelled as *NoRand* refer to the case without initial randomization, i.e. all sensor nodes contend for the first superframe slot. Instead, the curves labelled as *Rand* refer to the case with initial randomization, i.e. all sensor nodes choose the initial slot at which to start contention at random. As expected, the initial randomization significantly reduces the duration of the join phase. This is because, with inital randomization, joining nodes are more likely to start the join procedure at different time slots and, hence, they are less likely to experience collisions. The benefits of using the initial randomization are very clear for any considered value of N_j, and become more and more relevant as the number of joining nodes increases.

Figure 8.9 shows that the initial randomization also reduces the average energy consumed during the join procedure. This is because sensor nodes are less likely to experience collisions and, thus, the energy consumption is reduced, due to i) a lower number of slots during which collisions occur, and ii) a shorter join duration altogether.

8.8 Conclusion

We have presented JAMMY, a novel and distributed solution against selective jamming attacks in TDMA WSNs. JAMMY contrasts selective jamming by forcing the adversary to perform the attack at random, hence reducing its effectiveness to $1/N$, where N is the number of slots in the superframe. To the best of our knowledge, JAMMY is the first distributed solution against selective jamming in TDMA-based WSNs presented so far. When the network is in steady state condition, JAMMY does not introduce any communication or energy overhead, regardless the number of sensor nodes in the network. Hence, it outperforms a generic centralized solution, operating in similar conditions, in terms of available bandwidth and energy efficiency. We have also derived a DTMC analytical model of the join process in the presence of JAMMY, and validated it through simulation. Our results show that the join process always terminates in a short number of superframes, and introduces a limited energy consumption on joining nodes.

Part III
IEEE 802.14.e TSCH WSNs

Chapter 9

Background on IEEE 802.15.4e TSCH

9.1 Introduction

The solutions proposed in part I and II of this thesis significantly improve the performance/robustness of both 802.15.4 and TDMA-based networks. However, the fact that these networks rely on one single channel for communication can significanlty degrade their performance in real-world applications. WSNs typically share their radio medium with other wireless technologies such as WiFi [13], Bluetooth [14] or even cordless phones and microwave ovens [15] and, hence, suffer from external interference. In addition, the performance of WSNs is affected by multi-path fading since any wall, person, object in their surroundings acts as a reflector for RF signals [16]. Using multiple channels for communication, together with a channel hopping scheme, has been shown to be an effective way to mitigate both external interference and multi-path fading [16, 17]. For this reason, many industrial wireless technologies such as ISA [18] and WirelessHART [19] adopt channel hopping. In this perspective, IEEE has recently proposed the IEEE 802.15.4e standard that improves the IEEE 802.15.4 MAC to better address the emerging needs of embedded industrial applications. The 802.15.4e standard extends the previous 802.15.4 standard by introducing two different categories of MAC enhancements, namely *MAC behaviors*, to support specific application domains, and *general functional improvements* that are not tied to any specific application domain.

The MAC behavior modes defined by the 802.1.5.4e standard are listed below.

- *Radio Frequency Identification Blink* (BLINK): intended for applications such as item and people identification, location, and tracking;
- *Asynchronous multi-channel adaptation* (AMCA): targeted to application domains where large deployments are required (e.g., process automation/control, infrastructure monitoring, etc.);
- *Deterministic and Synchronous Multi-channel Extension* (DSME): aimed to support industrial and commercial applications with stringent timeliness and reliability requirements;
- *Low Latency Deterministic Network* (LLDN): intended for applications requiring very low latency requirement (e.g., factory automation, robot control)
- *Time Slotted Channel Hopping* (TSCH): targeted to application domains such as process automation.

The general functional enhancements, not specifically tied to a particular application domain, are as follows.

- *Low Energy* (LE). This mechanism is intended for applications that can trade latency for energy efficiency. It allows a device to operate with a very low duty cycle (e.g., 1% or below), while appearing to be always on to the upper layers. This mechanism is extremely important for enabling the Internet of Things paradigms as Internet protocols have been designed assuming that hosts are always on. However, it may be useful also in other applications scenarios (e.g., event-driven and/or infrequent communications, networks with mobile nodes).
- *Information Elements* (IE). The concept of IEs was already present in the 802.15.4 standard. It is an extensible mechanism to exchange information at the MAC sublayer.
- *Enhanced Beacons* (EB). Enhanced Beacons are an extension of the 802.15.4 beacon frames and provide a greater flexibility. They allow to create application-specific beacons, by including relevant IEs, and are used in the DSME and TSCH modes.

- *Multipurpose Frame*. This mechanism provides a flexible frame format that can address a number of MAC operations. It is based on IEs.
- *MAC Performance Metrics* are a mechanism to provide appropriate feedback on the channel quality to the networking and upper layers, so that appropriate decision can be taken. For instance the IP protocol running on top of 802.15.4e MAC may implement dynamic fragmentation of datagrams depending on the channel conditions.
- *Fast Association* (FastA). The 802.15.4 association procedure introduces a significant delay in order to save energy. For time-critical application latency has priority over energy efficiency. The FastA mechanism allows a device to associate in a reduced amount of time.

In the last part of this thesis we focus on analyzing the *Time Slotted Channel Hopping* (TSCH) MAC behavior mode which combines *time slotted* access, with *multi-channel* and *channel hopping* capabilities, thus providing increased network capacity, high reliability and predictable latency, while maintaining very low duty cycles (i.e., energy efficiency). These unique characteristics make TSCH one of the most promising technologies for future real-world WSNs applications.

The remainder of this chapter is organized as follows. In section 9.2 we describe the details of the IEEE 802.15.4e MAC behavior modes, with a special emphasis on TSCH. Then, in section 9.3 we motivate the work we present in chapter 10 and 11.

9.2 802.15.4e MAC behavior modes

In this section we describe the MAC behavior modes that have been introduced in the previous section.

The *Radio Frequency Identification Blink* (BLINK) mode is intended for application domains such as item/people identification, location, and tracking and is, thus, very relevant in the perspective of Internet of Things. Specifically, it allows a device to communicate its ID (e.g., a 64-bit source address) to other devices. The device can also transmit its alternate address and, optionally, additional data in the payload. No prior association is required and no acknowledgement is provided to the sending device. The

BLINK mode is based on a minimal frame consisting only of the header fields that are necessary for its operations. The BLINK frame can be used by "transmit only" devices to co-exist within a network, utilizing an Aloha protocol.

The *Asynchronous multi-channel adaptation* (AMCA) mode is targeted to application domains where large deployments are required, such as smart utility networks, infrastructure monitoring networks, and process control networks. In such networks using a single, common, channel for communication may not allow to connect all the devices in the same PAN. In addition, the variance of channel quality is typically large, and link asymmetry may occur between two neighboring devices (i.e., a device may be able to transmit to a neighbor but unable to receive from it). The AMCA mode relies on asynchronous multi-channel adaptation and can be used only in non Beacon-Enabled PANs.

The *Deterministic and Synchronous Multi-channel Extension* (DSME) mode is intended for the support of industrial applications (e.g., process automation, factory automation, smart metering), commercial applications (such as home automation, smart building, entertainment) and healthcare applications (e.g. patient monitoring, telemedicine). This kind of applications requires low and deterministic latency, high reliability, energy efficiency, scalability, flexibility, and robustness. The 802.15.4 standard provides Guaranteed Time Slots (GTSs). However, the GTS mode has a number of limitations. It only includes up to 7 slots and, thus, it is not able to support large networks. In addition, it relies on a single frequency channel. DSME enhances GTS by grouping multiple superframes to form a multi-superframe and using multi-channel operation. Like GTS, DSME runs on Beacon-enabled PANs. All the devices in the PAN synchronize to multi-superframes via beacon frames. A multi-superframe is a cycle of superframes, where each superframe includes the beacon frame, the Contention Access Period, and Contention Free Period (i.e., GTS slot). A pair of nodes wakes up at a reserved GTS slot to exchange a data frame and an ACK frame. In order to save energy, DSME uses CAP reduction, i.e., the Contention Access Period (CAP) is only in the first superframe of the multi-superframe, while it is suppressed in subsequent superframes.

The *Low Latency Deterministic Network* (LLDN) mode is mainly targeted to industrial

and commercial applications requiring low and deterministic latency. Typical application domains addressed by LLDN include factory automation (e.g., automotive manufacturing), robots, overhead cranes, portable machine tools, milling machines, computer-operated lathes, automated dispensers, cargo, airport logistics, automated packaging, conveyors. In this kind of applications typically there is a large number of sensors/actuators observing and controlling a system, e.g., a production line or a conveyor belt. In addition, applications have very low requirements in terms of latency (transmission of sensor data in 5-50 ms, and low round-trip time). To guarantee stringent latency requirements of target applications LLDN only supports the star (i.e., single hop) topology, and uses a superframe, based on timeslots, with small packets. Keeping the size of packets (and, hence, timeslots) short leads to superframes with short duration (e.g., 10 ms). Obviously, the number of timeslots in a superframe determines the number of devices that can access the channel. Since the number of devices may very large (there may be more than 100 devices per PAN coordinator) LLDN allows the PAN coordinator to use multiple transceivers on different channels. In the LLDN mode each superframe consists of a beacon timeslot, management timeslots (if present), and a number of base timeslots of equal size. Base timeslots include uplink timeslots and bidirectional timeslots. There are two categories of base timeslot, namely dedicated and shared group timeslots. Dedicated timeslots are assigned to a specific node (owner) that has the exclusive access on them, while shared group timeslots are assigned to more than one device. The devices use the slotted CSMA/CA algorithm described in chapter 2 to contend for shared group timeslots. In addition, they use a simple addressing scheme with 8-bit addresses. The LLDN mode includes a Group ACK (GACK) function to reduce the bandwidth overhead. GACK is sent by the PAN coordinator in a superframe to stimulate the retransmission of failed transmission in uplink timeslots.

The *Time Slotted Channel Hopping* (TSCH) mode is mainly intended for the support of process automation applications with a particular focus on equipment and process monitoring. Typical segments of the TSCH application domain include oil and gas industry, food and beverage products, chemical products, pharmaceutical products, water/waste water treatments, green energy production, climate control. TSCH combines time slotted access, already defined in the IEEE 802.15.4 MAC protocol, with multi-channel and channel hopping capabilities. Time slotted access increases the potential throughput

that can be achieved, by eliminating collision among competing nodes, and provides deterministic latency to applications. Multi-channel allows more nodes to exchange their frames at the same time (i.e., in the same time slot), by using different channel offsets. Hence, it increases the network capacity. In addition, channel hopping mitigates the effects of interference and multipath fading, thus improving the communication reliability. Hence, TSCH provides increased network capacity, high reliability and predictable latency, while maintaining very low duty cycles (i.e., energy efficiency) thanks to the time slotted access mode. TSCH is also topology independent as it can be used to form any network topology (e.g., star, tree, partial mesh or full mesh). It is particularly well-suited for multi-hop networks where frequency hopping allows for efficient use of the available resources.

In this thesis we focus on the TSCH MAC behavior mode. Hence, in the next section we provide a mored detailed description of it.

9.2.1 Time Slotted Channel Hopping (TSCH) mode

In TSCH mode nodes synchronize on a periodic slotframe (Fig. 9.1) consisting of a number of timeslots (Fig. 9.2). Each timeslot allows a node to send a maximum-size data frame and receive the related acknowledgement. If the latter is not received, the retransmission of the data frame is deferred to the next time slot assigned to the same (sender-destination) couple of nodes.

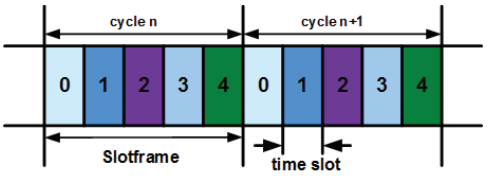


FIGURE 9.1: Slotframe

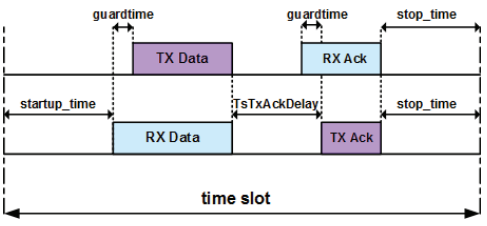


FIGURE 9.2: Timeslot

One of the main characteristics of TSCH is the multi-channel support, based on channel hopping. In principle 16 different channels are available for communication. Each channel is identified by a *channelOffset*, i.e. an integer value in the range [0, 15]. However, some of these frequencies could be blacklisted because of low quality channel. In TSCH a link is defined as the pairwise assignment of a directed communication between devices in a given timeslot on a given channel offset [20]. Hence, a link between two communicating nodes can be represented by a couple specifying the timeslot in the slotframe and the channel offset used by the nodes in that timeslot. Let $[n,\ channelOffset]$ denote a link between two nodes. Then, the frequency f, to be used for communication in timeslot n of the slotframe is derived as follows

$$f = F[(ASN + channelOffset)\%N_{channels}] \tag{9.1}$$

where ASN is the *Absolute Slot Number*, defined as the total number of timeslots elapsed since the start of the network and % is the modulo operator. The ASN increments globally in the network, at every timeslot, and is thus used by nodes as timeslot counter. Function F can be implemented as a lookup table. Thanks to the multi-channel mechanism several simultaneous communications can take place in the same timeslot, provided that different communications occur on different channel offsets. Also, Equation 9.1 implements the channel hopping mechanism by returning a different frequency for the same link at different timeslots. This assures that during time all the available channels are used for communications in a link and, hence, allows to mitigate the negative effect of external interference.

Figure 9.3 shows a possible link schedule for data collection in a simple sensor network with a tree topology. For simplicity, we assume that the slotframe consists of 4 timeslots and there are only 5 channel offsets available. We can see that, thanks to the multi-channel approach used by TSCH, 8 transmissions are accommodated in a time interval corresponding to 4 timeslots. In the allocation shown in Figure 9.3 all the links but one are *dedicated* links, i.e. allocated to a single (sender-destination) couple. TSCH also allows *shared* links, i.e. links intentionally allocated to more senders for transmission to the same destination. This is the case of the link [1, 0] allocated to sender nodes E and G. Shared links plays a key role in TSCH networks since they can be used to exchange routing/scheduling information (e.g. to bootstrap the network), provide basic

connectivity to nodes when dedicated links are not available or broken and add flexibility to the network.

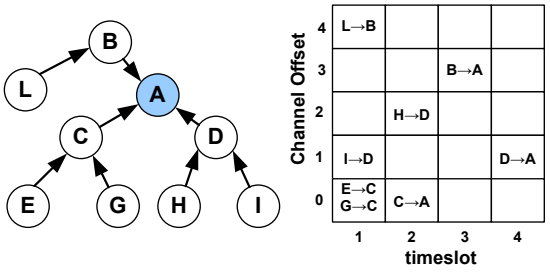


FIGURE 9.3: A possible link schedule for a WSN with a tree topology. Both dedicated and shared links (link [1, 0]) are used.

Since shared links are available to more sender nodes, the standard defines a CSMA/CA algorithm to regulate concurrent access to them and reduce the probability of repeated collisions. In chapter 11 of this thesis we focus on the TSCH CSMA/CA algorithm.

A key element in TSCH is the link schedule, i.e., the assignment of links to nodes for data transmissions. Of course, neighboring nodes may interfere and, hence, they should not be allowed to transmit in the same timeslot and with the same channel offset. The multi-channel mechanism makes the link scheduling problem easier with respect to the traditional scenario where a single channel is used. However, finding out an optimal schedule may not be a trivial task, especially in large networks with multi-hop topology. The problem is even more challenging in dynamic networks where the topology changes over time (e.g., due to mobile nodes). It may be worthwhile emphasizing here that the derivation of an appropriate link schedule is out of the scope of the 802.15.4e standard. The latter just defines mechanisms to execute a link schedule, however, it does not specify how to derive such a schedule. This is left to upper layers.

9.3 Our contribution

TSCH has received strong attention from the research community since its release. Researches have focused mainly on link scheduling [35][36][37], i.e. on the efficient assignment of dedicated links to nodes for communication. Palattella et al. proposed a centralized scheduling algorithm taking into consideration both network topology and traffic conditions of single nodes [35]. Conversely, Tinka et al. [36] and Accettura et al. [37] focused on distributed scheduling algorithms. Watteyne et al. [124] addressed

the interaction between TSCH and Internet-of-Things protocols while Stanislowski et al. [125] studied synchronization issues in TSCH networks. The majority of studies on TSCH consider a fully operative network. Conversely, in this thesis we investigate the mechanisms offered by TSCH to bootstrap/build the network. Specifically, in chapter 10 we analyze the network formation process of IEEE 802.15.4e TSCH network. We define a simple *random-based network advertisement algorithm* and analyze its performance, through both analysis and simulations, in terms of joining time, i.e. the total time taken by a new device to join the TSCH network. Then, in chapter 11, we focus on TSCH shared links that can be used to exchange routing/scheduling information (e.g. to bootstrap the network), provide basic connectivity to nodes when dedicated links are not available or broken and add flexibility to the network. We analyze the CSMA/CA algorithm used by TSCH nodes to concurrently access shared slots. Specifically, we develop an analytical model of TSCH CSMA/CA, based on Discrete Time Markov Chains (DTMC) and use it to predict the performance experienced by nodes when accessing shared links. Since capture effect - i.e. the ability of some radios to correctly receive a strong signal from one transmitter, despite significant interference from other transmitters - has a significant impact on the performance of real wireless networks, we also consider this aspect in our model. We validate the model through both simulation experiments and measurements in a real testbed. The obtained results clearly show the limitations of the TSCH CSMA/CA algorithm and the impact of different parameters on its performance. Also, it is shown that capture effect significantly improves the performance of the algorithm.

Chapter 10

IEEE 802.15.4e TSCH network formation process

10.1 Introduction

In this chapter we focus on the TSCH network formation process. This process is strongly influenced by the policy used to announce the network presence, which is based on sending special messages named *Enhanced Beacons* (EBs). However, the standard does not specify any advertising policy as it is under the responsibility of a layer above the MAC layer.

We define a *random-based advertisement* algorithm that is a generalization of the algorithm used in [124]. We also analyze the considered algorithm in terms of *joining time*, i.e., total time taken by a device to join the network. Since a device has to keep the radio always on, while connecting to the network, the joining time is also a measure of the energy spent by the device to connect. To evaluate the performance of the considered advertisement algorithm we rely both on analysis and simulation, and investigate its sensitiveness to different parameters, such as number of used channel offsets, node density, packet error rate, and number of available frequencies. We found that the joining time is strongly influenced by the number of channels used for advertising EB messages. Using more channels reduces the joining time (and, hence, the energy spent to join the network). On the other side, using more channels for advertising increases the number of transmitted EBs. Hence, it reduces the bandwidth available for data transmissions and

increases the energy spent for advertising the network. However, we found that, typically, it is not necessary using a large number of different channels, unless the network density is extremely high.

The remainder of this chapter is organized as follows. In section 10.2 we define our network advertisement algorithm. In section 10.3 we model the network formation process through a discrete-time Markov chain. In section 10.4 we discuss our analytical and simulation results. Finally, we draw some conclusions in section 10.5.

10.2 TSCH PAN Formation

Like the original 802.15.4 standard [10], TSCH supports two classes of devices, namely *Full Function Devices (FFDs)* and *Reduced Function Device (RFDs)*. FFDs implement all the functionalities defined in the standard and can act as network coordinators. Instead, RFDs implement only a subset of functionalities and cannot act as coordinators.

The network formation starts when a FFD, typically the PAN coordinator, advertises the network presence by sending *Enhanced Beacons* (EBs) at regular times. EB messages are special TSCH frames containing the following information.

1. *Synchronization information* (allows new devices to synchronize to the network);
2. *Channel hopping information* (allows new devices to learn the channel hopping sequence);
3. *Timeslot information* (describes when to expect a frame transmission and when to send an acknowledgment);
4. *Initial link and slotframe information* (allows new devices to know: (i) when to listen for transmissions from the advertising device, and (ii) when to transmit to the advertising device).

A device wishing to join the network starts scanning for possible EB messages. As soon as it receives a valid EB message from an advertiser node, the MAC layer notifies the higher layer. The latter initializes the slotframe and links, by exploiting the information in the received EB message, and switches the device into TSCH mode. At this point the

device is connected to the network. Then, the device typically goes through a procedure aimed to allocate communication resources (i.e., slotframes and links) to the joining device. This procedure may also include a security handshake to mutually authenticate the joining device, configure encryption keys, and configure routing information. The mechanism and rules for setting up communication resources and configure security and routing policies are not defined in the 802.15.4e standard, as they are under the responsibility of the higher layers (e.g., the application or network layer). Once connected and configured appropriately, devices may send EB messages, on their turn, to announce the network presence.

The EB advertising policy is not part of the 802.15.4e standard as it is under the responsibility of upper layers (e.g., the application layer). To define an advertising policy we need to decide *how* to advertise (i.e., which nodes should send EB messages) and *when* to advertise (i.e., the rate at which EB messages should be sent by advertiser nodes). The simplest idea is letting all (full function) devices, that have already joined the network, to advertise EB messages. As far as the advertising rate, there are two contrasting requirements. On one hand, the advertisement rate should be as high as possible to allow devices to join the network very quickly and save energy (it may be worthwhile pointing out that, while waiting for a valid EB message, the joining device must kept the radio always on). On the other hand, sending EB messages too often consumes bandwidth and energy at advertiser nodes. Also, in a real setting the advertising rate should vary dynamically, depending on the operating conditions (e.g., node density, packet error rate, available energy). To the purposes of our analysis we define a simple *random-based advertisement* algorithm that is a generalization of the algorithm used in the TSCH implementation described in [124].

10.2.1 Random-based Advertisemt Algorithm

In the random-based advertisement algorithm considered here *all* FFDs that have already joined the network act as advertiser nodes and broadcast periodic EB messages to announce the network. Each node is assigned, through the link scheduling algorithm, a link in the slotframe given by $[timeslot, channelOffset]$ for the transmission of EB messages. Timeslots devoted to EB advertisement repeat periodically, with a period T_{EB}. For any advertiser node, if T_{EB} (expressed in number of slots) and the number of

available channels N_c are relatively prime, the translation function of the TSCH multi-hopping mechanism ensures that EB messages are transmitted on different channels in subsequent cycles, and, above all, all the available N_c channels are used over N_c cycles.

A collision occurs during the transmission of an EB message, in a certain link, if two or more (advertiser) nodes transmit simultaneously. To reduce the collision probability each advertiser node transmits its EB message, at a scheduled link, with a probability p_{EB} (and refrains with probability $1 - p_{EB}$). The appropriate p_{EB} value is derived autonomously by each node, depending on the local operating conditions (i.e., number of neighbors), in such a way to minimize the collision probability. Hence, different nodes generally use different p_{EB} values. In Section 10.3 we derive the optimal value for p_{EB}.

10.3 Performance Analysis

In this section, we model the node connection process, based on the advertisement algorithm described in the previous section, through a *discrete-time Markov chain*, and derive the *joining time*, i.e., the total time taken by a node to connect to the network. We also derive the optimal p_{EB} value to be used by an advertiser node. We consider a single node *u,* willing to join the network (throughout referred as the *joining node*) and assume that there are N advertiser nodes from which node u can potentially receive an EB message. For simplicity, we assume that the advertisement period T_{EB} is the same for all the advertiser nodes (T_{EB} and the number of available channels N_c are assumed to be relatively prime). In our analysis we consider a *worst-case* scenario where all the advertiser nodes send their EB messages in the *same timeslot,* and using the *same channel offset,* at each period. In addition, they all are neighbors of each other.

In the considered scenario a simultaneous transmission of two (or more) advertiser nodes always results in a collision. As mentioned in the previous section, to reduce the collision probability each advertiser node actually transmits EB messages with a probability p_{EB} (and refrains with probability $1 - p_{EB}$). Hence, a successful EB transmission occurs when (**i**) only one advertiser node transmits its EB message and all the remaining $N-1$ nodes refrain, and (**ii**) the transmitted EB message is received correctly by the joining node. Assuming a packet error rate p_{ERR}, the probability that the joining node receives

a valid EB message can be expressed as.

$$P_{valid} = (1 - p_{ERR}) \cdot [N \cdot p_{EB} \cdot (1 - p_{EB})^{N-1}] \tag{10.1}$$

In Equation (10.1), the second term gives the probability that one advertiser node (out of N) transmits its EB message, while the first term is the probability that the transmitted EB message is correctly received by the joining node. It can be shown, through simple algebraic manipulations, that the above probability takes its maximum value at $p^*_{EB} = 1/N$. Hence, hereafter we will assume $p_{EB} = p^*_{EB} = 1/N$.

According to the TSCH network formation algorithm, a node willing to join the network turns on its radio and listens for EB messages on a given channel frequency f. Since T_{EB} (expressed in number of timeslots) and the number of available channels N_c are assumed to be relatively prime, due to the channel hopping mechanism, EB messages are transmitted on different frequencies in subsequent cycles of duration T_{EB} (throughout referred to as EB cycles, or cycles for short). Hence, at a given cycle, the joining node can receive an EB message only if the latter is transmitted on channel f in that cycle.

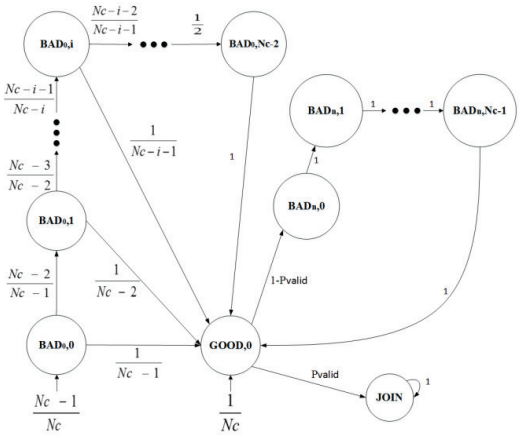


FIGURE 10.1: Discrete Time Markov Chain

We observe the state of the system at the beginning of each EB cycle (i.e., just before the transmission of the EB message). Let B_0 denote the EB cycle when the joining node starts listening for EB messages. We define an EB cycle as GOOD if node u is using the same channel frequency as the advertiser nodes in that cycle, and BAD otherwise. We also distinguish bad cycles as BAD_0, if a good cycle has not been encountered yet, and BAD_n otherwise. The state of node u at the beginning of any EB cycle B_i ($i = 0, 1, 2, \ldots$) can be represented by means of a couple ($cycle\_state$, $seqnum$), where

cycle_state may be GOOD, BAD_0, or BAD_n, while *seqnum* specifies the number of consecutive cycles already experienced in that state. We also consider a special state, referred to as JOIN where node u moves as soon as it received a valid EB message.

Figure 10.1 shows how the state of node u evolves over time. Let N_c denote the number of channels available for communication. Assuming that the joining node has no information about the frequency used by the advertiser nodes, initially it starts either in state (GOOD, 0), with probability $1/N_C$, or in state (BAD_0, 0), with probability $(N_c - 1)/N_C$.

Assuming that the joining node is initially in state (BAD_0, 0), in the next step it moves to state (GOOD, 0), if it encounters a good EB cycle, which occurs with probability $1/(N_c - 1)$, or to state (BAD_0, 1), if the next EB cycle is bad, which occurs with probability $(N_c - 2)/(N_c - 1)$. Similarly, let us assume that, at a given time, the joining node is in state (BAD_0, i), for $i = 1, 2, \ldots, N_c - 3$. This means that it has already experienced i consecutive bad cycles. Hence, in the next step it will move to state (GOOD, 0), if the next cycle is good, i.e., with probability $1/(N_c - i - 1)$, or to state (BAD_0, $i + 1$) in case of another bad cycle, i.e., with probability $(N_c - i - 2)/(N_c - i - 1)$. If the joining node always experience bad cycles, after $N_c - 2$ steps it reaches state (BAD_0, $N_c - 2$). However, the next cycle will be necessarily a good one, and the joining node will move to state (GOOD, 0).

Let us now consider the evolution from state (GOOD, 0). Since the EB cycle is good, the joining node has an opportunity to receive a valid EB message and complete the joining procedure moving to the absorbing state JOIN, where it remains forever. According to Equation (10.1), the latter transition occurs with a probability P_{valid}. If the joining node misses the EB message (i.e., with probability $1 - P_{valid}$) it has to wait for more $N_c - 1$ cycles to have a new opportunity. Hence, in Figure 10.1, the chain evolves through a sequence of states, each accessed with probability 1, and finally reaches state (GOOD, 0) again. This sequence models a deterministic delay equal to $(N_c - 1)$ EB cycles.

The random process described so far, whose transition state diagram is depicted in Figure 10.1, is a discrete time Markov chain, as (**i**) state transitions occur at discrete times, and (**ii**) the probability to move to a new state only depends on the previous state. Its transition probability matrix **P** can be easily derived by using the transition probabilities shown in Figure 10.1. Through the transition probability matrix, we can

derive the probability $P_{join}(k)$, i.e., the probability that the joining node receives a valid EB message, thus joining the network, after exactly k EB cycles from B_0. To this end, let us denote by Ω the set of states of the Markov chain. We can sort states so that the initial states (GOOD, 0) and (BAD_0, 0) are the first one and the second one in the sequence, and the absorbing state JOIN is the last one in the sequence. Let $\mathbf{v}_0$ denote the initial probability vector and $\mathbf{v}_k$ the probability vector after k EB cycles from $B_0 (k = 0, 1, 2, \ldots.)$. Without losing in generality we can assume

$$\mathbf{v}_0 = [1/N_C,\ (N_c - 1)/N_C, 0,\ \ldots,\ 0]$$

Hence, $\mathbf{v}_k = \mathbf{v}_0 \cdot \mathbf{P}^k$. Let $|\Omega|$ denote the cardinality of Ω, i.e. the total number of states. Since JOIN is the last state in the sequence (according to the selected order) it follows

$$P_{join}\,(k) = v_k\,[|\Omega|] \tag{10.2}$$

Let us indicate as $\mu_{(\text{GOOD, }0)}$ and $\mu_{(\text{BAD, }0)}$ the average time spent by the joining node u to reach state JOIN when it starts from state (GOOD, 0) and (BAD_0, 0), respectively. The computation of $\mu_i,\ \forall i \in \Omega$, can be performed solving the following system of equations, with μ_i as unknowns:

$$\mu_i = 1 + \sum_{j \in \Omega} p_{ij} \cdot \mu_j \tag{10.3}$$

where p_{ij} is the transition probability from state i to state j. Then, the average joining time τ_{join} can be derived as follows:

$$\tau_{join} = \frac{1}{N_c} \cdot \mu_{(\text{GOOD, }0)} + \frac{(N_c - 1)}{N_c} \cdot \mu_{(\text{BAD, }0)} \tag{10.4}$$

10.4 Results

In this section we evaluate the performance of the considered advertisement algorithm in terms of joining time. To this end, we use the analytical formulas derived in the previous section. We also use simulation to (**i**) validate our analytical model, and (**ii**) investigate the algorithm performance when the advertiser nodes use more than one channel offset to transmit their EB messages.

To this end, we implemented the advertisement algorithm using the ns2 simulation tool [54]. We modeled the same scenario considered in the analysis. However, in our simulation experiments we relaxed the assumption that all the advertiser nodes use the same channel offset to transmit their EB messages. Instead, channel offsets are deterministically assigned to advertiser nodes, so as to ensure a fair utilization of all channel offsets. Of course, the number of channel offsets (throughout referred to as N_o), must be less than, or equal to, the number of available channel frequencies N_c. When using more channel offsets, the number of (neighboring) advertiser nodes using the same channel offset reduces with N_o (on average it is equal to N/N_o). Hence, the EB collision probability reduces accordingly. In our experiments p_{EB} is set autonomously by each advertiser node, depending on the number of neighboring advertiser nodes using the same offset.

For each experiment we performed several independent replications. The simulation results shown below are averaged over all replications. We also derived confidence intervals by using the independent replication method and 95% confidence level. However, in our results, confidence intervals are so small that they cannot be appreciated in the presented plots. We found that simulation results almost overlap analytical results in all the considered scenarios. The analytical results, that refer to the case when a single channel offset is used, represent an upper bound for the joining time experienced by connecting nodes. Also, the analytical formulas allows to better understand the behavior of the system.

In the following discussion, unless stated otherwise, we will assume that there are 3 advertiser nodes from which the joining node can potentially receive an EB message i.e., (N= 3), all channels are available for communications (i.e., N_c= 16) and EB messages are never corrupted, i.e., the packet error rate (p_{ERR}) is null. In our analysis we varied each of the above-mentioned parameters, one at a time, to analyze its influence on the performance of the considered advertisement algorithm.

Fig. 10.2 shows the average joining time expressed in number of EB cycles (of duration T_{EB}), as a function of the number of advertiser nodes. We considered different numbers of channel offsets (N_o) used by the advertiser nodes to broadcast their EB messages. As expected, the highest joining time is experienced when a single channel offset is used. In addition, in this case, the average joining time increases very quickly with the

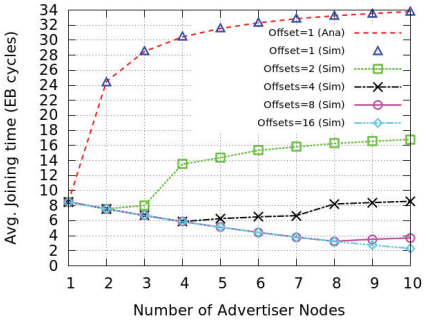


FIGURE 10.2: Average joining time for an increasing number of advertiser nodes ($N_c = 16$, $p_{ERR} = 0\%$.

number of (neighboring) advertiser nodes. This is due to repeated collisions experienced in the transmissions of EB messages (see Equation 10.1). Using more channel offsets for advertisement reduces significantly the average joining time and, hence, the energy spent by devices to join the network. On the other side, it reduces the bandwidth available for data transmissions. In addition, when using N_o different channel offsets, on average, N_o EB messages are transmitted - instead of one - at each EB cycle, since advertiser nodes transmit EB messages with a probability p_{EB} depending on the number of neighbors using the same channel offset. Hence, the total energy spent to advertise the network presence increases with N_o.

Fig. 10.2 also shows that increasing the value of N_o (e.g., passing from 2 to 4, or from 4 to 8, as in Fig. 10.2) is beneficial only if the number of neighboring advertiser nodes is larger than the current number of channel offsets. For this reason, in Fig. 6, curves corresponding to different values of $N_o (N_o = 2)$ overlap in the first part. In conclusion, using many different channel offsets (e.g., more than 4) may be useful only when the network is extremely dense and, thus, there are many neighboring advertiser nodes.

In the previous set of experiments we assumed that EBs are never corrupted by transmission errors. Of course, this is an optimistic assumption. Fig. 10.3 shows the influence of the packet error rate on the average joining time when there are 3 advertiser nodes from which the joining node can potentially receive an EB message (N= 3) and all channel frequencies are available (N_c= 16). As expected, the joining time increases as the wireless medium becomes more and more unreliable. The amount of this increase strongly depends on the number of channel offsets N_o.

When the packet error rate passes from 0% to 30%, the joining time increases by about 55% with a single offset, while the increase is approximately 25% and 15% with 2 or

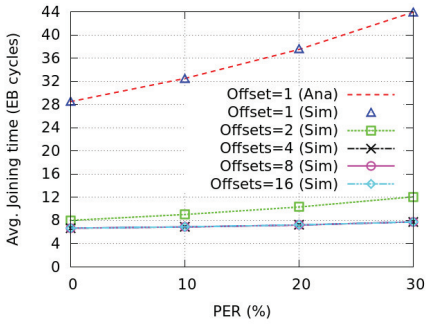


FIGURE 10.3: Average joining time for different packet error rates ($N = 3,\ N_c = 16$).

more channel offsets. This is because (**i**) the percentage of non-valid EB transmissions, due to collisions, is higher when the number of channel offset is low (e.g., $N_o = 1$, see Equation (10.1)) and (**ii**) the time a joining node has to wait, after each non-valid EB transmission, to have a new opportunity, decreases as N_o increases. Hence, the effects of transmission errors are much more severe when using a single channel offset.

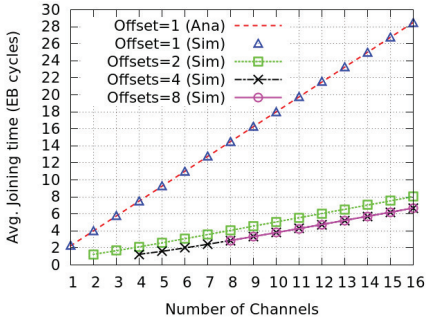


FIGURE 10.4: Average joining time for a varying number of available channels $N = 3,\ PER = 0\%$.

So far we have assumed that all the available channel frequencies can be used for advertising EBs. In the next set of experiments we assume that only a subset of channel frequencies are used for this purpose (we also assume that joining nodes are aware of the subset of frequencies used for EB advertisement). Fig. 10.4 shows that, for any number of channel offsets, the average joining time increases linearly with the number of channel frequencies N_c used for advertisement. Obviously, the number of channel offsets must be lower than, or equal to, the number of channel frequencies (for this reason some curves start from a given point in Fig. 10.4). The results show that, using only a subset of channel frequencies for advertisement would reduce significantly the joining time. However, joining nodes should be aware of the subset of frequencies that is used. Also, they should start listening on one of the frequencies in the subset.

10.5 Conclusions

In this chapter we have investigated the network formation process in 802.15.4e TSCH WSNs. Since the standard does not specify any algorithm for advertising Enhanced Beacons in order to announce the network presence, we have defined a simple, random-based advertisement algorithm. We have analyzed the proposed algorithm in terms of joining time, by analysis and simulation. We have found that the joining time is mainly influenced by the number of channel offsets used for advertising Enhanced Beacons. Using more channel offsets reduces significantly the average joining time and, hence, the energy spent by devices to join the network. On the other side, it increases the number of transmitted EB messages and, hence, the energy spent by advertiser nodes. Moreover, using a large number of different channel offsets for advertisement may be beneficial, in terms of reduced joining time, only when the network density is extremely high.

Chapter 11

IEEE 802.15.4e TSCH CSMA/CA Algorithm

11.1 Introduction

In this chapter we focus on TSCH shared links and, specifically, on the CSMA/CA algorithm used by TSCH nodes to concurrently access shared links. Shared links are expected to play a key role in future TSCH networks since they will be used - in combination with, or as an alternative to, dedicated links - during the network formation process (e.g. to exchange routing/scheduling information) and in case of network failure (e.g. when a free-of-collision communication schedule is not available). We develop an analytical model of TSCH CSMA/CA, based on Discrete Time Markov Chains (DTMC) and use it to predict the performance experienced by nodes when accessing shared links. Since capture effect - i.e. the ability of some radios to correctly receive a strong signal from one transmitter, despite significant interference from other transmitters [38] - has a significant impact on the performance of real wireless networks, we also consider this aspect in our model. We validate our model through simulation experiments and measurements in a real network. Our results show that the capture effect can significantly increase the packet delivery probability and decrease both packet latency and energy consumption of nodes using shared links. To summarize, in this chapter we provide the following contributions:

- We provide a very accurate analytical model of the TSCH CSMA/CA algorithm. The analysis allows to understand the behavior of the CSMA/CA algorithm in great detail. Also, the model represents a powerful tool for industrial practitioners to both perform a fine grained tuning of TSCH networks and to validate implementations of TSCH MAC on real hardware.
- A comprehensive evaluation of the TSCH CSMA/CA algorithm based on analysis, simulation and experiments on a real testbed is performed. The results clearly show the limitations of the algorithm and the impact of different parameters on its performance.
- We explicitly consider capture effect while modeling the CSMA/CA algorithm. The analytical results, validated by experiments, show that it significantly improves the performance of the algorithm since it turns a considerable percentage of collisions into successful transmissions.

The rest of the chapter is organized as follows. Section 11.2 describes the TSCH CSMA/CA algorithm used to access shared links. Section 11.3 describes the analytical model of TSCH CSMA/CA. Section 11.4 presents the analytical and experimental results. Finally, Section 11.5 draws some conclusions.

11.2 TSCH CSMA/CA Algorithm

Since shared links are available to more sender nodes, the standard defines a CSMA/CA algorithm to regulate concurrent access to them and reduce the probability of repeated collisions. It works as follows. Upon receiving a data frame destined to node r, a sender node s waits for the arrival of the first (dedicated or shared) link assigned to (s, r), and, then, transmits its data frame. If a shared link was used, and the transmission was unsuccessful (i.e. the acknowledgment was not received), very likely a collision occurred. Hence, the CSMA algorithm is executed by node s to avoid repeated collisions. Specifically, the following steps are performed.

1. A set of state variables is initialized, namely the number of retransmissions carried out for the on-going frame ($NB = 0$) and the backoff exponent ($BE = macMinBE$).

2. A random number $w \in [0,\ 2^{BE} - 1]$ is generated.

3. The frame retransmission is deferred for w shared links with destination r, or until a dedicated link with destination r is encountered.

4. If the retransmission occurs in a shared link and it is successful (i.e. the acknowledgement is received), the backoff exponent BE is reset to $macMinBE$ and the algorithm terminates. Instead, if the transmission is unsuccessful, state variables are updated as follows: $NB = NB{+}1$, $BE = min(BE{+}1,\ macMaxBE)$. Finally, if the number of retransmissions for the current frame has exceeded the maximum allowed value (i.e. $NB > macMaxFrameRetries$), the frame is dropped; otherwise the algorithm falls back to step 2.

If the frame retransmission is carried out in a dedicated link, and it is successful, BE is reset to $macMinBE$, unless there are other frames, destined to the same receiver, ready for transmission. In the latter case the value of BE is left unchanged.

11.3 Analytical Model

In this section, we develop our analytical model of the TSCH CSMA/CA algorithm and use it to derive performance metrics of interest such as packet delivery probability, average packet latency and total energy consumption.

We consider a general network topology and we assume that there are N competing nodes trying to transmit a data packet to a common receiver, say node r (e.g. their parent node or the network gateway), using shared timeslots (hereafter referred to as slots, for brevity) and, hence, executing the TSCH CSMA/CA algorithm. Hereafter, the following assumptions hold:

1. Each one of the N competing nodes tries to transmit a *single* data packet to the receiver node r. Also, all the nodes start transmitting their data packet during the same shared slot, namely t_0.

2. There are no dedicated links assigned to the transmitting nodes and the receiver, i.e. nodes can only use shared links to transmit their data packet. Hence, we are modeling a worst-case scenario.

3. The communication channel is ideal, i.e. data/acknowledgement frames are never corrupted, or lost, due to transmission errors.

4. Capture effect is modeled through a function $P_{CE}(n),\ 2 \leq n \leq N$, indicating the probability that node r correctly receives a data packet when n out of the N nodes concurrently transmit their packet during a slot (i.e. $P_{CE}(n)$ represents the probability of capture effect to occur). $P_{CE}(n)$ depends on a number of factors such as the physical network topology, differences in manufacturing of nodes, antennas gains, etc. Hence, $P_{CE}(n)$ has to be characterized through extensive measurements in the real network operating scenario.

The analysis is divided into three parts. First, we perform a single-node analysis, i.e. we focus on one of the N competing nodes and analyze its behavior in detail. Then, basing on the single-node analysis, we derive a Discrete Time Markov Chain (DTMC) describing the behavior of the entire system, i.e. the N competing nodes all together. Finally, we derive the performance metrics of interest using the DTMC.

11.3.1 Single-node analysis

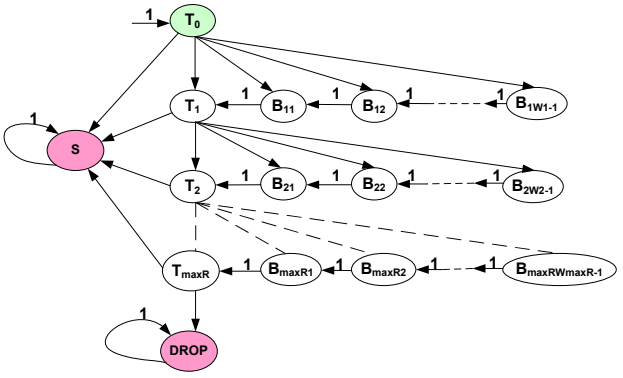


FIGURE 11.1: States of a competing node u over time. State T_0 is the initial state of the node while states S and $DROP$ are its possible final states.

In this section, we focus our attention on one of the N competing nodes, say node u, and derive all the possible states through which it may pass during the transmission of a packet on a shared link. Hereafter, we indicate as t_0 the shared slot at which all the N transmitting nodes start the transmission of their data packet and as $t_1,\ t_2,\ ...$, the sequence of shared slots, following t_0, that can be used by the N competing nodes to transmit packets to node r. Note that slots t_k $(k = 0,\ 1,\ ...)$ do not need to be

consecutive time slots in the slotframe. Also, they may belong to different slotframes. For brevity, in the following we will refer to the *macMaxFrameRetries* parameter as *maxR*.

We observe node u at the beginning of each shared slot t_k ($k = 0,\ 1,\ ...$). Figure 11.1 shows all the possible states of node u over time. Initially, i.e. during slot t_0, node u is in state T_0 and transmits its data packet. In case its packet is successfully received by node r, despite the N concurrent transmissions (i.e. if its packet experiences capture effect), node u moves to state S. Otherwise node u has to retry the transmission of its packet after waiting for a random number $w \in [0, W_1 - 1]$ of shared slots, where $W_1 = 2^{macMinBE}$. In the latter case, since there are W_1 different possible values for w, node u can move to W_1 different states. Specifically, if $w = 0$ the node will retry to transmit its data packet during t_1. Hence, it will move to state T_1. Conversely, if $w > 0$ the node will move to backoff states $B_{1j},\ 1 \leq j \leq W_1 - 1$, meaning that it is at its first retransmission attempt and has to wait for j shared time slots before retransmitting its data packet.

Let us now assume that node u is in one of the states $B_{1j},\ 2 \leq j \leq W_1 - 1$ at the beginning of time slot t_k. Node u will move from state B_{1j} to state B_{1j-1} with probability equal to 1. When node u reaches state B_{11} it will move to state T_1 since **i)** it has finished to wait for the randomly chosen backoff time and **ii)** it can proceed to packet retransmission.

Let us now describe the possible state transitions of node u assuming it is in state T_1 at the beginnning of t_k. If node u successfully transmits its packet, it will move to state S. Otherwise, i.e. if its transmission fails due to collision with other packets, node u will move to one of the states $T_2,\ B_{21},\ B_{22},\ ...,\ B_{2W_2-1}$, where $W_2 = 2^{min(macMinBE+1, macMaxBE)}$, i.e. the states of its second retransmission attempt.

The possible transitions of node u when in one of the states $\{T_i,\ B_{ij}, 2 \leq i \leq maxR - 1, 1 \leq j \leq W_i - 1\}$, can be easily derived using the same line of reasoning presented before and looking at Figure 11.1. Hence, we omit their description. Now, let us focus our attention on state T_{maxR}, i.e. the state corresponding to the last retransmission attempt. There are only two possible state transitions for node u starting from state T_{maxR}. Specifically, node u will move to state S in case of a successful transmission or to state $DROP$, if it experiences a transmission failure. Note that both state S and $DROP$ are absorbing states, i.e. node u performs no other action after entering one

of these states. Also, it can be easily observed that after, at most, L_{max} shared slots from t_0 ($L_{max} = W_1 + W_2 + \ ... \ + W_{maxR} + 1$) node u can only be in one of the two abovementioned states, i.e. the execution of TSCH CSMA/CA surely terminates after L_{max} slots from t_0.

11.3.2 Derivation of the Discrete Time Markov Chain

In this section, basing on the single-node analysis described above, we derive a Discrete Time Markov Chain (DTMC) modeling the behavior of the entire system (i.e. all the N competing nodes). In the following, we will indicate the set of all the possible states of a transmitting node u as $\mathbb{S}_u = \{T_0,\ T_1,\ ...,\ T_{maxR},\ S,\ DROP,\ B_{11},\ ...\}$.

We observe the system at the beginning of each shared slot t_k. The state of the system, at time t_k, can be represented as a map $\mathbb{X}_{t_k}$ whose key values are the elements of set $\mathbb{S}_u$, while mapped values indicate the number of nodes being in the state indicated by their respective key value at time slot t_k. Specifically, $\mathbb{X}_{t_k}[T_0]$ indicates the number of nodes in state T_0 at t_k, $\mathbb{X}_{t_k}[S]$ indicates the number of nodes in state S at t_k, etc. Since N nodes are in the system, $\forall \mathbb{X}_{t_k}$, $\sum_{g \in \mathbb{S}_u} \mathbb{X}_{t_k}[g] = N$.

At slot t_0, all the nodes are in state T_0. Hence, the initial state of the system $\mathbb{X}_{t_0}$ is such that $\mathbb{X}_{t_0}[T_0] = N$ while $\mathbb{X}_{t_0}[g] = 0,\ \forall g \neq T_0,\ g \in \mathbb{S}_u$. Our goal is to derive all the possible states of the system during time as well as the probability of transition from every state to one another. To this end, we use Algorithm 7 which takes an iterative approach. First, the algorithm focuses on state $\mathbb{X}_{t_0}$ and generates $G_{\mathbb{X}_{t_0}}$, i.e. the set of all the possible states where the system can move to, starting from $\mathbb{X}_{t_0}$. Then, for each new generated state the process of generation of new possible states is repeated. After some time, all the possible system states as well as the state transition probabilities are derived and, hence, the algorithm terminates. Now, we describe all the steps performed by the algorithm in detail.

Algorithm 7 starts by creating sets L_s and F_s (lines 1-2). L_s is the list of system states not yet analyzed by the algorithm, while F_s is the set of *all* the possible system states. Initially (line 3), Algorithm 7 creates state $\mathbb{X}_{t_0}$, i.e. the initial system state, and adds it to both set L_s and set F_s. Then (line 4), the algorithm enters a loop that ends when $L_s = \emptyset$, i.e. when there are no more states to be analyzed. At each step of the loop,

a state $\mathbb{X}_{t_k}$ of the system is analyzed. Specifically, for each state $\mathbb{X}_{t_k}$, the set $G_{\mathbb{X}_{t_k}}$ of all the possible states where the system can evolve to, starting from $\mathbb{X}_{t_k}$ is generated. Different actions are performed by the algorithm to generate $G_{\mathbb{X}_{t_k}}$, depending on the number $n = \sum_{x=0}^{maxR} \mathbb{X}_{t_k}[T_x]$ of nodes concurrently transmitting their packet when the state of the system is $\mathbb{X}_{t_k}$.

In Section 11.3.2.1 we describe the actions performed by the algorithm in case $n \leq 1$, i.e. when there are zero or one nodes transmitting their packet during t_k (lines 7-19). Instead, in Section 11.3.2.2 we analyze the case $n \geq 2$, i.e. when there are two or more transmitting nodes (lines 20-51).

11.3.2.1 Case $n \leq 1$

In this section we describe the actions performed by the algorithm to generate $G_{\mathbb{X}_{t_k}}$ when $n \leq 1$. In this case, since at most one packet is transmitted, no transmission failures occur. This implies that all the nodes in the system change their state in a deterministic way. Thus, there is only one possible state, namely $\mathbb{X}_{t_{k+1}}$ where the system can evolve to, starting from $\mathbb{X}_{t_k}$.

In order to generate $\mathbb{X}_{t_{k+1}}$, the algorithm performs the following actions. First, since only successful transmissions occur, the number of nodes in state S at time t_{k+1}, i.e. $\mathbb{X}_{t_{k+1}}[S]$, is calculated as $\mathbb{X}_{t_k}[S] + n$, i.e. as the sum of all the nodes that have already experienced a successful transmission and the n nodes transmitting at time t_k (line 9). Also, since no failures occur, the number of nodes in state $DROP$ will not change from t_k to t_{k+1}, i.e. $\mathbb{X}_{t_{k+1}}[DROP] = \mathbb{X}_{t_k}[DROP]$ (line 10). Then, the algorithm focuses on nodes in state B_{ij}, $1 \leq i \leq maxR$, $1 \leq j \leq W_i - 1$ during t_k (lines 11-15). Specifically, nodes in state B_{i1} move to T_i (line 12), while nodes in state B_{ij+1} move to B_{ij} (line 13).

In case $\mathbb{X}_{t_{k+1}}$ has not been generated before, it is added to both set L_s and F_s to be analyzed later (line 16). Finally (lines 17-18), the algorithm stores that the probability $P_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}}$ to move from state $\mathbb{X}_{t_k}$ to $\mathbb{X}_{t_{k+1}}$ is equal to 1 since only deterministic state transitions occur. Also, the algorithm calculates and stores the energy $E_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}}$ spent by the system when its state passes from $\mathbb{X}_{t_k}$ to $\mathbb{X}_{t_{k+1}}$. It is given by:

$$E_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}} = n \cdot E_s \tag{11.1}$$

$L_s = \emptyset$
$F_s = \emptyset$
Add $\mathbb{X}_{t_0}$ to L_s and F_s
while$(L_s \neq \emptyset)$
  extract $\mathbb{X}_{t_k}$ from L_s
  $n = \sum_{x=0}^{maxR} \mathbb{X}_{t_k}[T_x]$
  if$(n \leq 1)$ **/* At most one node transmits its packet */**
    create $\mathbb{X}_{t_{k+1}}$
    $\mathbb{X}_{t_{k+1}}[S] = \mathbb{X}_{t_k}[S] + n$
    $\mathbb{X}_{t_{k+1}}[DROP] = \mathbb{X}_{t_k}[DROP]$
    for i in $[1,\ maxR]$
      $\mathbb{X}_{t_{k+1}}[T_i] = \mathbb{X}_{t_k}[B_{i1}]$
      $\mathbb{X}_{t_{k+1}}[B_{ij}] = \mathbb{X}_{t_k}[B_{ij+1}],\ j \in [1,\ W_i - 2]$
      $\mathbb{X}_{t_{k+1}}[B_{iW_i-1}] = 0$
    end for
    if$(\mathbb{X}_{t_{k+1}} \notin F_s)$ Add $\mathbb{X}_{t_{k+1}}$ to L_s, F_s
    $P_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}} = 1$
    $E_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}} = n \cdot E_s$
  end if
  if$(n \geq 2)$ **/* Two or more nodes transmit packets */**
    gen. Ω_{C_g}: $f_i = \mathbb{X}_{t_k}[T_i], \forall i$ **/* When no CE occurs*/**
    for each $C_g \in \Omega_{C_g}$
      create $\mathbb{X}_{t_{k+1}}$
      $\mathbb{X}_{t_{k+1}}[S] = \mathbb{X}_{t_k}[S]$
      $\mathbb{X}_{t_{k+1}}[DROP] = \mathbb{X}_{t_k}[DROP] + \mathbb{X}_{t_k}[T_{maxR}]$
      for i in $[1,\ maxR]$
      $\mathbb{X}_{t_{k+1}}[T_i] = \mathbb{X}_{t_k}[B_{i1}] + c_{i-1_{T_i}}$
      $\mathbb{X}_{t_{k+1}}[B_{ij}] = \mathbb{X}_{t_k}[B_{ij+1}] + c_{i-1_{B_{ij}}}, j \leq W_i - 2$
      $\mathbb{X}_{t_{k+1}}[B_{iW_i-1}] = c_{i-1_{B_{iW_i-1}}}$
      end for
      if$(\mathbb{X}_{t_{k+1}} \notin F_c)$ Add $\mathbb{X}_{t_{k+1}}$ to L_s, F_s
      $P_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}} = (1 - P_{CE}(n)) \cdot P(C_g)$
      $E_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}} = n \cdot E_c$
    end for
    for x in $[0,\ maxR]$ **/* When CE occurs */**
      gen. Ω_{C_g} if $f_x = \mathbb{X}_{t_k}[T_x] - 1,\ f_j = \mathbb{X}_{t_k}[T_j],\ j \neq x$
      for each $C_g \in \Omega_{C_g}$
       create $\mathbb{X}_{t_{k+1}}$
       $\mathbb{X}_{t_{k+1}}[S] = \mathbb{X}_{t_k}[S] + 1$
       $\mathbb{X}_{t_{k+1}}[DROP] = \mathbb{X}_{t_k}[DROP] + f_{maxR}$
       for i in $[1,\ maxR]$
       $\mathbb{X}_{t_{k+1}}[T_i] = \mathbb{X}_{t_k}[B_{i1}] + c_{i-1_{T_i}}$
       $\mathbb{X}_{t_{k+1}}[B_{ij}] = \mathbb{X}_{t_k}[B_{ij+1}] + c_{i-1_{B_{ij}}}, j \leq W_i - 2$
       $\mathbb{X}_{t_{k+1}}[B_{iW_i-1}] = c_{i-1_{B_{iW_i-1}}}$
       end for
       if$(\mathbb{X}_{t_{k+1}} \notin F_c)$ Add $\mathbb{X}_{t_{k+1}}$ to L_s, F_s
       $P_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}} = P_{CE}^x \cdot P(C_g)$
       $E_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}} = E_s + (n-1) \cdot E_c$
      end for
    end for
  end if
end while
return $P,\ E$

Algorithm 7: Model generation

where $E_s = P_{tx} \cdot D_{tx} + P_{rx} \cdot D_{ack}$ is the energy spent to transmit a data packet and receive the respective acknowledgement and P_{tx} (P_{rx}) is the radio power in transmit (receive) mode, while D_{tx} (D_{ack}) is the data (ack) transmission time.

11.3.2.2 Case $n \geq 2$

Now, we describe the actions performed by the algorithm to generate $G_{\mathbb{X}_{t_k}}$ when $n \geq 2$, i.e. when there are two or more nodes transmitting their packet during t_k and, hence, transmission failures occur. In such a case, there are different states that can be entered by the system starting from $\mathbb{X}_{t_k}$. This is because nodes experiencing a transmission failure will change their state in a stochastic manner. Specifically, each node in a state $T_i,\ 0 \leq i \leq maxR - 1$, moves randomly to one of the states $\{T_{i+1},\ B_{i+1x},\ 1 \leq x \leq W_{i+1} - 1\}$, when it experiences a transmission failure.

To correctly generate $G_{\mathbb{X}_{t_k}}$ we need to derive all the possible ways in which nodes experiencing a transmission failure during t_k can evolve. To this end, let us denote by f_0, f_1, ..., f_{maxR} the number of nodes in state T_0, T_1, ..., T_{maxR} respectively, experiencing a transmission failure during t_k. The f_{maxR} nodes in state T_{maxR} will move to state $DROP$ since they exceeded the maximum number of retransmissions for a packet. Instead, the f_i nodes in state $T_i,\ 0 \leq i \leq maxR - 1$, randomly spread over the states corresponding to the $(i+1)$-th transmission attempt, as shown in Figure 11.2.

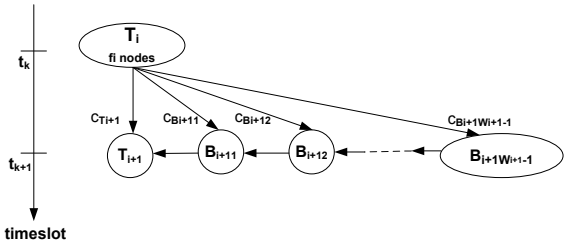


FIGURE 11.2: f_i nodes moving randomly to the states of the $(i+1)$-tx attempt

Now, we focus on the f_i nodes in state T_i and derive all the possible ways they can spread over states $\{T_{i+1},\ B_{i+1x},\ 1 \leq x \leq W_{i+1} - 1\}$. With reference to Figure 11.2, let us denote by $c = \{c_{T_{i+1}},\ c_{B_{i+11}},\ ...,\ c_{B_{i+1W_{i+1}-1}}\}$ a possible distribution of nodes over states $\{T_{i+1},\ B_{i+1x},\ 1 \leq x \leq W_{i+1} - 1\}$ so that $c_{T_{i+1}}$ nodes will be in state T_{i+1}, $c_{B_{i+11}}$ nodes in state B_{i+11} and so on. Obviously, $c_{T_{i+1}} + c_{B_{i+11}} + \ ... \ + c_{B_{i+1W_{i+1}-1}} = f_i$.

It can be observed that c is one of the possible *weak compositions* of integer f_i into W_{i+1} parts. A weak composition of an integer f_i into W_{i+1} parts is a sequence of W_{i+1} non-negative integers that sum to f_i. It can be demonstrated that there are $\binom{f_i+W_{i+1}-1}{W_{i+1}-1}$ weak compositions of f_i into W_{i+1} parts. Let us indicate as $C_i(f_i)$ the set of all the possible weak compositions of integer f_i into W_{i+1} parts, i.e. the set of all the possible ways in which the f_i nodes in state T_i can spread over the W_{i+1} states of transmission attempt $i+1$. The probability $P(c)$ of $c \in C_i(f_i)$ to occur can be calculated as:

$$P(c) = \binom{f_i}{c_{T_{i+1}}\ c_{B_{i+11}}\ ...\ c_{B_{i+1W_{i+1}-1}}} \cdot \left(\frac{1}{W_{i+1}}\right)^{f_i} \tag{11.2}$$

where $\binom{-}{-}$ is the multinomial coefficient. The multinomial coefficient provides the total number of ways the f_i nodes in state T_i can distribute at the next transmission attempt. The second term of eq. 11.2 gives the probability of each single outcome to occur.

It can be observed that the f_0 nodes in state T_0 can move according to any of the compositions in set $C_0(f_0)$, the f_1 nodes in state T_1 can move in any of the compositions in set $C_1(f_1)$, and so on. Also, it must be pointed out that transitions of nodes in state T_i are completely independent from transitions of nodes in state $T_j,\ j \neq i$.

Let us indicate as stochastic system transition $C_g = \{c_0,\ c_1,\ ...,\ c_{maxR-1}\}$ the case in which the f_0 nodes in state T_0 distribute over states of transmission attempt 1 according to composition c_0, the f_1 nodes in state T_1 distribute over states of transmission attempt 2 according to c_1, etc, with $c_0 \in C_0(f_0)$, $c_1 \in C_1(f_1)$, ... The set Ω_{C_g} of all the possible stochastic system transitions C_g that can occur when f_0 nodes in state T_0 fail their transmission, f_1 nodes in state T_1 fail their transmission, etc., can be derived by performing the cartesian product between sets $C_i(f_i)$, $1 \leq i \leq maxR-1$, i.e.

$$\Omega_{C_g} = \bigtimes_{i=0}^{maxR-1} C_i(f_i) \tag{11.3}$$

Also, the probability $P(C_g)$ of a stochastic system transition $C_g \in \Omega_{C_g}$ to occur can be calculated as

$$P(C_g) = \prod_{i=0}^{maxR-1} P(c_i) \tag{11.4}$$

where $P(c_i)$ is calculated through eq. 11.2. Note that the occurrence of each $C_g \in \Omega_{C_g}$ causes the system to evolve in a different state.

When n competing nodes transmit simultaneously during slot t_k a successful transmission may or may not occur, depending on the capture effect. Of course, to correctly generate $G_{\mathbb{X}_{t_k}}$, the algorithm needs to consider both the case when capture effect does not occur (lines 21-34), and when it occurs (lines 35-50). We recall that $P_{CE}(n)$, $2 \leq n \leq N$, indicates the probability that capture effect (CE) occurs when n nodes transmit their packet in the same slot.

We first focus on the case when no CE occurs (lines 21-34), that happens with probability equal to $(1 - P_{CE}(n))$. In such a situation, all the n nodes transmitting during t_k experience a transmission failure. Hence, the number f_i of nodes experiencing a failure while in state T_i is $f_i = \mathbb{X}_{t_k}[T_i]$, $\forall i$. Algorithm 7 generates the set Ω_{C_g} of all the stochastic system transitions that can occur when $f_i = \mathbb{X}_{t_k}[T_i]$, $\forall i$ (line 21) by using eq. 11.3. Then, for each $C_g \in \Omega_{C_g}$ a new state $\mathbb{X}_{t_{k+1}}$ has to be generated (lines 23-33). The following actions are performed to generate $\mathbb{X}_{t_{k+1}}$. Since no successes occur during t_k in this case, $\mathbb{X}_{t_{k+1}}[S] = \mathbb{X}_{t_k}[S]$ (line 24). Second, since all the nodes in the transmission state of the last retransmission attempt will drop their packet, $\mathbb{X}_{t_{k+1}}[DROP] = \mathbb{X}_{t_k}[DROP] + \mathbb{X}_{t_k}[T_{maxR}]$ (line 25). Then, for each retransmission attempt $i \in [1,\ maxR]$ the following operations are performed. First (line 27), the number of nodes in state T_i at time t_{k+1} is calculated as the sum of $\mathbb{X}_{t_k}[B_{i1}]$, i.e. the nodes that have finished to wait for the chosen backoff time and c_{i-1T_i} with $c_{i-1} \in C_g$, i.e. the nodes in state T_{i-1} at time t_k that move to state T_i in case C_g occurs. A similar operation is performed also for states B_{ij}, $i \in [1,\ maxR]$, $j \in [1,\ W_i - 1]$ (lines 28-29). Then, if state $\mathbb{X}_{t_{k+1}}$ has not been generated before, it is added to both set L_s and F_s for further analysis (line 31). Finally, the algorithm stores that the probability $P_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}}$ to go from state $\mathbb{X}_{t_k}$ to $\mathbb{X}_{t_{k+1}}$ is equal to $(1-P_{CE}(n)) \cdot P(C_g)$, i.e. state $\mathbb{X}_{t_{k+1}}$ is generated from state $\mathbb{X}_{t_k}$ in case no CE occurs and the failuring nodes move in the way reported by C_g. In addition, the total energy $E_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}}$ consumed by the system when it passes from $\mathbb{X}_{t_k}$ to $\mathbb{X}_{t_{k+1}}$ is calculated as

$$E_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}} = n \cdot E_c \tag{11.5}$$

where $E_c = P_{tx} \cdot D_{tx} + P_{rx} \cdot D_{to}$ and D_{to} is the timeout duration.

Let us consider now the case of capture effect (lines 35-50), that occurs with probability $P_{CE}(n)$. In this case, one of the n transmitting nodes, namely u, successfully transmits its packet and moves to state S, while all the other $n-1$ nodes experience a transmission

failure and have to retransmit their packet. The algorithm considers $maxR+1$ different situations (line 35) corresponging to all the possible states of node u, i.e. T_0, T_1, ..., T_{maxR}. Let us consider the case in which node u is one of the nodes in state T_x ($0 \leq x \leq maxR$). The probability P_{CE}^x, that node u was in state T_x at time t_k can be calculated as:

$$P_{CE}^x = P_{CE}(n) \cdot \left(\frac{\mathbb{X}_{t_k}[T_x]}{n} \right) \tag{11.6}$$

with $\sum_{x=0}^{maxR} P_{CE}^x = P_{CE}(n)$.

While u moves to S, all the other $\mathbb{X}_{t_k}[T_x] - 1$ nodes in state T_x, as well as the nodes in state $T_j,\ j \neq x$, experience a transmission failure. Hence, in this case, $f_x = \mathbb{X}_{t_k}[T_x] - 1$ while $f_j = \mathbb{X}_{t_k}[T_j],\ j \neq x$.

For each transmitting state $T_x,\ 0 \leq x \leq maxR$ (line 35), the algorithm performs the following steps. First, it generates Ω_{C_g} when $f_x = \mathbb{X}_{t_k}[T_x] - 1$ and $f_j = \mathbb{X}_{t_k}[T_j],\ j \neq x$, i.e. the set of all the possible stochastic system transitions that can occur when one of the nodes in state T_x successfully transmits its data packet. Then, for each C_g, a new state $\mathbb{X}_{t_{k+1}}$ is generated. Since a successful transmission has occurred $\mathbb{X}_{t_{k+1}}[S] = \mathbb{X}_{t_k}[S] + 1$ (line 39). Second, all the nodes in state T_{maxR} that fail their transmission are moved to state $DROP$ (line 40). For each retransmission attempt $i \in [1,\ maxR]$ the following steps are performed (lines 41-45). The number of nodes in state T_i, at time t_{k+1}, is calculated as the sum of $\mathbb{X}_{t_k}[B_{i1}]$ (i.e. the nodes that exhausted their backoff time) and c_{i-1T_i} with $c_{i-1} \in C_g$, (i.e. the nodes in state T_{i-1} at time t_k that move to state T_i in case the transition C_g occurs) (line 42). A similar operation is performed also for states B_{ij}, $i \in [1,\ maxR]$, $j \in [1,\ W_i - 1]$ (lines 43-44). Then (line 46), if state $\mathbb{X}_{t_{k+1}}$ has not been generated before it is added to both set L_s and F_s to be analyzed. Finally, the algorithm stores that the probability $P_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}}$ to go from state $\mathbb{X}_{t_k}$ to $\mathbb{X}_{t_{k+1}}$ is equal to $P_{CE}^x \cdot P(C_g)$, i.e. the system transits from $\mathbb{X}_{t_k}$to $\mathbb{X}_{t_{k+1}}$ in case node u is one of the nodes in state T_x and the stochastic network transition C_g occur. In addition, the energy $E_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}}$ spent by the network when its state passes from $\mathbb{X}_{t_k}$ to $\mathbb{X}_{t_{k+1}}$ is calculated as

$$E_{\mathbb{X}_{t_k}\mathbb{X}_{t_{k+1}}} = E_s + (n-1) \cdot E_c \tag{11.7}$$

since one node succeds in transmitting its packet while all the other $n-1$ transmitting nodes experience a failure.

Algorithm 7 terminates its execution by returning matrices P and E. P is the transition probability matrix, i.e. for each couple of states $(X,\ Y)$, $X,\ Y \in F_s$, P_{XY} gives the probability that the system state changes from X to Y passing from time t_k to time t_{k+1}. Instead, each entry E_{XY} of matrix E with $X,\ Y \in F_s$ indicates the energy spent by the network when its state passes from X to Y. Note that since **i)** state transitions occur at discrete times, and **ii)** the probability to move to a new state only depends on the previous state, the random process described so far, is a Discrete Time Markov Chain (DTMC).

11.3.3 Derivation of performance metrics

In this section we use matrices P and E and the theory of DTMCs to derive the following performance metrics:

- Delivery Probability ($\overline{D}$): defined as the ratio between the number of data frames correctly received by node r and the total number of transmitted frames.
- Avg. Packet Latency ($\overline{L}$): defined as the average time (expressed in timeslots) between the start of a data frame transmission and its correct reception by node r.
- Energy Consumption ($\overline{E}$): defined as the average total energy consumed by all the nodes in the network to transmit their data frames.

We sort the states of the Markov Chain so that the initial state of the system, i.e. $\mathbb{X}_{t_0}$, is the first one in the sequence. Also, let v_0 be the initial probability vector and v_k, $k \geq 1$ the probability vector related to timeslot t_k. With no loss of generality we can assume that $v_0 = [1,\ 0,\ ...,\ 0]$, thus $v_k = v_0 \cdot P^k$.

To derive the delivery probability $\overline{D}$ we observe that the execution of the CSMA/CA algorithm terminates after at most L_{max} slots (from t_0). Specifically, at slot $t_{L_{max}}$, each transmitting node can be only in state S or $DROP$. It follows that there are $N+1$ possible absorbing states corresponding to the $N+1$ different ways the N transmitting nodes can be spread over the two states S and $DROP$. We indicate the final states of the network as $\mathbb{X}_f^i,\ 0 \leq i \leq N$, where $\mathbb{X}_f^i$ describes a state in which i packets are correctly received by node r while $N-i$ packets are dropped by transmitting nodes, i.e.

$\mathbb{X}_f^i[S] = i \ \wedge \ \mathbb{X}_f^i[DROP] = N - i \ \wedge \ \mathbb{X}_f^i[g] = 0, \forall g \in \mathbb{S}_u, \ g \neq S, \ DROP$. Indicating as $p_i, \ 0 \leq i \leq N$, the positions of states $\mathbb{X}_f^i$ in the probability vector, $\overline{D}$ can be calculated as:

$$\overline{D} = \frac{1}{N} \cdot \sum_{i=0}^{N} i \cdot v_{L_{max}}[p_i] \tag{11.8}$$

where $v_{L_{max}} = v_0 \cdot P^{L_{max}}$ is the probability vector after L_{max} timeslots from t_0.

Let us focus now on the average packet latency $\overline{L}$. We first derive the probability $P(t)$ that one packet is correctly received by node r at slot t. Let us denote by $\Omega_{s_i} = \{\mathbb{X}_{t_k} \in F_s : \ \mathbb{X}_{t_k}[S] = i\}, \ 0 \leq i \leq N$, the subset of system states where i packets have been successfully received by node r. We indicate as $P\{\Omega_{s_i}^t\}$ the probability for the system to be in any of the states of set Ω_{s_i} at the beginning of timeslot t, i.e. $P\{\Omega_{s_i}^t\} = \sum_{\mathbb{X}_{t_k} \in \Omega_{s_i}} v_t[p_{\mathbb{X}_{t_k}}]$ where $p_{\mathbb{X}_{t_k}}$ is the position of state $\mathbb{X}_{t_k}$ in the probability vector.

The probability $P(t)$ that a packet is successfully received during slot t can be derived as follows:

$$P(t) = \sum_{i=0}^{N-1} P\{\Omega_{s_i}^t \rightarrow \Omega_{s_{i+1}}^{t+1}\} \tag{11.9}$$

where $P\{\Omega_{s_i}^t \rightarrow \Omega_{s_{i+1}}^{t+1}\}$ indicates the probability that the number of successful transmissions occurred in the system passes from i to $i+1$ during timeslot t. It is calculated as $P\{\Omega_{s_i}^t \rightarrow \Omega_{s_{i+1}}^{t+1}\}$

$$= \begin{cases} P\{\Omega_{s_{i+1}}^{t+1}\} - \left[P\{\Omega_{s_{i+1}}^{t}\} - P\{\Omega_{s_{i+1}}^{t} \rightarrow \Omega_{s_{i+2}}^{t+1}\}\right] \\ \qquad\qquad\qquad if \ i < N-1 \\ P\{\Omega_{s_N}^{t+1}\} - P\{\Omega_{s_N}^{t}\} \\ \qquad\qquad\qquad if \ i = N-1 \end{cases} \tag{11.10}$$

Equation 11.9 can be explained as follows. The probability to receive a packet during slot t is equal to the probability that the number of successes occurred in the system increases by 1 from slot t to $t+1$. Hence, $P(t)$ is derived by summing the probabilities that, from t to $t+1$, the number of successes occurred in the system passes from i to $i+1$ ($0 \leq i \leq N-1$).

Since a packet can be successfully received during any slot from t_0 to $t_{L_{max}}$, the average packet latency $\overline{L}$ is:

$$\overline{L} = \frac{\sum_{t=0}^{L_{max}} t \cdot P(t)}{\sum_{t=0}^{L_{max}} P(t)} \tag{11.11}$$

Specifically, $\overline{L}$ is calculated by performing a weighted sum of timeslots $t,\ t \in [0,\ L_{max}]$, using $P(t)$ as weights.

Now, we derive the average energy $\overline{E}$ spent by all the N transmitting nodes during the execution of the CSMA/CA algorithm. We recall that the execution terminates when the system reaches one of the absorbing states $\mathbb{X}_f^i,\ 0 \leq i \leq N$. Thus, $\overline{E}$ can be calculated as follows:

$$\overline{E} = \mu_{\mathbb{X}_{t_0}} \tag{11.12}$$

where $\mu_{\mathbb{X}_{t_0}}$ indicates the average energy consumed by all the N nodes to reach any of the states $\mathbb{X}_f^i,\ 0 \leq i \leq N$, starting from state $\mathbb{X}_{t_0}$. According to [116] the average energy μ_X consumed by the network to reach any of the states $\mathbb{X}_f^i,\ 0 \leq i \leq N$, starting from any state $X \in F_s$, can be obtained by solving the following linear equation system, with μ_X as unknowns, and $\mu_{\mathbb{X}_f^i} = 0,\ 0 \leq i \leq N$.

$$\mu_X = \sum_{Y \in F_s} P_{XY} \cdot (E_{XY} + \mu_Y),\ \forall X \in F_s \tag{11.13}$$

11.4 Results

In this section we evaluate the performance achieved by TSCH nodes when using shared links using the analytical model derived in the previous section. To validate our analytical results we rely on simulation experiments and measurements in a real network. To this end, we implemented the TSCH MAC protocol in the Contiki OS [126]. For simulations, we used the Contiki simulation tool Cooja [127] that is able to simulate a network of emulated nodes (Tmote-sky motes [58] in our case). In Cooja, the radio medium is simulated while emulated nodes run our implementation of TSCH MAC for Contiki. Experimental measurements have been carried out in a testbed consisting of Tmote-sky motes equipped with CC2420 radio [55]. Both emulated and real nodes run exactly the same code. Finally, we point out that the energy consumption of nodes,

during both simulations and experiments, has been calculated according to equations 11.1, 11.5 and 11.7.

Our results refer to a scenario where a number of nodes try to transmit a single packet to the same receiver node r. Transmitting nodes are located in a circle around node r and each of them tries to transmit a single packet. As assumed in the analysis, all the nodes start their transmission at the same timeslot t_0. Unless stated otherwise, the parameter values used in the analysis are as shown in Table 11.1.

The analysis is organized in two parts. First, we validate our analytical model through simulation and evaluate the performance of TSCH CSMA/CA with different parameter values. Then, we investigate the impact of the capture effect in a real environment.

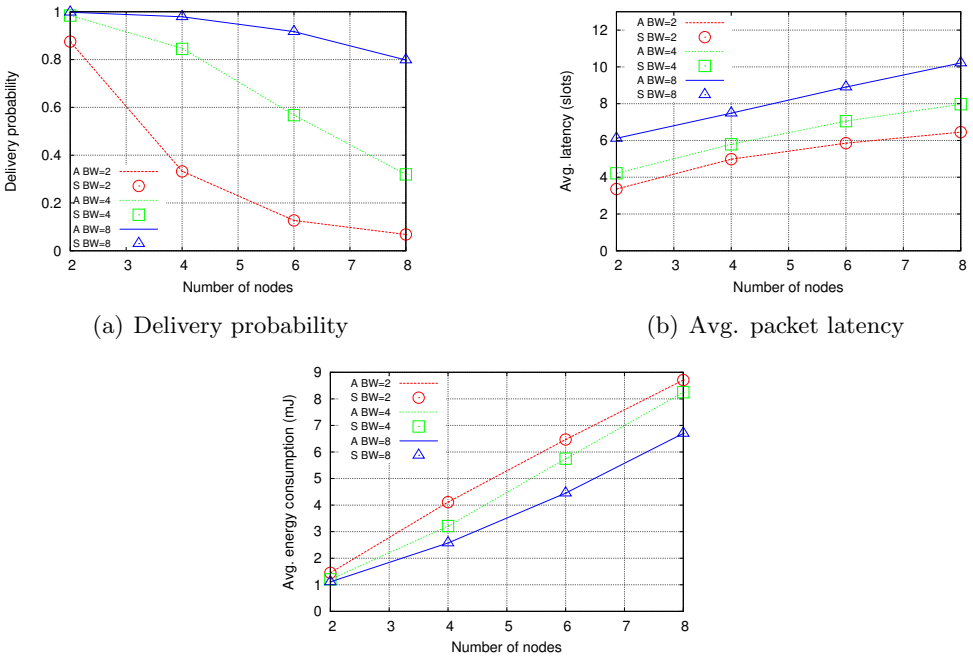
Parameter	Value
Data transmission rate (R)	250 Kbit/s
Power Consumption in TX mode (P_{TX})	52.21 mW
Power Consumption in RX mode (P_{RX})	59.11 mW
Packet transmission time (D_{tx})	4.256 ms
Ack transmission time (D_{ack})	352 μs
Timeout interval (D_{to})	864 μs

TABLE 11.1: Parameters used in our analysis.

11.4.1 Model Validation

To validate our analytical model we compare the delivery probability ($\overline{D}$), average packet latency ($\overline{L}$) and total energy consumption ($\overline{E}$) derived from eq. 11.8, 11.11 and 11.12, respectively – with the corresponding simulation results. For each simulation experiment we perform 10 independent replications (each replication consists of 100000 simultaneous transmissions of a single data packet from all the sending nodes). The simulation results presented below are averaged over all the replications. We also derive confidence intervals using the independent replications method and 95% confidence level. In these experiments we assume that capture effect cannot occur. This means that both in analysis and simulations the concurrent transmission of two or more data packets always results into a transmission failure. Also, for simplicity, we consider $macMinBE$=$macMaxBE$, i.e. the backoff window size (BW) remains unchanged after each collision. If not stated otherwise, we use $BW = 8$ and $maxR = 3$.

Figures 11.3(a), 11.3(b), 11.3(c) show the delivery probability, average packet latency and total energy consumption, respectively vs. the number of transmitting nodes N, for different backoff window sizes (BW). Similarly, Figures 11.4(a), 11.4(b), 11.4(c) show



(a) Delivery probability

(b) Avg. packet latency

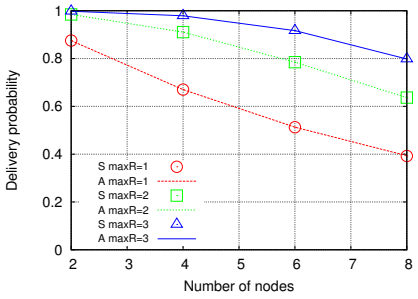
(c) Avg. energy consumption

FIGURE 11.3: Effect of the backoff window size (BW) on performance: Analytical vs. Simulation results. Increasing the backoff window size improves the reliability of the protocol (Fig. 11.3(a)) and at the same time decreases energy consumption (Fig. 11.3(c).

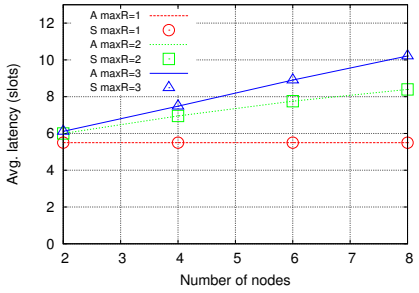
the same performance indices for different values of $maxR$ (i.e. the maximum number of retransmissions). In all these plots, analytical results (lines) and simulation outcomes (bullets) perfectly overlap. Hence, our model is validated.

As expected, the delivery probability decreases when the number of competing nodes increases, while the average packet latency and total energy consumption exhibit the opposite behavior. This is because, when the number of competing nodes grows, the collision probability increases accordingly. Hence, a high percentage of packets is dropped by the MAC layer due to exceeded number of retransmissions, which justifies the decrease in the delivery probability. In addition, collided packets are retransmitted, even several times, which increases the packet latency and total energy consumption.

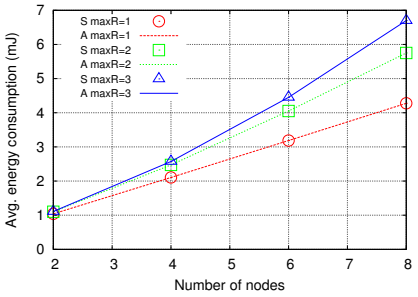
Increasing the backoff window size or the maximum allowed number of retransmissions, for a given number of contending nodes, is beneficial in terms of delivery probability (see Figures 11.3(a) and 11.4(a)). This is because, in the former case the collision probability decreases, while in the latter case nodes have more chances to transmit their packets. In both cases, the increase in the delivery probability comes at the cost of a higher latency. In terms of total energy consumption, we can observe a different trend for the two cases



(a) Delivery probability



(b) Avg. packet latency



(c) Avg. energy consumption

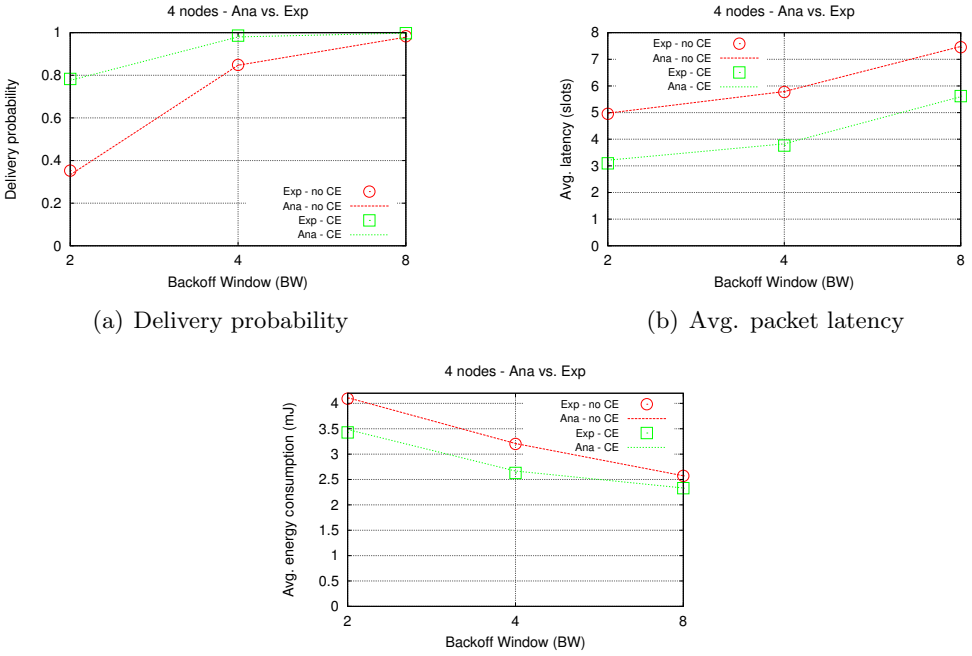
FIGURE 11.4: Effect of the maximum number of retransmissions ($maxR$) on performance: Analytical vs. Simulation results. Increasing the maximum allowed number of retransmissions improves the reliability of the protocol (Fig. 11.4(a)) but increases also energy consumption (Fig. 11.4(c)).

(see Figures 11.3(c) and 11.4(c)). Increasing the maximum number of retransmissions clearly causes a higher energy consumption at nodes. Instead, using a larger backoff window size decreases the collision probability and, hence, nodes consume less energy.

Based on the previous results we can draw the following conclusions. The performance of TSCH CSMA/CA algorithm is very good when at most two nodes are competing for channel access, while it sharply decreases with more nodes. This suggests that shared slots have to be used carefully, especially in dense network scenarios. In addition, the results clearly show that, to increase the delivery probability, it is more effective, in terms of energy savings, using a larger backoff window rather than increasing the number of retransmissions.

11.4.2 Experimental evaluation

In this section we evaluate the TSCH CSMA/CA algorithm in a real environment. Our objective is twofold. We intend to verify the ability of our analytical model to predict the performance of TSCH CSMA/CA in a real environment and investigate the



(a) Delivery probability

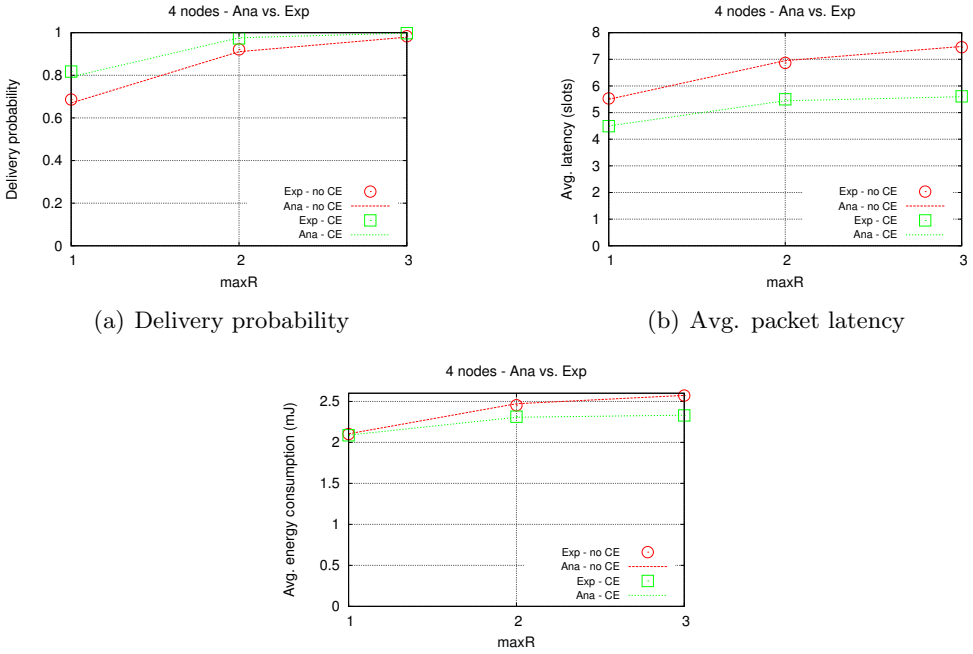
(b) Avg. packet latency

(c) Avg. energy consumption

FIGURE 11.5: Effect of the backoff window size (BW) and capture effect (CE) on performance: Analytical vs. Experimental results. Capture effect improves the reliability of the protocol and, at the same time, decreases both the avg. packet latency and energy consumption.

impact of the capture effect on the protocol performance. Our testbed is composed of 4 transmitting motes reporting data to a common receiver mote r. Experiments are carried out in a laboratory and motes are located over a table, 100cm above the ground. All the sending motes are placed at the same distance of 50cm from the receiver. Each packet transmission is repeated 1000 times.

We perform two sets of experiments. In the first set we tune our setup to exclude capture effect, while in the second one we explicitly consider it. Throughout, the two sets of experiments will be referred to as no-CE and CE experiments, respectively. In no-CE experiments the transmission power of the sending motes is appropriately adjusted, so as to achieve approximately the same received signal strength at node r, for all the transmitting nodes. Hence, the capture effect can not occur. Conversely, in the CE experiments all the motes use the same transmission power. Despite the distance from r is the same for all the transmitting nodes, we observe different received signal strenghts. Hence, the capture effect might occur. Specifically, we measured the probability $P_{CE}(n)$ of having capture effect when n motes transmit simultaneously (i.e. a packet is received correctly despite of collision). We found that $P_{CE}(n)$ is very high for $n = 2$ (0.84),



(a) Delivery probability

(b) Avg. packet latency

(c) Avg. energy consumption

FIGURE 11.6: Effect of the maximum number of retransmissions ($maxR$) and capture effect (CE) on performance: Analytical vs. Experimental results. Capture effect improves the reliability of the protocol and, at the same time, decreases both the avg. packet latency and energy consumption.

decreases sharply when $n = 3$ (0.20) and becomes negligible for $n = 4$ and beyond. We used the measured value of $P_{CE}(n)$ as an input for our analytical model. This allows us to compare analytical and experimental results in the same conditions.

Figures 11.5(a), 11.5(b), 11.5(c) compare analytical and experimental results, in terms of delivery probability, packet latency and energy consumption, respectively, for different backoff window sizes. Both the cases with and without capture effect are considered. Similarly, Figures 11.6(a), 11.6(b), 11.6(c) show the same comparison for different values of the maximum number of retransmissions. As a general remark, we point out that experimental and analytical results almost overlap, which shows that our analytical model is able to predict the performance of TSCH CSMA/CA also in a real environment.

The results clearly show that, in a real WSN, the capture effect can significantly improve the performance of the TSCH CSMA/CA algorithm. In all the considered scenarios it produces an increase in the delivery probability (up to 100%) and, at the same time, a reduction of both the average packet latency and total energy consumption. This is because, when the capture effect occurs, a collision is turned into a successful packet transmission.

11.5 Conclusion

In this chapter we have presented an analytical model of the TSCH CSMA/CA algorithm based on Discrete Time Markov Chains, that we validated using both simulations and experiments performed in a real testbed. We believe that the analysis allows to understand the behavior of the algorithm in great detail. Also, the model represents a powerful tool for industrial practitioners for both performing a fine tuning of TSCH networks and to validate implementations of TSCH MAC on real hardware. The model has been used to derive the performance achieved by nodes using shared links in terms of delivery probability, average packet latency and total energy consumption. Our results show that the performance of the algorithm strongly depends on the CSMA/CA parameter values. Also, the performance of the algorithm is significantly improved by the capture effect.

Chapter 12

Conclusions and future directions

In this chapter we summarize the main contributions of this thesis and point out some directions of future work.

12.1 Conclusions

In this thesis we studied the suitability of *Wireless Sensor Networks* (WSNs) for supporting *critical* applications, i.e. applications where, in addition to *energy efficiency*, other requirements such as *reliability*, *scalability*, *timeliness* and *security* need to be considered.

The thesis has been organized into three main parts. In the first part we analyzed WSNs leveraging the IEEE 802.15.4 standard, that is considered the reference technology for commercial WSNs. First, we provided an analytical model of the unslotted IEEE 802.15.4 CSMA/CA algorithm that, unlike previous proposed analytical models, is both accurate and tractable. Through the model it is possible to compute a number of different performance metrics such as delivery ratio, packet latency and energy consumption of sensor nodes and investigate the impact of different parameters on the performance of the CSMA/CA algorithm. However, the most important property of the proposed model is that it is able to analyze WSNs with a large number of nodes. This is because the model is not matrix-based. Instead it undertakes an approach called *Event Chains Computation* (ECC), that makes the analysis very accurate yet computationally tractable. In chapter 3 we have shown that ECC is scalable and, unlike previous techniques, is

particularly suitable for parallelization, due to its intrinsic concurrent structure. This contributes to drastic reduction of computation time. The results presented in Chapter 3 highlight that 802.15.4 WNSs suffer from severe limitations in terms of reliability and scalability that make them unsuitable for critical applications. A number of different strategies, leveraging the tuning of 802.15.4 CSMA/CA parameters values, have been proposed in the literature to increase the reliability of 802.15.4 WSNs. Our second contribution is a detailed comparison of their performance (chapter 4). The analysis we performed reports the pros and the cons of the various approaches and, hence, can help a network designer in choosing the most appopriate solution. Also, the analysis highlighted a number of limitations of current approaches such as that they are memory-less. This motivated us in designing a new algorithm, called JIT-LEAP (that we presented in chapter 5), able to overcome these limitations. JIT-LEAP is the first adaptive algorithm for the tuning of 802.15.4 CSMA/CA algorithm parameters that leverages a learning-based approach and uses a theoretically-grounded Change Detection Test (CDT). Our results demonstrate that it outperforms all the previous solutions presented in the literature for the tuning of 802.15.4 CSMA/CA parameters and that, through the use of JIT-LEAP, it is possible to significantly increase the reliability of 802.15.4 networks, making them suitable for applications having reliability as their main concern. However, JIT-LEAP is not a viable solution for time-critical applications since the increase in reliability comes at the cost of increased packet latency.

Since TDMA is typically used when applications have stringent requirements in terms of timeliness and reliability, the second part of the thesis focuses on TDMA-based WSNs. We first highlighted that, despite TDMA provides guaranteed bandwidth, high energy efficiency, absence of collisions (i.e. high reliability), low and predictable latency, it also has a number of limitations (e.g. it requires a strict synchronization between nodes, an allocation of communications slots to nodes and suffers from selective jamming attack). Then, we proposed two original contributions to overcome some limitations of TDMA-based WSNs. First (chapter 7), we presented LOCALL, a localized algorithm for allocation of transmission slots to sensor nodes that, thanks to its localized approach, does not require the exchange of messages to establish a communication schedule. This minimizes energy consumption of sensor nodes and makes LOCALL particularly suitable both for environments where packets can be corrupted or missed and dynamic networks. Then, we proposed JAMMY (chapter 8), a novel distributed and self-adaptive solution

against selective jamming attacks in TDMA-based WSNs. Unlike previous approaches, JAMMY is completely distributed, i.e. it does not suffer from typical limitations of centralized solutions (e.g. single point of failure). Moreover, JAMMY introduces a negligible computation overhead and no communication overhead at all.

Real-world WSNs typically share their radio medium with other wireless technologies such as WiFi and Bluetooth and, hence, suffer from external interference. Moreover, the performance of WSNs is usually affected by multi-path fading since any wall, person, object in their surroundings acts as a reflector for RF signals. Using multiple channels for communication, together with a channel hopping scheme, has been shown to be an effective way to mitigate both external interference and multi-path fading. For these reasons, IEEE has recently proposed the IEEE 802.15.4e *Time Slotted Channel Hopping* (TSCH) a new MAC protocol that combines time slotted access, with multi-channel and channel hopping capabilities, thus providing increased network capacity, high reliability and predictable latency, while maintaining very low duty cycles (i.e., energy efficiency). These unique characteristics make TSCH one of the most promising technologies for future real-world critical WSN applications. Therefore, in the last part of the thesis we focused on analyzing IEEE 802.15.4e TSCH. Differently from the majority of studies on TSCH networks we analyzed the mechanisms offered by TSCH to bootstrap/build the network. Specifically, in chapter 10 we studied the network formation process of IEEE 802.15.4e TSCH networks. We defined a simple *random-based network advertisement* algorithm and analyzed its performance, through both analysis and simulations, in terms of *joining time*, i.e. the total time taken by a new device to join the TSCH network. This is the first work analyzing the formation process in TSCH networks. In chapter 11, we focused on TSCH shared links, i.e special communication slots that are assigned to more than one transmitter. Shared links are expected to play a key role in future TSCH networks since they will be used - in combination with, or as an alternative to, dedicated links (i.e. slots assigned to one single transmitter) - during the network formation process (e.g. to exchange routing/scheduling information) and also in case of network failure (e.g. when a free-of-collision communication schedule is not available). We analyzed the CSMA-CA algorithm used by TSCH nodes to concurrently access shared slots. Specifically, we developed an analytical model of TSCH CSMA-CA, based on Discrete Time Markov Chains (DTMC) and used it to predict the performance experienced by nodes when accessing shared links. Our model also considers capture effect. We validated

the model through both simulation experiments and measurements in a real testbed. The obtained results clearly show the limitations of the TSCH CSMA/CA algorithm and the impact of different parameters on its performance. Also, it is shown that capture effect significantly improves the performance of the algorithm.

In the next section we point out some directions of future work.

12.2 Future directions

The work presented in this thesis can be extended following two main research directions, that we describe in the remainder of this section.

In this thesis we mainly considered the impact of the MAC protocol on the WSN performance. However, in multi-hop scenarios, the performance of a WSN can be significantly affected by the routing protocol in use also. Hence, one possible direction to extend this work of thesis could be to consider the joint impact of the routing and MAC layers to assess the suitability of a WSN for supporting critical applications. A first step could be to evaluate the performance of a multi-hop WSN adopting JIT-LEAP at the MAC layer and perform a comparison with a WSN not using an adaptive MAC. We expect that a multi-hop WSN will benefit from JIT-LEAP in two different ways. First, JIT-LEAP improves the reliability of communication links, and, hence, the reliability of the network overall. Second, having more reliable and stable links can help the routing protocol in building better network topologies. A second aspect that merits investigation is the study of mechanisms to allow an efficient interaction between the routing and MAC layers. WSNs adopting a time-slotted MAC protocol (e.g. TSCH) require a schedule of links to be established in order to operate. When the routing protocol triggers a change in the network topology, a new link schedule has usually to be established. Since in multi-hop WSNs the routing protocol can demand several changes in the network topology during time, an efficient mechanism to reduce the overhead to establish a schedule is needed. One possibility to minimize the scheduling overhead could be that of adopting a localized solution like LOCALL. Hence, a study evaluating the integration between the routing layer and a solution similar to LOCALL would be of sure interest.

Another way to extend this work of thesis could be to investigate in more detail the performance of WSNs composed of sensor nodes with mobility capacity, i.e. nodes

that can frequently leave and join the network during time. In such scenario, efficient algorithms to allow sensor nodes to quickly discover, join and leave the network are needed. One first step in this direction could be to extend the analysis we performed in chapter 10 of this thesis considering nodes with mobility capacity. It could be worth to investigate how mobility affects the joining time and the probability of a successful join. Also, an investigation of what is the optimal network advertising rate, for different mobility patterns of sensor nodes, could be of interest. Finally, another aspect that merits attention is how to efficiently manage the traffic generated by mobile nodes since the amount of traffic generated by a mobile node is difficult to be predicted in advance and is of bursty nature.

Appendix A

Appendix of Chapter 3

A.1 Derivation of $N_{P_{ij}}$

In this section we derive sets $N_{P_{ij}},\ 1 \leq i \leq B_{max},\ 1 \leq j \leq T_{max}$, where $N_{P_{ij}}$ is defined as the set of all time instants $t:\ t < f_m$ at which it is not possible, for any of the N_r sensor nodes, to have performed a CCA while in state B_{ij}, if the events in chain c occurred. To this end, we distinguish the three following types of time instants.

- **Instants when a successful transmission occurred.** If a successful transmission occurred whose CCA started at t, then $\exists e_v \in s_c:\ t_v = t \wedge T_v = S$. Since the N_r nodes have not experienced any success, it is not possible that they have performed a CCA at time t in any state B_{ij}. Therefore,

  if $\exists e_v \in s_c:\ t_v = t \wedge T_v = S \implies t \in N_{P_{ij}}\ \forall i,\ \forall j$

- **Instants at which the channel was idle.** We observe that if **(i)** no events have started at time t and **(ii)** the channel was idle at time t, then no nodes could have performed a CCA at time t, irrespective of their state. This is because, if the channel was idle at a certain time t and, at least one node performed a CCA at time t, then a successful/failure event would occur at time t. Hence,

  $\forall t < f_m$, **if** $(\nexists e_v \in s_c:\ t_v = t) \wedge (\nexists e_v \in s_c:\ t \in [t_v + D_{bp},\ f_v)) \implies t \in N_{P_{ij}}\ \forall i,\ \forall j.$

- **Instants when a collision occurred.** We observe that if **(i)** a collision occurred at time t and the collision is not the last event in the chain c and **(ii)** all instants $x \in [t_v + D_{rtx},\ t_v + D_{rtx} + (W_1 - 1)D_{bp}]$ are such that a successful transmission occurred at x or the channel was idle at x, then it is not possible for any of the N_r nodes to have performed a CCA at time t during a transmission attempt $j \neq T_{max}$. This is because, in this case, all the nodes that are involved in the collision occurred at time t, have either reached the maximum number of retransmissions for the data packet or they have successfully transmitted their data packet after the collision. Since the considered nodes have not experienced a success, only the first option is possible, i.e. it is only possible for such nodes to have performed a CCA at time t during the last transmission attempt. Hence,

  if $(\exists e_v \in s_c : e_v \neq e_m \wedge t_v = t \wedge T_v = F) \wedge (\forall x \in [t_v + D_{rtx},\ t_v + D_{rtx} + (W_1 - 1)D_{bp}],$ the channel was idle at x or a success occurred at $x)$ $\implies t \in N_{P_{ij}}, \forall i,\ 1 \leq j < T_{max}$.

A.2 Derivation of $\mathbb{P}\{\overline{N} \mid c\}$ when $T_m = F$

To derive the probability $\mathbb{P}\{\overline{N} \mid c\}$ when $T_m = F$ we need to consider the two different cases, depending on whether a node may (may not) have performed a CCA at time $t = t_m$ during the last transmission attempt (i.e. in state $B_{iT_{max}},\ 1 \leq i \leq B_{max}$). The first case occurs when $\sum_{i=1}^{B_{max}} \mathbb{P}\{CCA_{iT_{max}}^{t_m} \mid c\} > 0$, while the second one occurs when the same sum is equal to zero.

In the former case, the probability $\mathbb{P}\{\overline{N} \mid c\}$ can be calculated as follows:

$$\mathbb{P}\{\overline{N} \mid c\} = \binom{N_r}{\overline{N}} \cdot \mathbb{P}\{A_p \mid c\}^{N_p} \cdot \mathbb{P}\{A_{np} \mid c\}^{N_{np}} \cdot \mathbb{P}\{D\}^{N_d} \tag{A.1}$$

where $\binom{N_r}{\overline{N}}$ is the multinomial coefficient. The second, third and fourth term in equation A.1 give i) the probability that N_p nodes are still active and *have* participated to e_m ii) the probability that N_{np} are still active and *have not* participated to e_m iii) the probability that N_d nodes have dropped their packet and are no more active. Finally,

the multinomial coefficient $\binom{N_r}{\overline{N}}$ considers all possible combinations of nodes that could lead to the specific composition $\overline{N} = [N_p, N_{np}, N_d]$.

Let us consider now the case when $\sum_{i=1}^{B_{max}} \mathbb{P}\{CCA_{iT_{max}}^{t_m} \mid c\} = 0$. In this case, all the compositions $\overline{N} : N_p < 2$ cannot occur, i.e. $\mathbb{P}\{\overline{N} \mid c\} = 0$ if $N_p < 2$. Hence, Equation A.2 shows the computation of $\mathbb{P}\{\overline{N} \mid c\}$ for $\overline{N} : N_p \geq 2$. It is derived in the same way as eq. 3.20.

$$\mathbb{P}\{\overline{N} \mid c\} = \frac{\binom{N_r}{\overline{N}} \cdot \mathbb{P}\{A_p \mid c\}^{N_p} \cdot \mathbb{P}\{N_{np} \mid c\}^{N_{np}} \cdot \mathbb{P}\{D\}^{N_d}}{\sum_{\overline{N}:N_p \geq 2} \binom{N_r}{\overline{N}} \mathbb{P}\{A_p \mid c\}^{N_p} \mathbb{P}\{A_{np} \mid c\}^{N_{np}} \mathbb{P}\{D\}^{N_d}} \tag{A.2}$$

Please note that, since we are considering the case $\overline{N} : N_p \geq 2$, in eq. A.2 we need to normalize $\mathbb{P}\{\overline{N} \mid c\}$ considering only the probability of possible compositions, i.e. $\overline{N} : N_p \geq 2$.

A.3 Analysis of energy consumption

Given a chain $c : s_c = \{e_1,\ e_2,\ \cdots,\ e_m\}$ describing a possible outcome of the CSMA/CA algorithm the average energy consumed by all nodes in the network when the events reported by s_c happen, i.e. en_c, can be calculated as:

$$en_c = \overline{E}(e_1) + \overline{E}(e_2) + ... + \overline{E}(e_m) \tag{A.3}$$

where $\overline{E}(e_k)$ indicates the average energy consumed by nodes during event e_k. Hence, in the following, we focus on deriving $\overline{E}(e_k)$. We have to discriminate between two different cases depending on e_k is or is not the last event in chain c. In the following we describe the two different cases in detail.

Event e_k is not the last event in the chain

Let us consider the chain of events c_{k-1} derived by chain c eliminating from $s_c = \{e_1,\ e_2,\ ...,\ e_m\}$ all the events occurring in the network after e_{k-1}, i.e., $s_{k-1} = \{e_1,\ e_2, ...,\ e_{k-1}\}$. Let $\overline{E}(e_k \mid \overline{N})$ denote the average energy spent by nodes in the network when event e_k

occurs, given the composition of nodes $\overline{N}$ after e_{k-1}. Then, the average energy $\overline{E}(e_k)$ spent by all nodes in the network when event e_k occurs can be calculated as:

$$\overline{E}(e_k) = \frac{\sum_{\overline{N}} \mathbb{P}\{\overline{N} \mid c_{k-1}\} \cdot \mathbb{P}\{e_k \mid \overline{N}\} \cdot \overline{E}(e_k \mid \overline{N})}{\sum_{\overline{N}} \mathbb{P}\{\overline{N} \mid c_{k-1}\} \cdot \mathbb{P}\{e_k \mid \overline{N}\}} \tag{A.4}$$

Equation A.4 calculates the average energy spent by all nodes during event e_k by considering all the possible combinations that can lead to the occurrence of event e_k after the sequence of events $e_1,\ e_2, ..., e_{k-1}$. Specifically, the formula considers all the possible compositions $\overline{N}$ of nodes after events $e_1,\ e_2, \ldots, e_{k-1}$. Then, a weighted sum of $\overline{E}(e_k \mid \overline{N}), \forall \overline{N}$ is performed using as weights the probabilities $\mathbb{P}\{\overline{N} \mid c_{k-1}\} \cdot \mathbb{P}\{e_k \mid \overline{N}\}$, i.e. the probability of event e_k to occur given the composition of nodes $\overline{N}$. Note that $\mathbb{P}\{\overline{N} \mid c_{k-1}\} \cdot \mathbb{P}\{e_k \mid \overline{N}\}$ is normalized with $\sum_{\overline{N}} \mathbb{P}\{\overline{N} \mid c_{k-1}\} \cdot \mathbb{P}\{e_k \mid \overline{N}\}$.

To solve Equation A.4 we need to derive $\overline{E}(e_k \mid \overline{N})$. In the following, we show its computation.

Let us consider chain c_{k-1} and let us assume a specific composition $\overline{N} = (N_p,\ N_{np},\ N_d)$ of nodes after chain c_{k-1}. In general, there are different combination of events that can lead to the occurrence of event e_k given a composition $\overline{N}$ of nodes. In the following we indicate as e_{k_i} a particular combination of events that can lead to the occurrence of event e_k, given a composition $\overline{N}$ of nodes. Also, we indicate as $\overline{E}(e_{k_i} \mid \overline{N})$ the average energy consumption of the network when situation e_{k_i} occurs.

For each e_{k_i}, we indicate as $m_i \leq N_p + N_{np}$ the number of nodes that participate to event e_k when situation e_{k_i} occurs. Also, we indicate as $[N_p + N_{np} - m_i]$ the number of nodes that do not participate to event e_k. In addition, we denote as $n_{p_{e_{k-1}}}$ and as $n_{np_{e_{k-1}}}$ the number of nodes that do not participate to event e_k and have (resp. have not) participated to e_{k-1}.

Energy $\overline{E}(e_{k_i} \mid \overline{N})$ can be computed by considering two different components, namely $E_{fix}(e_{k_i} \mid \overline{N})$ and $E_{var}(e_{k_i} \mid \overline{N})$. In detail, $E_{fix}(e_{k_i} \mid \overline{N})$ is the amount of energy consumed by the m_i nodes that provoke event e_k in e_{k_i}, while $E_{var}(e_{k_i} \mid \overline{N})$ is the energy consumed by the nodes that do not participate to e_k in e_{k_i}. Let us first focus on $E_{fix}(e_{k_i} \mid \overline{N})$. We have to consider two different cases depending on e_k is a successful or failuring event. In case e_k is a successful event only one node participates to event

e_k. Hence, in this case, $m_i = 1,\ \forall e_{k_i}$ and $E_{fix}(e_{k_i} \mid \overline{N})$ can be computed as:

$$E_{fix}(e_{k_i} \mid \overline{N}) = P_{rx} \cdot D_{cca} + P_{tx} \cdot D_{tx} + P_{rx} \cdot D_{ack} \tag{A.5}$$

where P_{rx} (resp. P_{tx}) indicates the power consumption of the radio while in receive (resp. transmission) mode. Equation A.5 can be explained as follows. The first term in the formula, i.e. $P_{rx} \cdot D_{cca}$, accounts for the energy spent by the node involved in e_k to perform a CCA. Instead, the second and third term indicate the energy for transmitting the packet and receiving the ACK, respectively.

In case e_k is a failuring event, $E_{fix}(e_{k_i} \mid \overline{N})$, for each e_{k_i}, is derived as

$$E_{fix}(e_{k_i} \mid \overline{N}) = m_i \cdot \{P_{rx} \cdot D_{cca} + P_{tx} \cdot D_{tx} + P_{rx} \cdot D_{to}\} \tag{A.6}$$

Equation A.6 accounts for the energy spent by all the m_i nodes participating to e_k during situation e_{k_i} to perform a CCA operation, transmit m_i data packets, wait for an ACK until the retransmission timeout D_{to} expires.

Now we can turn our attention on deriving $E_{var}(e_{k_i} \mid \overline{N})$, which is the average energy spent by the nodes that do not participate to event e_k during situation e_{k_i}. Note that, in general, there could be two different kinds of nodes that do not participate to e_k. Specifically, we have to consider both the case of nodes that have participated to event e_{k-1} and the case of nodes that have not participated to e_{k-1}.

First, let us indicate as $f_{e_{k-1}}$ the finish time of event e_{k-1} and as t_{e_k} (resp. f_{e_k}) the starting (resp. finish) time of event e_k. The nodes we are considering do not participate to event e_k. Hence, their energy consumption, during event e_k is only due to the CCAs they perform in the interval $[t_{e_k},\ f_{e_k})$. In order to compute the average energy consumption of such nodes, we have to derive the probability for a node to have performed a number $x,\ 0 \leq x \leq B_{max}$ of consecutive CCAs during interval $[t_{e_k},\ f_{e_k})$. To this end, we have first to derive the probability for a node to have performed a CCA operation in one of the instants $t \in [f_{e_{k-1}},\ f_{e_{k-1}} + M_w \cdot D_{bp})$ given that i) it has (has not) participated to event e_{k-1} and ii) it has not taken part to e_k.

Let us first focus on nodes that have participated to event e_{k-1} and that have not participated to e_k. A node that has participated to event e_{k-1} is in one of the states $B_{1j}, 2 \leq j \leq T_{max}$, after event e_{k-1}. Hence, it performs a CCA in one of the time instants

$t \in [f_{e_{k-1}} + 2 \cdot D_{bp},\ f_{e_{k-1}} + 2 \cdot D_{bp} + (W_1 - 1) \cdot D_{bp}]$ with a probability of $1/W_1$ after event e_{k-1}. However, since we are supposing that (i) event e_k has occurred in the network after e_{k-1} and (ii) the node has not taken part to e_k, the node could only have performed a CCA operation at a time instant $t \in [t_{e_k} + D_{bp},\ f_{e_{k-1}} + 2 \cdot D_{bp} + (W_1 - 1) \cdot D_{bp}]$. Hence, the probability $\mathbb{P}\{CCA^t_{e_{k-1}} \mid n_{e_k}\}$ for the node to have performed a CCA operation in one of the time instants $t \in [t_{e_k} + D_{bp},\ f_{e_{k-1}} + 2 \cdot D_{bp} + (W_1 - 1) \cdot D_{bp}]$ can be calculated as:

$$\mathbb{P}\{CCA^t_{e_{k-1}} \mid n_{e_k}\} = \frac{1}{\frac{(f_{e_{k-1}} + 2 \cdot D_{bp} + (W_1 - 1) \cdot D_{bp} - t_{ek} - D_{bp})}{D_{bp}} + 1}$$

where the fraction at the denominator is the number of possible CCA instants for the node in the time interval $[t_{e_k} + D_{bp},\ f_{e_{k-1}} + 2 \cdot D_{bp} + (W_1 - 1) \cdot D_{bp}]$. Note that $\mathbb{P}\{CCA^t_{e_{k-1}} \mid n_{e_k}\}$ is the probability for the node to "directly" perform a CCA at a certain time t after event e_{k-1}.

Let us focus now on nodes that have not participated both to event e_{k-1} and e_k. As before, since we are assuming that (i) event e_k occurred in the network after e_{k-1} and (ii) the nodes have not participated to e_k, it is only possible for a node to have performed a CCA in the time interval $\tau = [t_{e_k} + D_{bp},\ f_{e_{k-1}} + M_w \cdot D_{bp}]$. The probability $\mathbb{P}\{CCA^{t,i,j}_{ne_{k-1}} \mid n_{e_k}\}$ for one of the considered nodes to have performed a CCA during a certain state $B_{ij}, 1 \leq i \leq B_{max}, 1 \leq j \leq T_{max}$ can be calculated as:

$$\mathbb{P}\{CCA^{t,i,j}_{ne_{k-1}} \mid n_{e_k}\} = \frac{\mathbb{P}\{CCA^t_{ij} \mid c_{k-1}\}}{\sum_{t^* \in \tau} \mathbb{P}\{CCA^{t^*} \mid c_{k-1}\}}$$

where $\mathbb{P}\{CCA^t_{ij} \mid c_{k-1}\}$ is computed using Equations (3.14) and (3.22). Note that $\mathbb{P}\{CCA^t_{ij} \mid c_{k-1}\}$ is normalized with the probability for a node to perform a CCA, during any state B_{ij}, at any time instant $t^* \in [t_{e_k} + D_{bp},\ f_{e_{k-1}} + M_w \cdot D_{bp}]$ since we are considering the case in which e_k occurred in the network and nodes not taking part to e_k.

Now we can focus on deriving the average number of CCA operations performed by a node during the occurrence of event e_k. In the following, we indicate as $P^x_{e_{k-1}}$ (resp. $P^x_{ne_{k-1}}$), $0 \leq x \leq B_{max}$ the probability for a node that has (resp. has not) participated to e_{k-1} to have performed a number x of consecutive CCAs during the occurence of event e_k. Let us first calculate $P^0_{e_{k-1}}$ and $P^0_{ne_{k-1}}$, i.e. the probability for

the nodes to have performed 0 CCA operations during event e_k. Equations A.7 and A.8 hold.

$$P^0_{ne_{k-1}} = \sum_{t \geq f_{e_k}} \sum_{i=1}^{B_{max}} \sum_{j=1}^{T_{max}} \mathbb{P}\{CCA^{t,i,j}_{ne_{k-1}} \mid n_{e_k}\} \tag{A.7}$$

$$P^0_{e_{k-1}} = \sum_{t \geq f_{e_k}} \mathbb{P}\{CCA^{t}_{e_{k-1}} \mid n_{e_k}\} \tag{A.8}$$

They can be explained as follows. The probability for a node to have performed 0 CCAs during the interval $[t_{e_k},\ f_{e_k})$ is equal to the probability for the node to have performed a CCA after time instant f_{e_k}. Hence, formulas A.7 and A.8 simply calculate the probability for a node to perform a CCA after time f_{e_k} during any state B_{ij}. Let us focus now on deriving the probabilities $P^1_{ne_{k-1}}$ and $P^1_{e_{k-1}}$, i.e. the probability for a node to perform exactly one CCA operation during the occurrence of e_k. The following equations show the computation of both $P^1_{ne_{k-1}}$ and $P^1_{e_{k-1}}$.

$$\begin{aligned} P^1_{ne_{k-1}} = & \sum_{\substack{t \geq (t_{e_k} + D_{bp}) \\ t < f_{e_k}}} \sum_{j=1}^{T_{max}} \mathbb{P}\{CCA^{t,B_{max},j}_{ne_{k-1}} \mid n_{e_k}\} + \\ & \sum_{\substack{t \geq (t_{e_k} + D_{bp}) \\ t < f_{e_k}}} \sum_{j=1}^{T_{max}} \sum_{i=1}^{B_{max}-1} \mathbb{P}\{CCA^{t,i,j}_{ne_{k-1}} \mid n_{e_k}\} \cdot \\ & \frac{W_{i+1} - \frac{f_{e_k} - D_{bp} - t}{D_{bp}}}{W_{i+1}} \end{aligned} \tag{A.9}$$

$$\begin{aligned} P^1_{e_{k-1}} = & \sum_{\substack{t \geq (t_{e_k} + D_{bp}) \\ t < f_{e_k}}} \mathbb{P}\{CCA^{t}_{e_{k-1}} \mid n_{e_k}\} \\ & \cdot \frac{W_2 - \frac{f_{e_k} - D_{bp} - t}{D_{bp}}}{W_2} \end{aligned} \tag{A.10}$$

Let us first focus on Equation A.9, which calculates the probability for a node that has not participated to e_{k-1} to perform exactly one CCA during the time interval $[t_{e_k},\ f_{e_k})$. There are two different cases that could lead a node to perform one single CCA during the occurrence of e_k. The first one is when the node performs a CCA at any time instant $t \in [t_{e_k},\ f_{e_k})$ during one of the states $B_{B_{max}j},\ 1 \leq j \leq T_{max}$, which means during the last backoff stage of any transmission attempt. In fact, in this case, the node performs its CCA operation and then terminates its channel access procedure since it reaches the maximum allowed number of consecutive CCAs. The first term in Equation A.9 computes the probability of this case to occur. The second case we have to consider is when a node performs a CCA operation at any time instant $t \in [t_{e_k},\ f_{e_k})$ during any state $B_{ij}, 1 \leq i < B_{max}$ (i.e. at a backoff stage different from the last one), finds the channel busy at time t and, then, extracts a value for the backoff time so that it performs the subsequent CCA operation after time f_{e_k}. The probability of this second case to occur can be calculated as reported by the second term in Equation A.9.

As far as Equation A.10 is concerned, it calculates the probability for a node that has participated to e_{k-1} to have performed exactly one CCA during the occurrence of e_k. The only case that leads the nodes to perform one single CCA is when they perform a CCA at any time $t \in [t_{e_k},\ f_{e_k})$, find the channel busy and, then, perform a CCA after f_{e_k}. The formula can be explained following the same arguments used for Equation A.9.

Now, we compute the probability $P^x_{ne_{k-1}}$ and $P^x_{e_{k-1}}, 2 \leq x \leq B_{max}$ for a node to perform a number $x \geq 2$ of consecutive CCAs during the occurrence of e_k. As a first step, let us indicate as t_0 time $t_{e_k} + D_{bp}$, as t_1 time $t_{e_k} + 2 \cdot D_{bp}$, ..., as t_F time $f_{e_k} - D_{bp}$. Also, let us denote by $T = \{t_0,\ t_1,\ ...,\ t_F\}$ the set of all the abovementioned time instants. In order for a node to be able to perform a number x of consecutive CCA operations, it has to be in one of the states $B_{ij}, 1 \leq j \leq T_{max},\ i : (B_{max} - i) + 1 \geq x$, i.e. the node has to be at a backoff stage i so that it can be able to perform x CCAs before dropping the packet for maximum number of consecutive CCAs. In the following, we indicate as $\mathbb{P}\{CCA^{t,i}_{ne_{k-1}} \mid n_{e_k}\} = \sum_{j=1}^{T_{max}} \mathbb{P}\{CCA^{t,i,j}_{ne_{k-1}} \mid n_{e_k}\}$ the probability for a node that has not participated to e_{k-1} to perform a CCA operation at a backoff stage i of any transmission attempt $j, 1 \leq j \leq T_{max}$. Note that a node that has participated to event e_{k-1} will always be able to perform a number $x \leq B_{max}$ of consecutive CCAs during e_k, since it is in one of the states B_{1j} after event e_{k-1}. Now, we can derive the probability for a node to perform exactly x CCAs during e_k. We have to point out that the node

could perform its x consecutive CCAs anwhere in the period $[t_{e_k},\ f_{e_k})$. Let us indicate as C the set of all the possible combinations $\{c_1,\ c_2, ...,\ c_x\}$ of x time instants $c_i \in T$. Then, we can compute $P^x_{ne_{k-1}}$ as reported below (we only show the formula to compute $P^x_{ne_{k-1}}$ since $P^x_{e_{k-1}}$ can be computed in a very similar way).

$$\begin{aligned} P^x_{ne_{k-1}} = & \sum_{i:(B_{max}-i)+1>x} \ \sum_{\{c_1,\ c_2,\ ...,c_x\}\in C_p} \mathbb{P}\{CCA^{c_1,i}_{ne_{k-1}} \mid n_{e_k}\} \\ & \cdot \left(\prod_{j=1}^{x-1} \frac{1}{W_{i+j}}\right) \cdot \left(\frac{W_{i+x} - \frac{f_{e_k}-c_x}{D_{bp}}}{W_{i+x}}\right) + \sum_{i:(B_{max}-i)+1=x} \\ & \sum_{\{c_1,\ c_2,\ ...,c_x\}\in C_v} \mathbb{P}\{CCA^{c_1,i}_{ne_{k-1}} \mid n_{e_k}\} \left(\prod_{j=1}^{x-1} \frac{1}{W_{i+j}}\right) \end{aligned} \tag{A.11}$$

Equation A.11 is composed of two different macroterms. The first macroterm considers backoff stages $i : (B_{max} - i) > x$ while the second macroterm only considers backoff stage $i : (B_{max} - i) = x$. Let us first describe the first macroterm. It calculates the probability that a node could perform x consecutive CCAs in time interval $[t_{e_k},\ f_{e_k})$ and, then, performs its $(x+1) - th$ CCA operation after time f_{e_k}. As it can be observed, for each backoff stage $i : (B_{max} - i) > x$ the formula takes into consideration all combinations $\{c_1,\ c_2, ...c_x\} \in C_p$ where $C_p = \{\{c_1,\ c_2, ...c_x\} \in C : c_j \leq c_{j-1} + D_{bp} + (W_{i+j-1})D_{bp}, 2 \leq j \leq x \wedge f_{e_k} \leq c_x + D_{bp} + (W_{i+x-1})D_{bp}\}\}$ i.e. the combinations in set C so that it is possible for a sensor node to have performed a CCA at time c_1 during backoff stage i, and, then, a CCA operation at time c_2 during backoff stage $i+1$ ..., and, then, a CCA operation at time c_x during backoff stage $i+x-1$ and, then, a CCA at backoff stage $i+x$ after time f_{e_k}. Inside the inner sum there are 3 different terms. The first one indicates the probability for a node to perform a CCA at time c_1 during a backoff stage i. The second one, i.e. $\left(\prod_{j=1}^{x-1} \frac{1}{W_{i+j}}\right)$ indicates the probability for the node to perform a series of CCAs at time instants $c_2,\ c_3, ...,\ c_x$ after having performed a CCA at time c_1 during stage i. Finally, the third term is the probability for the node to perform its $(x+1) - th$ CCA operation after time f_{e_k}. Let us consider the second macroterm in the equation. It calculates the probability for a node to perform x CCA operations during time interval $[t_{e_k},\ f_{e_k})$ and, specifically, to perform the $x - th$ CCA operation during the last backoff stage (and hence to drop the packet due to the maximum number of CCAs). As it can be observed, also in this case, the formula considers all possible combinations of x time instants in set

T at which the node could perform its CCAs. Specifically, we consider the combinations in set C_v where $C_v = \{\{c_1,\ c_2, ...c_x\} \in C : c_j \leq c_{j-1} + D_{bp} + W_{i+j-1}D_{bp}, 2 \leq j \leq x\}$. The first term in the inner sum indicates the probability to perform a CCA at time c_1 during backoff stage i while the second term $\left(\prod_{j=1}^{x-1} \frac{1}{W_{i+j}}\right)$ is the probability to perform a CCA at times $c_2,\ c_3, ...,\ c_x$ after the CCA at time c_1. Note that we have no other terms in the sum since in this case the node performs a CCA at time c_x during the last backoff stage and, thus, stops the execution of its CSMA/CA algorithm.

Now we are able to calculate the average energy consumed by a node that has (has not) participated to event e_{k-1} and that has not participated to e_k. Let us denote by $E_{var}^{ne_{k-1}}$ and $E_{var}^{e_{k-1}}$ the average energy consumptions of a node during event e_k given that it has / has not participated to e_{k-1}, respectively. They can be calculated as follows.

$$E_{var}^{ne_{k-1}} = \sum_{x=0}^{B_{max}} P_{ne_{k-1}}^{x} \cdot x \cdot (P_{rx} \cdot D_{cca}) \tag{A.12}$$

$$E_{var}^{e_{k-1}} = \sum_{x=0}^{B_{max}} P_{e_{k-1}}^{x} \cdot x \cdot (P_{rx} \cdot D_{cca}) \tag{A.13}$$

In both Equations, $x \cdot (P_{rx} \cdot D_{cca})$ is the energy consumed by a node to perform x CCA operations. As it can be observed, the formula simply performs a weighted sum of $x \cdot (P_{rx} \cdot D_{cca})$ using the $P_{ne_{k-1}}^{x}$ and $P_{e_{k-1}}^{x}$ as weights. The average energy $E_{var}(e_{k_i} \mid \overline{N})$ spent by nodes not participating to e_k in situations e_{k_i} can now be easily calculated as follows:

$$E_{var}(e_{k_i} \mid \overline{N}) = n_{p_{e_{k-1}}} \cdot E_{var}^{e_{k-1}} + n_{np_{e_{k-1}}} \cdot E_{var}^{ne_{k-1}} \tag{A.14}$$

Also, $\overline{E}(e_{k_i} \mid \overline{N})$ can be computed as the sum of $E_{fix}(e_{k_i} \mid \overline{N})$ and $E_{var}(e_{k_i} \mid \overline{N})$, i.e. $\overline{E}(e_{k_i} \mid \overline{N}) = E_{fix}(e_{k_i} \mid \overline{N}) + E_{var}(e_{k_i} \mid \overline{N})$. Finally, $\overline{E}(e_k \mid \overline{N})$ can be calculated as follows:

$$\overline{E}(e_k \mid \overline{N}) = \frac{\sum_{e_{k_i}} \overline{E}(e_{k_i} \mid \overline{N}) \cdot \mathbb{P}\{e_{k_i} \mid \overline{N}\}}{\mathbb{P}\{e_k \mid \overline{N}\}} \tag{A.15}$$

Event e_k is the last event in the chain

We consider now the case in which e_k is the last event in the chain, i.e. no nodes are more active in the network after e_k. In the following, we will indicate as c_{k-1} the chain obtained by c eliminating event $e_k = e_m$, i.e. c_{k-1} is characterized by the sequence of events $s_{k-1} = \{e_1,\ e_2,\ ...,\ e_{k-1}\}$. In general, there are several situations that can lead to the occurrence of e_k, after the events in c_{k-1}, and cause e_k to be the last event in the chain. Let us consider a specific composition $\overline{N}$ of nodes after the events $e_1,\ e_2,\ ...,\ e_{k-1}$ and let us refer as $e_{k_1},\ e_{k_2},\ ...$ all the possible situations that lead to the occurence of e_k given $\overline{N}$. Also, let us indicate as $\mathbb{P}\{e_{k_i} \mid \overline{N}\}$ the probability of occurrence of situation e_{k_i} and as $\mathbb{P}\{D \mid e_{k_i} \wedge \overline{N}\}$ the probability that all nodes in the network terminate the execution of their CSMA/CA algorithm when situation e_{k_i} occurs and given $\overline{N}$. Hence, by indicating as $\overline{E}(\overline{N} \wedge e_{k_i} \wedge D)$ the average energy consumption of the network when occurs e_{k_i} given $\overline{N}$, and, all nodes in the network are no more active after e_k, we can calculate $\overline{E}(e_k)$ as:

$$\overline{E}(e_k) = \frac{\sum_{\overline{N}} \sum_{e_{k_i}} \mathbb{P}\{\overline{N} \mid c_{k-1}\} \cdot \mathbb{P}\{e_{k_i} \mid \overline{N}\} \cdot \mathbb{P}\{D \mid e_{k_i} \wedge \overline{N}\} \cdot \overline{E}(\overline{N} \wedge e_{k_i} \wedge D)}{\sum_{\overline{N}} \sum_{e_{k_i}} \mathbb{P}\{\overline{N} \mid c_{k-1}\} \cdot \mathbb{P}\{e_{k_i} \mid \overline{N}\} \cdot \mathbb{P}\{D \mid e_{k_i} \wedge \overline{N}\}} \quad \text{(A.16)}$$

Equation A.16 calculates the average energy spent by all nodes in the network during event e_k, when e_k is the last event in the chain, by considering all possible situations that can lead to the occurrence of e_k as the last event in the network. Specifically, the formula considers all the possible compositions $\overline{N}$ of nodes after the chain of events c_{k-1} and all the situations e_{k_i} that cause event e_k to occur. Then, for each couple $(\overline{N},\ e_{k_i})$ a weighted sum of the $\overline{E}(\overline{N} \wedge e_{k_i} \wedge D)$ is performed, using as weights $\mathbb{P}\{\overline{N} \mid c_{k-1}\} \cdot \mathbb{P}\{e_{k_i} \mid \overline{N}\} \cdot \mathbb{P}\{D \mid e_{k_i} \wedge \overline{N}\}$, i.e. the probability that situation e_{k_i} occurs in the network given $\overline{N}$, and that all nodes are no more active after e_k. Note that each $\mathbb{P}\{\overline{N} \mid c_{k-1}\} \cdot \mathbb{P}\{e_{k_i} \mid \overline{N}\} \cdot \mathbb{P}\{D \mid e_{k_i} \wedge \overline{N}\}$ is normalized with $\sum_{\overline{N}} \sum_{e_{k_i}} \mathbb{P}\{\overline{N} \mid c_{k-1}\} \cdot \mathbb{P}\{e_{k_i} \mid \overline{N}\} \cdot \mathbb{P}\{D \mid e_{k_i} \wedge \overline{N}\}$, i.e. the probability of event e_k to occur and to be the last event in the chain considering all possible couples $(\overline{N},\ e_{k_i})$.

In order to solve equation A.16 we have to derive formulas to compute both $\mathbb{P}\{D \mid e_{k_i} \wedge \overline{N}\}$ and $\overline{E}(\overline{N} \wedge e_{k_i} \wedge D)$. In the following we show the computation of both of them. Let us first focus on $\mathbb{P}\{D \mid e_{k_i} \wedge \overline{N}\}$. First, let us indicate, for each couple $(\overline{N},\ e_{k_i})$, as $m_{p_{e_{k-1}}}$ ($m_{np_{e_{k-1}}}$) the number of nodes that have (have not) participated to e_{k-1} and that

participate to event e_k during situation e_{k_i}. Also, let us indicate as $n_{p_{e_{k-1}}}$ ($n_{np_{e_{k-1}}}$) the number of nodes that have (have not) participated to e_{k-1} and that do not participate to e_k during situation e_{k_i}. Now, we derive the following four different probabilities:

1. $\mathbb{P}\{D^{e_k}_{e_{k-1}}\}$: the probability for a node that has participated to both e_{k-1} and e_k to be no more active after e_k.
2. $\mathbb{P}\{D^{e_k}_{ne_{k-1}}\}$: the probability for a node that has not participated to event e_{k-1} and that has participated to e_k to be no more active after e_k.
3. $\mathbb{P}\{D^{ne_k}_{e_{k-1}}\}$: the probability for a node that has participated to event e_{k-1} and not to e_k to be no more active after e_k.
4. $\mathbb{P}\{D^{ne_k}_{ne_{k-1}}\}$: the probability for a node that has not participated to both e_{k-1} and e_k to be no more active after e_k.

Let us first focus on probabilities $\mathbb{P}\{D^{e_k}_{e_{k-1}}\}$ and $\mathbb{P}\{D^{e_k}_{ne_{k-1}}\}$, i.e. the case of nodes taking part to e_k. To correctly compute both of them, we have to discriminate between the case in which e_k is a successful event and the case in which e_k is a failuring event. In case e_k is a successful event all nodes participating to e_k surely terminate the execution of their CSMA/CA algorithm. Hence both $\mathbb{P}\{D^{e_k}_{e_{k-1}}\}$ and $\mathbb{P}\{D^{e_k}_{ne_{k-1}}\}$ are equal to 1 in this case. Let us consider now the case in which e_k is a failuring event. In this case, a node participating to e_k is no more active after e_k only if it is in one of the states $B_{iT_{max}}$, $1 \leq i \leq B_{max}$ (i.e. a backoff stage of the last transmission attempt) during the carrier sensing operation at instant t_{e_k}. Hence, in this case, $\mathbb{P}\{D^{e_k}_{e_{k-1}}\}$ and $\mathbb{P}\{D^{e_k}_{ne_{k-1}}\}$ can be calculated as follows:

$$\mathbb{P}\{D^{e_k}_{ne_{k-1}}\} = \frac{\sum_{i=1}^{B_{max}} \mathbb{P}\{CCA^{t_{e_k}}_{iT_{max}} \mid c_{k-1}\}}{\sum_{i=1}^{B_{max}} \sum_{j=1}^{T_{max}} \mathbb{P}\{CCA^{t_{e_k}}_{ij} \mid c_{k-1}\}} \tag{A.17}$$

$$\mathbb{P}\{D^{e_k}_{e_{k-1}}\} = \frac{\sum_{i=1}^{B_{max}} \mathbb{P}\{CCA^{t_{e_{k-1}}}_{iT_{max}-1} \mid c_{k-1}\}}{\sum_{i=1}^{B_{max}} \sum_{j=1}^{T_{max}-1} \mathbb{P}\{CCA^{t_{e_{k-1}}}_{ij} \mid c_{k-1}\}} \tag{A.18}$$

Equation A.17 takes into consideration nodes that have not participated to e_{k-1}. It calculates $\mathbb{P}\{D^{e_k}_{ne_{k-1}}\}$ as the ratio between 1) the probability $\sum_{i=1}^{B_{max}} \mathbb{P}\{CCA^{t_{e_k}}_{iT_{max}} \mid c_{k-1}\}$ for the node to perform a CCA at time t_{e_k} during the last transmission attempt and 2)

the probability $\sum_{i=1}^{B_{max}} \sum_{j=1}^{T_{max}} \mathbb{P}\{CCA_{ij}^{t_{e_k}} \mid c_{k-1}\}$ for the node to perform a CCA at time t_{e_k} during any state B_{ij}.

Equation A.18 takes into consideration nodes that have participated to e_{k-1}. In this case, the probability for the node to have performed a CCA at time t_{e_k} during the last transmission attempt can be calculated as the probability for the node to have performed a CCA at time $t_{e_{k-1}}$ during the transmission attempt $T_{max}-1$. Hence, equation A.18 calculates $\mathbb{P}\{D_{e_{k-1}}^{e_k}\}$ as the ratio between 1) the probability $\sum_{i=1}^{B_{max}} \mathbb{P}\{CCA_{iT_{max}-1}^{t_{e_{k-1}}} \mid c_{k-1}\}$ for the node to have performed a CCA at time $t_{e_{k-1}}$ during the transmission attempt $T_{max}-1$ and 2) $\sum_{i=1}^{B_{max}} \sum_{j=1}^{T_{max}-1} \mathbb{P}\{CCA_{ij}^{t_{e_{k-1}}} \mid c_{k-1}\}$, i.e. the probability for the node to have performed a CCA at time $t_{e_{k-1}}$ during any transmission attempt $j, 1 \leq j \leq T_{max}-1$.

Let us focus now on the probabilities $\mathbb{P}\{D_{e_{k-1}}^{ne_k}\}$ and $\mathbb{P}\{D_{ne_{k-1}}^{ne_k}\}$. The probability, for a node that has not participated to e_k, to be no more active after e_k is equal to the probability for the node to perform a CCA operation during the last backoff stage of any transmission attempt, before time f_{e_k}. As in the previous section, let us indicate as $\mathbb{P}\{CCA_{e_{k-1}}^{t,i} \mid n_{e_k}\}$ and $\mathbb{P}\{CCA_{ne_{k-1}}^{t,i} \mid n_{e_k}\}$ the probability for a node that has (has not) participated to event e_{k-1} and that has not participated to e_k to have performed a CCA operation at time t during backoff stage i. Also, let us indicate as t_0 time $t_{e_k} + D_{bp}$, as t_1 time $t_{e_k} + 2 \cdot D_{bp}$, ..., as t_F time $f_{e_k} - D_{bp}$ and denote by $T = \{t_0,\ t_1,\ ...,\ t_F\}$ the set of all the abovementioned time instants. In addition, we indicate as C the set of all the possible combinations $\{c_1,\ c_2, ...,\ c_x\}$ of x time instants $c_i \in T$. Then, $\mathbb{P}\{D_{e_{k-1}}^{ne_k}\}$ and $\mathbb{P}\{D_{ne_{k-1}}^{noe_k}\}$ can be calculated as follows:

$$\mathbb{P}\{D_{e_{k-1}}^{ne_k}\} = \sum_{i=1}^{B_{max}} \sum_{\{c_1,\ c_2,...,\ c_x\}: c_j \leq c_{j-1} + D_{bp} + (W_{i+j-1}-1)D_{bp}} \mathbb{P}\{CCA_{e_{k-1}}^{c_1,i} \mid n_{e_k}\} \left(\prod_{j=1}^{B_{max}-i} \frac{1}{W_{i+j}} \right) \tag{A.19}$$

$$\mathbb{P}\{D_{ne_{k-1}}^{ne_k}\} = \sum_{i=1}^{B_{max}} \sum_{\{c_1,\ c_2,...,\ c_x\}: c_j \leq c_{j-1} + D_{bp} + (W_{i+j-1}-1)D_{bp}} \mathbb{P}\{CCA_{ne_{k-1}}^{c_1,i} \mid n_{e_k}\} \left(\prod_{j=1}^{B_{max}-i} \frac{1}{W_{i+j}} \right) \tag{A.20}$$

Since the structure of Equations A.19 and eq. A.20 is very similar to that of Equation A.11, we omit their description. At this point, we can calculate $\mathbb{P}\{D \mid e_{k_i} \wedge \overline{N}\}$, i.e. the probability that all nodes are no more active in the network after the occurence of e_k, as follows:

$$\mathbb{P}\{D \mid e_{k_i} \wedge \overline{N}\} = \mathbb{P}\{D^{e_k}_{e_{k-1}}\}^{m_{p_{e_{k-1}}}} \cdot \mathbb{P}\{D^{e_k}_{ne_{k-1}}\}^{m_{np_{e_{k-1}}}} \cdot \mathbb{P}\{D^{ne_k}_{e_{k-1}}\}^{n_{p_{e_{k-1}}}} \cdot \mathbb{P}\{D^{ne_k}_{noe_{k-1}}\}^{n_{np_{e_{k-1}}}} \tag{A.21}$$

The formula calculates the probability $\mathbb{P}\{D \mid e_{k_i} \wedge \overline{N}\}$ by performing the product between 1) the probability $\mathbb{P}\{D^{e_k}_{e_{k-1}}\}^{m_{p_{e_{k-1}}}}$ that all the $m_{p_{e_{k-1}}}$ nodes participating to both e_{k-1} and e_k are no more active after e_k, 2) the probability $\mathbb{P}\{D^{e_k}_{ne_{k-1}}\}^{m_{np_{e_{k-1}}}}$ that all the $m_{np_{e_{k-1}}}$ nodes not participating to e_{k-1} and taking part to e_k are no more active after e_k, 3) the probability $\mathbb{P}\{D^{ne_k}_{e_{k-1}}\}^{n_{p_{e_{k-1}}}}$ that all the $n_{p_{e_{k-1}}}$ nodes participating to e_{k-1} and not to e_k are no more active after e_k and 4) the probability $\mathbb{P}\{D^{ne_k}_{ne_{k-1}}\}^{n_{np_{e_{k-1}}}}$ that all the $n_{np_{e_{k-1}}}$ nodes not participating to both e_k and e_{k-1} are no more active after e_k. Now, we can focus on deriving $\overline{E}(\overline{N} \wedge e_{k_i} \wedge D)$, i.e. the average energy spent by nodes in the network when occurs e_k and all nodes are no more active after e_k. $\overline{E}(\overline{N} \wedge e_{k_i} \wedge D)$ can be seen as divided into two different components: $E_{fix}(e_{k_i} \mid \overline{N})$ and $E_{var}(e_{k_i} \mid \overline{N})$. $E_{fix}(e_{k_i} \mid \overline{N})$ is the energy consumed by nodes participating to event e_k during situation e_{k_i}. Instead, $E_{var}(e_{k_i} \mid \overline{N})$ indicates the average energy consumed by nodes that do not participate to e_k during situation e_{k_i}. Let us first focus on $E_{fix}(e_{k_i} \mid \overline{N})$. In case e_k is a successful event there is only one node taking part to e_k and its energy consumption can be calculated as $P_{rx} \cdot D_{cca} + P_{tx} \cdot D_{tx} + P_{rx} \cdot D_{ack}$. Hence $E_{fix}(e_{k_i} \mid \overline{N}) = P_{rx} \cdot D_{cca} + P_{tx} \cdot D_{tx} + P_{rx} \cdot D_{ack}$ in this case. Let us consider now the case in which e_k is a failuring event. In such a case all nodes participating to e_k spend an energy equal to $P_{rx} \cdot D_{cca} + P_{tx} \cdot D_{tx} + P_{rx} \cdot D_{to}$. Hence, in this case, $E_{fix}(e_{k_i} \mid \overline{N})$ can be calculated as:

$$E_{fix}(e_{k_i} \mid \overline{N}) = (m_{p_{e_{k-1}}} + m_{np_{e_{k-1}}}) \cdot (P_{rx} \cdot D_{cca} + P_{tx} \cdot D_{tx} + P_{rx} \cdot D_{to}) \tag{A.22}$$

Now, we can focus on deriving the average energy consumed by nodes that do not participate to event e_k when they all are no more active after e_k. The energy consumed by nodes not taking part to e_k is only due to the CCAs they perform during time

interval $[t_{e_k},\ f_{e_k})$. Let us first focus on deriving the average energy $E_{var}^{e_{k-1}}$ consumed by all nodes that have participated to e_{k-1}. Since all these nodes are in one of the states $B_{1j},\ 1 \leq j \leq T_{max}$, after e_{k-1}, they have to perform a number of CCAs equal to B_{max} in order to be no more active after e_k. Hence, the energy consumption of such node is equal to $P_{rx} \cdot D_{cca} \cdot B_{max}$, i.e. the energy spent to perform B_{max} CCA operations. Hence $E_{var}^{e_{k-1}}$ can be calculated as:

$$E_{var}^{e_{k-1}} = (P_{rx} \cdot D_{cca} \cdot B_{max}) \cdot n_{p_{e_{k-1}}} \tag{A.23}$$

Let us focus now on deriving the average energy consumed by a node that has not taken part both to e_{k-1} and e_k when it is no more active after e_k. A node that has not taken part to e_{k-1} can be, ideally, in any state B_{ij} after e_{k-1}. Specifically, the number of CCAs it performs during the interval $[t_{e_k},\ f_{e_k})$ depends on the specific backoff stage $i,\ 1 \leq i \leq B_{max}$ the node is after e_{k-1}. Hence, we need to derive the probability for the node to be in one specific backoff stage $i,\ 1 \leq i \leq B_{max}$ after event e_{k-1}. We start by doing some considerations. First, given that 1) event e_k occurred in the network after e_{k-1}, 2) the node we are considering has not taken part to e_k and 3) the node is no more active after f_{e_k}, it is only possible for the node to have performed a CCA operation at a time instant $t \in [t_{e_k} + D_{bp},\ f_{e_k})$. Second, since the node is no more active after f_{e_k} it is not possible for it to have performed a CCA operation at a time t during a backoff stage i if $\frac{(f_{e_k} - t)}{Dbp} < (B_{max} - i)$, i.e. a sensing at a time t is only possible at a backoff stage i so that backoff stage B_{max} can be reached by the node before f_{e_k}. By doing an example, it is not possible for the node to have performed a CCA at time $t = f_{e_k} - D_{bp}$ during backoff stages different from B_{max}. Let us indicate as $(t,\ i,\ j)$, all the possible combinations of time instants $t \in [t_{e_k},\ f_{e_k})$, and states $(i,\ j)$, $1 \leq i \leq B_{max}$, $1 \leq j \leq T_{max}$ not affected by the previous considerations. Hence, the probability $\mathbb{P}\{CCA_{n_{e_{k-1}}}^{t,\ i,\ j} \mid n_{e_k} \wedge D\}$ for the node to have performed a CCA at a time t during state B_{ij} given that it has not participated to both e_{k-1} and e_k and it is no more active after e_k can be calculated as:

$$\mathbb{P}\{CCA_{n_{e_{k-1}}}^{t,\ i,\ j} \mid n_{e_k} \wedge D\} = \frac{\mathbb{P}\{CCA_{ij}^{t} \mid c_{k-1}\}}{\sum_{(t,\ i,\ j)} \mathbb{P}\{CCA_{ij}^{t} \mid c_{k-1}\}} \tag{A.24}$$

where $\mathbb{P}\{CCA_{ij}^{t} \mid c_{k-1}\}$ is calculated using both equation 3.14 and 3.22. Now, we can calculate P_{cca}^{x}, i.e. the probability for the node to have performed exactly $x, 1 \leq x \leq$

B_{max} CCAs during time interval $[t_{e_k},\ f_{e_k})$ as follows:

$$P_{cca}^{x} = \sum_{t \in [t_{e_k},\ f_{e_k})} \sum_{j=1}^{T_{max}} \mathbb{P}\{CCA_{n_{e_{k-1}}}^{t,\ (B_{max}-x)+1,\ j} \mid n_{e_k} \wedge D\} \tag{A.25}$$

The formula can be explained as follows. The probability for the node to have performed exactly x CCA operations is equal to the probability for the node to have performed a CCA operation at any time instant $t \in [t_{e_k},\ f_{e_k})$ during backoff stage $i = (B_{max} - x) + 1$ (i.e. the backoff stage so that the node has to perform x CCAs to reach backoff stage B_{max}). Hence, the formula simply performs the sum of $\mathbb{P}\{CCA_{n_{e_{k-1}}}^{t,\ (B_{max}-x)+1,\ j} \mid n_{e_k} \wedge D\}$ for any $t \in [t_{e_k},\ f_{e_k})$ and transmission attempt $j, 1 \leq j \leq T_{max}$. At this point, the average energy consumed by the node can be calculated as $\sum_{x=1}^{B_{max}} x \cdot P_{cca}^{x} \cdot (P_{rx} \cdot D_{cca})$. Hence the energy $E_{var}^{ne_{k-1}}$ consumed by all the $n_{np_{e_{k-1}}}$ nodes not participating to e_{k-1} can be calculated as follows:

$$E_{var}^{ne_{k-1}} = n_{np_{e_{k-1}}} \cdot \sum_{x=1}^{B_{max}} x \cdot P_{cca}^{x} \cdot (P_{rx} \cdot D_{cca}) \tag{A.26}$$

Finally, $E_{var}(e_{k_i}\ |\overline{N})$ can be calculated as $E_{var}(e_{k_i}\ |\overline{N}) = E_{var}^{e_{k-1}} + E_{var}^{ne_{k-1}}$ while $\overline{E}(\overline{N} \wedge e_{k_i} \wedge D)$ is equal to $E_{fix}(e_{k_i}\ |\overline{N}) + E_{var}(e_{k_i}\ |\overline{N})$.

Appendix B

Appendix of Chapter 5

B.1 PROOF OF CLAIM 1

Claim 1. Assuming that *(i)* packet transmission errors are independent from each other, and *(ii)* the PER is the same for both data packets and acknowledgements, then

$$\alpha = 1 - \left(\frac{\mathrm{R_F} - \mathrm{PER}}{(1 - \mathrm{PER})\mathrm{R_F}} \right)^{\mathrm{macMAXFrameRetries}+1}$$

where $\mathrm{R_F}$ denotes the transmission failure ratio, i.e., the probability that a data packet transmission fails for any reason.

Proof. Under non-ideal channel conditions, a packet transmission failure is experienced by the sensor node if one of the following disjoint events occur: (**i**) the data packet experiences a collision; (**ii**) no collision occurs but the transmitted data packet is corrupted by the channel; (**iii**) the data packet is received correctly by the sink but the corresponding acknowledgment is corrupted by the channel. Let p_{coll}, p_{txfail}, and $p_{ACKfail}$ denote the probability of event (**i**), (**ii**) and (**iii**), respectively. Hence, the following equations immediately follow.

$$\begin{cases} R_F = \ p_{coll} + p_{txfail} + p_{ACKfail} \\ p_{txfail} = (1 - p_{coll}) \cdot \ PER \\ p_{ACKfail} = (1 - p_{coll}) \cdot (1 - PER) \cdot \ PER \end{cases}$$

Solving the previous system yields $p_{ACKfail} = \frac{(1-R_F)\cdot PER}{(1-PER)}$. If a data packet transmission is not acknowledged, the probability that the packet has been received correctly by the sink is equal to $\beta = \frac{p_{ACKfail}}{R_F} = \frac{(1-R_F)\cdot PER}{(1-PER)\cdot R_F}$. Therefore, if a packet is dropped after *macMaxFrameRetries* retransmissions, the probability that it has been correctly received by the sink is equal to

$$\alpha = 1 - (1-\beta)^{macMAXFrameRetries+1}.$$

B.2 CONTROLLED TUNING ALGORITHM

In this Appendix we detail the Controlled Tuning Algorithm introduced in Section 5.3.5. Let j be the identified parameter setting and let $R_D(i)$ and $R_M(i)$ be the delivery ratio and miss ratio statistics contained in the Data Cluster for set i, (for $i = j-1, \quad j, \quad j+1$), provided that an entry for set i is available in the Data Cluster. The Controlled Tuning algorithm checks whether the reliability constraints are satisfied (line 4). If both reliability indexes are satisfied by using setting j and statistics are available in the Data Cluster for setting *j-1*, then the algorithm performs a fine tuning between sets j and *j-1* (lines 6-10). Specifically, it computes the distance of $R_D(j)$ and $R_M(j)$ from the corresponding threshold (i.e., R_D^{min} and R_M^{max}, respectively) and considers the index closer to the threshold. Then, the new parameter set $\mathbf{par}_{next}$, to be used in the next Beacon Interval, is changed from j to *j-1* with a probability proportional to the distance of the considered index from its threshold (line 10).

On the contrary, when at least one of the two reliability indexes does not meet the application requirements, the algorithm performs a similar fine tuning between sets j and *j+1* (lines 12-17). If no entry is available for set *j+1* in the Data Cluster the algorithms returns *j+1* as the set to be used in the next Beacon Interval (line 12). Otherwise, as described above, the distance between $R_D(j)$ and $R_M(j)$ and the corresponding thresholds is calculated but, in this case, the index with the larger distance is considered and the set to be used in the next Beacon Interval is changed from j to *j+1* with a probability proportional to the latter distance (line 17).

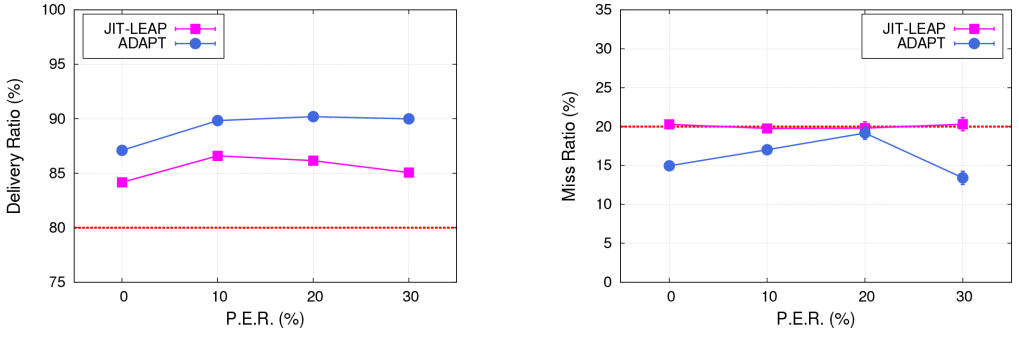
1: **for** $i = j - 1$ **to** $j + 1$
2: $R_D(i) = Cluster\ [i].\ R_D$; $R_M(i) = Cluster\ [i].\ R_M$;
3: **end for**
4: **if** $\left((R_D(j) \geq R_D^{min})\ and\ (R_M(j) \leq R_M^{max})\right)$
5: **if** $(Cluster\ [j-1] = NULL)$ **return** $j - 1$;
6: **else**
7: $p_D = \left(R_D(j) - R_D^{min}\right)/(R_D(j) - R_D(j-1))$;
8: $p_M = (R_M^{max} - R_M(j))/(R_M(j-1) - R_M(j))$;
9: $p = min(p_D,\ p_M)$;
10: **return** j $(j-1)$ with probability: $(1-p)$ p
11: **else**
12: **if** $(Cluster\ [j+1] = NULL)$ **return** $j + 1$;
13: **else**
14: $p_D = \left(R_D^{min} - R_D(j)\right)/(R_D(j+1) - R_D(j))$;
15: $p_M = (R_M(j) - R_M^{max})/(R_M(j) - R_M(j+1))$;
16: $p = max(p_D,\ p_M)$;
17: **return** j $(j+1)$ with probability: $(1-p)$ p

Algorithm 8: Controlled Tuning algorithm

B.3 ANALYSIS: UNRELIABLE WIRELESS MEDIUM

This appendix extends section 5.5 where we presented the simulation results. Specifically, below we discuss the results obtained under non-ideal channel conditions (i.e., when packet errors/losses can occur), both in stationary and dynamic scenarios.

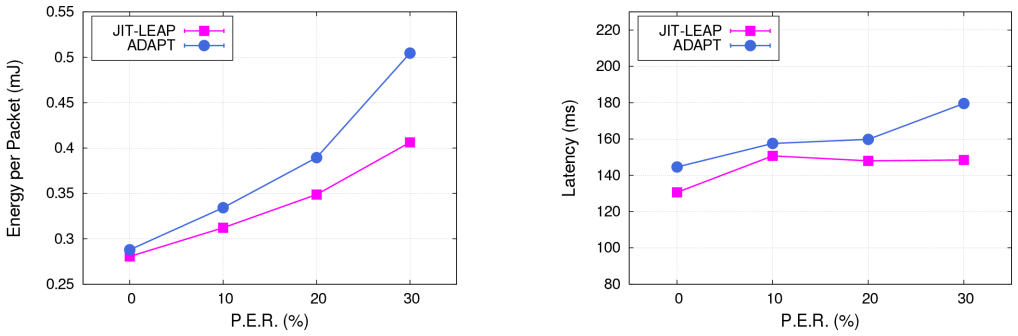
B.3.1 Analysis in stationary conditions



(a) (b)

FIGURE B.1: Packet delivery ratio (left) and miss ratio (right) vs. Packet Error Rate

In section 5.5.1 we presented the results obtained in stationary conditions, assuming an ideal channel (i.e., PER = 0). Here, we consider non-ideal channel conditions. Specifically, in the set of experiments we present, we assumed a PER in the range [0% - 30%] and considered a fixed number of nodes, i.e. 30. As anticipated in section 5.5, this



(a) (b)

FIGURE B.2: Average per-packet energy consumption (left) and average latency (right) vs. Packet Error Rate

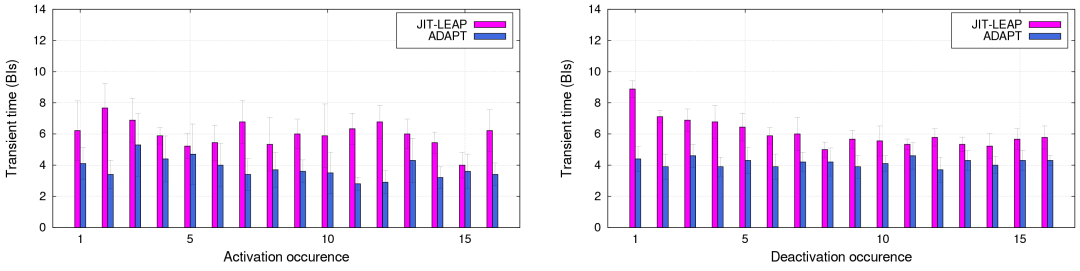
analysis focuses on ADAPT and JIT-LEAP only, as both the model-based algorithms assume ideal channel conditions. Figure B.1 shows that both JIT-LEAP and ADAPT satisfy the application requirements (in terms of both R_D and R_M). The difference between the two algorithms is mainly due to the different way ADAPT and JIT-LEAP estimate the delivery ratio experienced by a sensor node. ADAPT does not consider the effect of lost/corrupted acknowledgements and, thus, it tends to underestimate the delivery ratio experienced by the sensor node. Hence, it uses CSMA/CA parameter values higher than necessary, which results in higher energy consumption and average latency (see Figure B.2).

B.3.2 Analysis in dynamic conditions

In section 5.5.2 we presented the results obtained in dynamic conditions, i.e., with a time-varying number of nodes, assuming ideal channel conditions (i.e., PER = 0). Here, we consider a dynamic scenario where the PER changes over time, whereas the number of sensor nodes is constant and equal to 30. We assumed that PER changes periodically (every 300 Beacon Intervals) and abruptly, from 0% to 30% and vice versa. Similarly to the analysis in stationary conditions with non-ideal channel, this part of the analysis is limited to ADAPT and JIT-LEAP. As shown in Figure B.3, ADAPT exhibits shorter transient times than JIT-LEAP. This difference can be explained as follows. In ADAPT retransmissions are generally disabled ($macMaxFrameRetries = 0$) and are enabled only when a packet error/loss rate higher than the R_D^{loss}[1] is experienced. When this occurs *macMaxFrameRetries* is set to the maximum value allowed by the algorithm

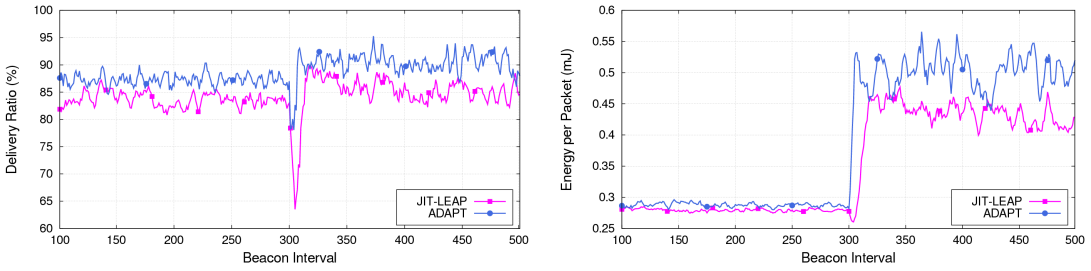
[1] In our experiments we considered $R_D^{loss} = 1 - (R_D^{low} + R_D^{high})/2$.

(i.e., 3). Obviously, such an approach makes ADAPT very reactive. However, since retransmissions are very energy consuming [30], this approach also introduces a significant energy consumption (see Figure B.4(b)). On the contrary, JIT-LEAP derives the exact value of *macMaxFrameRetries* in order to satisfy the reliability constraints required by the application. In addition, as explained above, ADAPT tends to underestimate the delivery ratio experienced by the sensor node, thus consuming more energy than necessary. For all these reasons JIT-LEAP outperforms ADAPT in terms of energy efficiency (as shown in Figure B.4(b)).



(a) (b)

FIGURE B.3: Transient Time when the Packet Error Rate increases (left) and when decreases (right)



(a) (b)

FIGURE B.4: Delivery ratio (left) and energy per packet (right) when Packet Error Rate increases

Appendix C

Appendix of Chapter 8

C.1 Alternative implementation of permute()

In Section 8.3 we considered the Knuth shuffle algorithm to perform a random permutation of slots assigned to active sensor nodes at each superframe. Here, we present an alternative optimized version of the *permute()* function, namely *permute_no_vec()*. As shown in Algorithm 9, a generic node u, currently using the i-th slot in the superframe, invokes *permute_no_vec()* providing i as input argument. The function returns the index of the slot to be used in the next superframe. It does not rely on any actual vector, thus saving a considerable amount of memory. This is extremely important when dealing with resource constrained devices such as sensor nodes.

```
1.  int permute_no_vec(unsigned old_index){
2.    unsigned i;
3.    unsigned n;
4.    unsigned new_index = old_index;
5.    for (i = N - 1; i >= 0; i--) {
6.      n = random() % (i + 1);
7.      if (i == new_index)
8.        new_index = n;
9.      else if (n == new_index)
10.        new_index = i;
11.     // else new_index = new_index;
12.   }
13.   return new_index;
14. }
```

Algorithm 9: *permute_no_vec* function.

C.2 Proof of SSP Algorithm Properties

The Secure Slot Permutation (*SSP*) algorithm, presented in Section 8.3.2, guarantees that, at every superframe, each slot is accessed by at most one sensor node, i.e. no collisions occur. In the following, we formally prove that *SSP* is collision free. More precisely, the following claim holds.

Theorem 1. Let us assume that, at the beginning of a superframe T_m, the network is in steady state condition, with $N_A \leq N$ sensor nodes accessing one slot each, i.e. the following conditions hold.

P1. Each node u accesses only one slot, say the i-th one in the superframe, i.e. $v_u[i] = 1$ and $v_u[j] = 0,\ \forall\ j \neq i$.
P2. Each slot s in the superframe, is accessed by at most one node, i.e. if $v_u[s] = 1\ \wedge\ v_w[s] = 1 \implies u \equiv w$.

Then, conditions **P1** and **P2** hold for any $T_n,\ n > m$.

Proof. Let us observe that the network can be represented by a matrix M, $M \in \mathbb{N}^{N \times N}$, such that each row of M, namely M_i, coincides with the permutation vector of the i-th sensor node. If N_A ($N_A < N$) sensor nodes are present, then $(N - N_A)$ rows in M are entirely composed of elements equal to 0. Conversely, each column in M, namely M_j, is associated to the j-th slot in the superframe. At superframe T_m, from properties *P1* and *P2*, it follows that

P3. All rows in M include at most one element equal either to 1 or 0. All other elements are equal to 0.
P4. All columns in M include at most one element equal either to 1 or 0. All other elements are equal to 0.

At the end of each superframe, every node u considers its permutation vector v_u, and locally performs N swap operations, as described in Section 8.3.1. Specifically, during the k-th operation, $k = \{1, \ldots, N\}$, two elements in v_u, namely the a-th and the b-th

one, are swapped. Also, while $a \equiv (N-k+1)$, b is a pseudo random number between 1 and $N-k+1$. Since sensor nodes share the same cryptographic key K, and update the value z over time in a synchronized way, they all consider the same pair $\{a, b\}$ at each swap. Hence, swap operations do not take into account values of swapped elements, but consider only their position in permutation vectors.

If we consider this process from a network standpoint, a single swap operation results in swapping the a-th and b-th column of M. More formally, let us consider matrix M at the beginning of superframe T_m, namely $M_0^{T_m}$. Then, we consider a permutation π of N elements, i.e. $\pi : \{1, \ldots, N\} \rightarrow \{1, \ldots, N\}$ defined as follows.

$$\pi(x) = \begin{cases} x & \text{if } x \notin \{a, b\} \\ a & \text{if } x = b \\ b & \text{if } x = a \end{cases} \tag{C.1}$$

Finally, let us consider the identity matrix $I \in \mathbb{N}^{N \times N}$, and define e_i as the i-th row of I. Also, we introduce the permutation matrix P_π, such that the i-th row of P_π coincides with $\pi(i)$-th row of I. More formally, we have $P_\pi = \left[e_{\pi(1)}; \ldots; e_{\pi(N)}\right]$. Then, swapping the a-th and b-th column of M, produces a matrix $M^{'} = MP_\pi$.

After the k-th swap round, $k = \{1, \ldots, N\}$, we have $M_k^{T_m} = M_{k-1}^{T_m} P_\pi$. That is, the resulting matrix $M_k^{T_m}$ is obtained by swapping the a-th and b-th column of $M_{k-1}^{T_m}$. As a consequence, also $M_k^{T_m}$ displays properties P1, P2, P3, and P4. Also, this is true after all N swap operations have been performed. Thus, the above mentioned properties are valid at the beginning of every superframe, which proves that the SSP algorithm is collision free. □

C.3 Discrete Time Markov Chain

For the sake of simplicity, in Section 8.6.2 we derived a Discrete Time Markov Chain (DTMC) for the simple case when the superframe consists of $N = 3$ slots, there is just one active node (i.e. $N_A = 1$), and $N_j = 2$ nodes start their join procedure simultaneously at superframe T_j. In this section, we derive the DTMC model in the general case, i.e., for any values of N, N_A, N_j. Specifically, we first define a *generate_states()* function,

used to generate all possible states of the DTMC. To this end, we refer to the event probabilities derived in Section 8.6.1. Then, we rely on *generate_states()* to derive the DTMC.

We recall that we observe the system at the beginning of every superframe T_m. Specifically, we represent the system state as a vector $X_m = [n_1,\ n_2, ...,\ n_N]$, where element n_i refers to the i-th slot, namely s_i, of T_m. Specifically, n_i indicates the number of joining nodes that directly try contention at slot s_i. Since we consider the presence of N_j joining nodes, we have $n_i \leq N_j, \forall\ i$. In the following, we refer to T_j as the superframe at which joining nodes start to execute the slot acquisition process, and N_A as the number of acquired slots at the beginning of superframe T_j. Also, we define $S_m = [\overline{s}_1,\ \overline{s}_2,\ ...,\ \overline{s}_N]$ as the slot allocation pattern at the beginning of a given superframe T_m. Specifically, $\overline{s}_i$ indicates the status of slot s_i, i.e. if it is either *free* (F) or *acquired* (A). Of course, superframe T_j features N_A slots whose status is A.

Initially, the system state is $X_j = [N_j,\ 0, ...,\ 0]$, i.e. all joining nodes have scheduled their transmission at the first slot of superframe T_j. Then, the system can evolve in different states, depending on both the slot allocation pattern S_j and the specific events that occur during the contention for each slot.

State generation

Algorithm 10 details function *generate states*(X_m, S_m). Such a function takes two parameters as input (line 1), namely X_m and S_m, that are the network state and the slot allocation pattern, at the beginning of superframe T_m, respectively. It returns a set $G_{X_m S_m}$ of *transition* vectors, each one of which includes the following three fields:

- $X_{m+1} = [n_1,\ n_2, ...,\ n_N]$ represents one of the possible states where the network can evolve to from superframe T_m to superframe T_{m+1}, starting from state X_m and slot allocation pattern S_m.
- $p_{X_m X_{m+1}}$ is the probability that the system changes its state from X_m to X_{m+1}, given the S_m slot allocation pattern at superframe T_m.
- $e_{X_m X_{m+1}}$ is the energy consumed by joining nodes when the system changes its state from X_m to X_{m+1}, given the S_m slot allocation pattern at superframe T_m.

1: $G_{X_m S_m}$ ***generate_states*** (X_m, S_m)
2: *transition init*
3: $init.X_{m+1} = X_m$
4: $init.p_{X_m X_{m+1}} = 1$
5: $init.e_{X_m X_{m+1}} = E_u \cdot (N_j - \sum_{i=1}^{N} X_m[i])$
6: $\Omega_0 = \emptyset$
7: $\Omega_0 = \Omega_0 \cup init$
8: **for** i **in** 1 ... N
9: **for each** $\omega \in \Omega_{i-1}$
10: **if** $\omega.X_{m+1}[i] = 0$ **then**
11: *transition* $gen = \omega$
12: $\Omega_i = \Omega_i \cup gen$
13: **continue**
14: **end if**
15: **if** $\overline{s}_i \in S_m = A$ **then**
16: *transition* $gen = \omega$
17:

$$gen.X_{m+1}[(i+1)\%N] = \omega.X_{m+1}[(i+1)\%N] + \omega.X_{m+1}[i]$$

18: $gen.X_{m+1}[i] = 0$
19: $gen.p_{X_m X_{m+1}} = \omega.p_{X_m X_{m+1}} \cdot 1$
20: $gen.e_{X_m X_{m+1}} = \omega.e_{X_m X_{m+1}} + \omega.X_{m+1}[i] \cdot E_{CS}$
21: $\Omega_i = \Omega_i \cup gen$
22: **end if**
23: **if** $\overline{s}_i \in S_m = F$ **then**
24: *transition* $gen = \omega$
25:

$$gen.X_{m+1}[(i+1)\%N] = \omega.X_{m+1}[(i+1)\%N] + \omega.X_{m+1}[i] - 1$$

26: $gen.X_{m+1}[i] = 0$
27: $gen.p_{X_m X_{m+1}} = \omega.p_{X_m X_{m+1}} \cdot P_s^F(\omega.X_{m+1}[i])$
28: $gen.e_{X_m X_{m+1}} = \omega.e_{X_m Y_{m+1}} + E_s(\omega.X_{m+1}[i])$
29: $\Omega_i = \Omega_i \cup gen$
30: **for** $k = 2$ to $\omega.X_{m+1}[i]$
31: *transition* $gen = \omega$
32:

$$gen.X_{m+1}[(i+1)\%N] = \omega.X_{m+1}[(i+1)\%N] + (\omega.X_{m+1}[i] - k)$$

33: $gen.X_{m+1}[i] = k$
34: $gen.p_{X_m X_{m+1}} = \omega.p_{X_m X_{m+1}} \cdot P_c^F(k \mid \omega.X_{m+1}[i])$
35: $gen.e_{X_m X_{m+1}} = \omega.e_{X_m X_{m+1}} + E_c(k \mid \omega.X_{m+1}[i])$
36: $\Omega_i = \Omega_i \cup gen$
37: **end for**
38: **end if**
39: **end for**
40: **end for**
41: **return** $G_{X_m S_m} = \Omega_N$
42: **end function**

Algorithm 10: Function *generate_states()*

In practice, the function *generate_states*() produces $G_{X_m S_m}$, i.e. the set of all possible states X_{m+1} where the system can evolve to from state X_m and slot allocation pattern S_m. For each X_{m+1}, the transition probability $p_{X_m X_{m+1}}$ and the energy consumption $e_{X_m X_{m+1}}$ are also reported.

Initialization. The function *generate_states*() relies on an iterative approach to generate all possible *transition* vectors. First of all, *generate_states*() creates and initializes transition *init* (line 2-7). Specifically, $init.X_{m+1}$ is set to X_m, while $init.p_{X_m X_{m+1}}$ is set to 1, since no events have been considered yet. Instead, $init.e_{X_m X_{m+1}}$ assumes an initial value equal to $E_u \cdot (N_j - \sum_{i=1}^{N} X_m[i])$, where E_u is the energy spent by joining nodes that have already completed their join procedure. Since there are $N_j = N_F$ joining nodes at superframe T_j , the number of joining nodes that have already completed their join procedure can be calculated as $N_j - \sum_{i=1}^{N} X_m[i]$. Then, transition *init* is added to set Ω_0. In general, Ω_i, $1 \leq i \leq N$, represents the set of *transitions* generated by *generate_states*() after having examined i slots in the current superframe. During slot s_i, $1 \leq i \leq N$, function *generate_states*() considers all transitions $\omega \in \Omega_{i-1}$ (lines 8-9). In case no joining nodes are contending slot s_i, the current transition is skipped, and the next one is considered (lines 10-14). Otherwise, it behaves differently depending on whether slot s_i is *acquired* (A) or *free* (F).

Acquired slot. First, let us consider the case when slot s_i is acquired (lines 15-22). In such a case, all nodes contending for slot s_i find the channel busy while performing their carrier sense operation and, thus, schedule a retry at the next slot s_{i+1}. Then, for each transition $\omega \in \Omega_{i-1}$, a new transition $gen = \omega$ is created (line 16), and the following operations are performed (lines 17-21).

1. All the $\omega.X_{m+1}[i]$ nodes contending for slot s_i retry at the next slot s_{i+1} (Algorithm 2), during which other $\omega.X_{m+1}[(i+1)\%N]$ nodes are going to contend, i.e.

$$gen.X_{m+1}[(i+1)\%N] = \omega.X_{m+1}[(i+1)\%N] + \omega.X_{m+1}[i] \quad \text{(C.2)}$$

2. No nodes remain on slot s_i, then

$$gen.X_{m+1}[i] = 0 \quad \text{(C.3)}$$

3. All contending nodes retry at the next slot for sure, hence the overall transition probability does not change, that is

$$gen.p_{X_m X_{m+1}} = \omega.p_{X_m X_{m+1}} \cdot 1 \tag{C.4}$$

4. All the $\omega.X_{m+1}[i]$ nodes perform a channel sensing during slot s_i. Hence, the overall energy consumed by joining nodes during slot s_i is equal to $\omega.X_{m+1}[i] \cdot E_{CS}$. More formally, we have

$$gen.e_{X_m X_{m+1}} = \omega.e_{X_m X_{m+1}} + \omega.X_{m+1}[i] \cdot E_{CS} \tag{C.5}$$

5. Finally, being a possible transition state generated at slot s_i, *gen* is added to the set Ω_i, that is

$$\Omega_i = \Omega_i \cup gen \tag{C.6}$$

Free slot. Now we consider the case when slot s_i is *free* (F) (lines 23-39). In such a case, different kinds of events can occur during the contention for slot s_i. Specifically, for each $\omega \in \Omega_{i-1}$, the algorithm considers all the events that can occur at slot s_i in the presence of $\omega.X_{m+1}[i]$ sensor nodes trying to access it. Thus, there are exactly $\omega.X_{m+1}[i]$ events to be considered, which can be classified into two different classes, namely *success* and *defeat*. The former regards the cases when only one sensor node wins the contention and successfully transmits its packet at slot s_i, while all other sensor nodes find the medium busy. The latter refers to the cases when two or more sensor nodes experience a collision at slot s_i, while the remaining ones find the medium busy.

Success. First, let us focus on the case when one of the $\omega.X_{m+1}[i]$ contending sensor nodes successfully transmits its packet (lines 24-29). Then, a new transition $gen = \omega$ is created, and the following operations are performed.

1. All nodes contending slot s_i but one, i.e. $\omega.X_{m+1}[i] - 1$, retry at the next slot s_{i+1}, during which other $\omega.X_{m+1}[(i+1)\%N]$ nodes are going to contend, i.e.

$$gen.X_{m+1}[(i+1)\%N] = \omega.X_{m+1}[(i+1)\%N] + (\omega.X_{m+1}[i] - 1) \tag{C.7}$$

2. The $\omega.X_{m+1}[i]-1$ nodes that have lost the contention at slot s_i, retry at the next slot s_{i+1}, whereas the winner node terminates its own join procedure. Thus, no joining nodes remain in slot s_i, then

$$gen.X_{m+1}[i] = 0 \tag{C.8}$$

3. We have to consider both i) the probabilities of all events occurred before slot s_i, i.e. $\omega.p_{X_m X_{m+1}}$; and ii) the probability of the specific event happened during slot s_i, i.e. $P_s^F(\omega.X_{m+1}[i])$. Hence,

$$gen.p_{X_m X_{m+1}} = \omega.p_{X_m X_{m+1}} \cdot P_s^F(\omega.X_{m+1}[i]) \tag{C.9}$$

4. In the presence of $\omega.X_{m+1}[i]$ contending nodes, the overall energy consumed by joining nodes is equal to $E_s(\omega.X_{m+1}[i])$. Hence, we have

$$gen.e_{X_m X_{m+1}} = \omega.e_{X_m X_{m+1}} + E_s(\omega.X_{m+1}[i]) \tag{C.10}$$

5. Finally, being a possible transition state generated at slot s_i, gen is added to the set Ω_i, that is

$$\Omega_i = \Omega_i \cup gen \tag{C.11}$$

Collision. Now we consider the case when two ore more sensor nodes experience a collision at slot s_i (lines 30-37). For each possible number of colliding nodes k, $2 \leq k \leq \omega.X_{m+1}[i]$, a new transition $gen = \omega$ is created, and the following operations are performed.

1. All nodes that have found the medium busy, i.e. $\omega.X_{m+1}[i]-k$, retry at the next slot s_{i+1} (Algorithm 2), during which other $\omega.X_{m+1}[(i+1)\%N]$ nodes are going to contend. Hence,

$$gen.X_{m+1}[(i+1)\%N] = \omega.X_{m+1}[(i+1)\%N] + (\omega.X_{m+1}[i] - k) \tag{C.12}$$

2. All the k colliding nodes remain in slot s_i, then

$$gen.X_{m+1}[i] = k \tag{C.13}$$

3. We have to consider both i) the probabilities of all events occurred before slot s_i, i.e. $\omega.p_{X_m X_{m+1}}$; and ii) the probability that k sensor nodes out of the $\omega.X_{m+1}[i]$ contending ones collide with one another, i.e. $P_c^F(k \mid \omega.X_{m+1}[i])$. Hence,

$$gen.p_{X_m X_{m+1}} = \omega.p_{X_m X_{m+1}} \cdot P_c^F(k \mid \omega.X_{m+1}[i]) \tag{C.14}$$

4. In the presence of $\omega.X_{m+1}[i]$ contending nodes, k of which collide with one another, the overall energy consumed by joining nodes is equal to $E_c(k \mid \omega.X_{m+1}[i])$. Thus, we have

$$gen.e_{X_m Y_{m+1}} = \omega.e_{X_m Y_{m+1}} + E_c(k \mid \omega.X_{m+1}[i]) \tag{C.15}$$

5. Finally, being a possible transition state generated at slot s_i, *gen* is added to the set Ω_i, that is

$$\Omega_i = \Omega_i \cup gen \tag{C.16}$$

Epilogue. As it can be observed, when all the N slots in the superframe have been examined, the set Ω_N contains all possible states X_{m+1} reachable from state X_m, in the presence of a slot allocation pattern S_m. Thus, *generate_states*() assigns Ω_N to $G_{X_m S_m}$, and returns it (line 41). Note that it is possible that two different *transitions* in $G_{X_m S_m}$ contains the same X_{m+1}. This is because, in general, state X_{m+1} can be reached from state X_m in different ways, depending on the particular events occurred during superframe T_m.

Derivation of the DTMC

Algorithm 11 details function *generate_DTMC(*N_j, N_A*)*. Such a function takes two parameters as input, namely N_j and N_A, i.e., the number of joining nodes and the number of already assigned slots, respectively (line 1). The function returns a pair of matrices (P, E). In particular, P is the transition probability matrix, and each element P_{XY} indicates the probability that the system state changes from X to Y. Instead, matrix E indicates the energy consumption of joining nodes. Specifically, each element E_{XY} represents the average energy consumed by joining nodes when the network state changes from X to Y.

The *generate_DTMC()* function mainly relies on the following three sets.

1: (P, E) ***generate_DTMC*** (N_j, N_A)
2: $Unprocessed = \emptyset$
3: $Examined = \emptyset$
4: $F = \emptyset$
5: $X_j = [N_j,\ 0,\ ...,\ 0]$
6: $Unprocessed = Unprocessed \cup X_j$
7: **while**($Unprocessed! = \emptyset$)
8: extract X_m from $Unprocessed$
9: $Examined = Examined \cup X_m$
10: $n_j = \sum_{i=1}^{N} X_m[i]$
11: **for each** $S_m : |i : S_m[i] = A| = N_A + N_F - n_j$
12: $G_{X_m S_m} = generate\_states(X_m, S_m)$
13: **for each** $g \in G_{X_m S_m}$
14: **if** $g.X_{m+1} \notin Unprocessed \wedge g.X_{m+1} \notin Examined$
15: $Unprocessed = Unprocessed \cup g.X_{m+1}$
16: **end if**
17: $g.p_{X_m X_{m+1}} = g.p_{X_m X_{m+1}} \cdot \frac{1}{\binom{N}{N_A + N_F - n_j}}$
18: $F.insert(\{X_m, g.X_{m+1}, g.p_{X_m X_{m+1}}, g.e_{X_m X_{m+1}}\})$
19: **end for**
20: **end for**
21: **end while**
22: $P = [\ ]$
23: $E = [\ ]$
24: **for each** $X_m \in Examined$
25: **for each** f *in* $F : f.X_m = X_m$
26: $P_{X_m X_{m+1}} += f.p_{X_m X_{m+1}}$
27: **end for**
28: **end for**
29: **for each** $X_m \in Examined$
30: **for each** f *in* $F : f.X_m = X_m$
31: $E_{X_m X_{m+1}} += f.e_{X_m X_{m+1}} \cdot \frac{f.p_{X_m X_{m+1}}}{P_{X_m X_{m+1}}}$
32: **end for**
33: **end for**
34: **return** (P, E)
35: **end**

Algorithm 11: Function *generate_DTMC()*

- *Unprocessed*, a set including all the states that are still to be examined.
- *Examined*, the set including all network states that have been already examined.
- F: a list of elements $f = \{X_m,\ X_{m+1},\ p_{X_mX_{m+1}},\ e_{X_mX_{m+1}}\}$, each one of which represents one of the possible transitions from a given state X_m to state X_{m+1}. In particular, f is composed of four elements. X_m and X_{m+1} are the initial and final state of transition f, respectively. Instead, $p_{X_mX_{m+1}}$ and $e_{X_mX_{m+1}}$ are the occurrence probability of transition f, and the energy spent by joining nodes during f, respectively.

Initially, the above mentioned sets are created and initialized (lines 2-4). Also, vector $X_j = [N_j,\ 0, ...,\ 0]$, representing the network state at superframe T_j, is created and added to set *Unprocessed* (lines 5-6). The join procedure terminates when there are no joining nodes that have still not acquired a slot, i.e. the network state is $X_f = [0,\ 0, ...,\ 0]$.

Then, the function performs a *while* loop (lines 7-21), that ends when there are no more states to be analyzed, i.e. when *Unprocessed* $= \emptyset$. At each step of the loop, one element X_m is extracted from *Unprocessed* (line 8), and added to *Examined* (line 9). Thereafter, *generate_DTMC()* generates all possible states X_{m+1} where the network may evolve to from state X_m at a given superframe T_m. The goal is to derive, for every pair $\{X_m, X_{m+1}\}$, both the probability $P_{X_mX_m+1}$ that the system state changes from X_m to X_{m+1}, and $E_{X_mX_m+1}$, i.e. the average energy consumed by joining nodes when such a transition occurs. In order to do that, all the possible slot allocation patterns $S_m = [\bar{s}_1, \bar{s}_2, ..., \bar{s}_N]$ at superframe T_m are considered (line 11). Since i) the network state at superfame T_m is X_m; ii) there were exactly N_A assigned slots when the join procedure started; and iii) we have supposed that the number of joining nodes $N_j = N_F$, then the number of assigned slots at a given superframe T_m is equal to $N_A + N_F - n_j$, where $n_j = \sum_{i=1}^{N} X_m[i]$. Specifically, such a formula calculates the total number of assigned slots at the beginning of T_m, by subtracting the number of joining nodes that are still competing to acquire a slot, i.e. $n_j = \sum_{i=1}^{N} X_m[i]$, from the total number of slots in the superframe, i.e. $N = N_A + N_F$. Having said that, a slot allocation pattern S_m is considered if and only if it satisfies the following condition: $S_m : |\{i \in [1,\ ...,\ N] : S_m[i] = A\}| = N_A + N_F - n_j$, where $|\{\cdot\}|$ indicates the cardinality of a given set. That is, considered slot allocation patterns are such that the number of

assigned slots is equal to $N_A + N_F - n_j$. So doing, the function *generate_states()* is invoked for each considered pair $\{X_m, S_m\}$ (line 12). Such a function returns $G_{X_m S_m}$, i.e. the set of all possible *transitions* that can occur given the network state X_m and the slot allocation pattern S_m. Then, for each *transition* $g \in G_{X_m S_m}$, the following operations are performed. First, if state $g.X_{m+1}$ is neither in the set *Unprocessed* nor in the set *Examined*, it is added to *Unprocessed*, since it still has to be analyzed (lines 14-16). Second, the probability $g.p_{X_m X_{m+1}}$, i.e. the probability that transition g occurs, is multiplied by $P(S_m \mid X_m)$, i.e. the probability that the slot allocation pattern is S_m at the beginning of T_m, given the network state X_m (line 17). This is because the probability $g.p_{X_m X_{m+1}}$ is computed considering a specific slot allocation pattern S_m at superframe T_m. Specifically, the probability $P(S_m \mid X_m)$ is equal to $1/\binom{N}{N_A+N_F-n_j}$ where $n_j = \sum_{i=1}^{N} X_m[i]$. In fact, the number of possible slot allocation patterns S_m at superframe T_m is equal to $\binom{N}{N_A+N_F-n_j}$ and all slot allocation patterns S_m are equiprobable. Finally, the tuple $\{ X_m,\ g.X_{m+1},\ g.p_{X_m X_{m+1}},\ g.e_{X_m X_{m+1}}\}$ is added to list F (line 18). Once the analysis has been completed, set *Examined* contains all possible system states X_m. Also, list F contains all possible transitions from a given state X_m to any other state X_{m+1}. Then, *generate_DTMC()* computes matrix P (lines 24-28). Specifically, the probability to reach state $Y \in Examined$ from state $X \in Examined$ is computed by considering all elements f in F such that $f.X_m = X \wedge f.X_{m+1} = Y$. More specifically, the probability to reach state XY from state X, i.e. P_{XY}, is calculated as:

$$P_{XY} = \sum_{f : f.X_m = X \wedge f.X_{m+1} = Y} f.p_{X_m X_{m+1}} \tag{C.17}$$

Finally, the matrix E is computed (lines 29-33). Specifically, the average energy spent by joining nodes when the network state changes from X to Y, i.e. E_{XY}, is calculated as:

$$E_{XY} = \sum_{f : f.X_m = X \wedge f.X_{m+1} = X} f.e_{X_m X_{m+1}} \cdot \frac{f.p_{X_m X_{m+1}}}{P_{XY}} \tag{C.18}$$

That is, E_{XY} is calculated as the weighted sum of energies $f.e_{X_m X_{m+1}}$ spent by joining nodes during each possible transition f. As weight for each f, we consider $\frac{f.p_{X_m X_{m+1}}}{P_{XY}}$, i.e. the probability that the specific transition f occurs, given an occured transition from X to Y.

C.4 Analysis of energy consumption

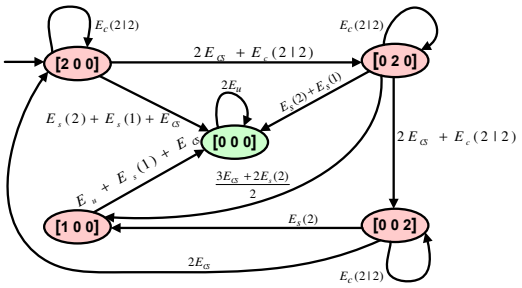


FIGURE C.1: Energy consumptions for $N = 3$, $N_A = 1$, $N_j = 2$.

Figure C.1 shows the average energy consumption associated with each transition derived according to Equation 8.5 and 8.6. For the sake of simplicity, we only explain the computation of the average energy consumption for transitions originating from the inital state of the system, i.e. $[2,\ 0,\ 0] \rightarrow [2,\ 0,\ 0]$, $[2,\ 0,\ 0] \rightarrow [0,\ 2,\ 0]$, and $[2,\ 0,\ 0] \rightarrow [0,\ 0,\ 0]$. The average energy spent during all the other transitions can be derived following the same line of reasoning. Let us focus on transition $[2,\ 0,\ 0] \rightarrow [2,\ 0,\ 0]$. As mentioned above, this transition occurs when the two joining nodes experience a collision at the first slot of the superframe. Hence, according to Equation 8.6, the average energy consumption is equal to $E_c(2 \mid 2)$ in this case. Transition $[2,\ 0,\ 0] \rightarrow [0,\ 2,\ 0]$ occurs when: i) the two joining nodes find the channel busy during the first slot and move to the second slot and ii) they experience a collision. During the first slot the two nodes perform a channel sensing operation. Hence, they spend an energy equal to $2 \cdot E_{CS}$. Instead, the energy consumed due to the collision at the second slot is equal to $E_c(2 \mid 2)$. Thus, the energy consumption for this transition is $2 \cdot E_{CS} + E_c(2 \mid 2)$. The last transition we consider is $[2,\ 0,\ 0] \rightarrow [0,\ 0,\ 0]$. This transition occurs when both joining nodes success in acquiring a free slot during the superframe. There are three different cases that cause this transition. The first one happens when i) $S_m = [A,\ F,\ F]$ and ii) two successes occur, one at the second slot and one during the third slot. The energy spent in this case is equal to $2 \cdot E_{CS} + E_s(2) + E_s(1)$ since the two nodes sense the channel during the first slot and there are two successful events. Also, the probability for this case to occur is equal to $\frac{1}{3} \cdot P_s^F(2)$. The second case is when $S_m = [F,\ A,\ F]$ and two successful transmissions occur, one at the first slot and one at the third slot. Hence, the energy spent by the joining nodes is equal to $E_s(2) + E_{CS} + E_s(1)$. The probability of this case to occur is $\frac{1}{3} \cdot P_s^F(2)$. The last case happens when $S_m = [F,\ F,\ A]$ and two

successes occur, one at the first and one at the second slot. The energy spent in this case is equal to $E_s(2) + E_s(1)$ while the probability for the case to occur is $\frac{1}{3} \cdot P_s^F(2)$. As it can be observed all the three cases have the same probability to occur. Hence, we can calculate the average energy consumption simply by performing an arithmetic mean of the energies spent in the three different cases. Thus, the average energy consumption for this transition is equal to $E_s(2) + E_s(1) + E_{CS}$.

List of Figures

Bibliography

[1] Andreas Willig. Recent and emerging topics in wireless industrial communications: A selection. *Industrial Informatics, IEEE Transactions on*, 4(2):102–124, 2008.

[2] Aleksandar Milenković, Chris Otto, and Emil Jovanov. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Computer communications*, 29(13):2521–2533, 2006.

[3] Daniele Miorandi, Elisabeth Uhlemann, Stefano Vitturi, and Andreas Willig. Guest editorial: Special section on wireless technologies in factory and industrial automation, part i. *Industrial Informatics, IEEE Transactions on*, 3(2):95–98, 2007.

[4] Michael Lemmon, Qiang Ling, and Yashan Sun. Overload management in sensor-actuator networks used for spatially-distributed control systems. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 162–170. ACM, 2003.

[5] Bruno Sinopoli, Cory Sharp, Luca Schenato, Shawn Schaffert, and S Shankar Sastry. Distributed control applications within sensor networks. *Proceedings of the IEEE*, 91(8):1235–1246, 2003.

[6] Glenn Platt, Matt Blyde, Sean Curtin, and John Ward. Distributed wireless sensor networks and industrial control systems—a new partnership. In *Embedded Networked Sensors, 2005. EmNetS-II. The Second IEEE Workshop on*, pages 157–158. IEEE, 2005.

[7] Kay Soon Low, Win Nu Nu Win, and Meng Joo Er. Wireless sensor networks for industrial environments. In *Computational Intelligence for Modelling, Control and*

Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on, volume 2, pages 271–276. IEEE, 2005.

[8] Giuseppe Anastasi, Marco Conti, Mario Di Francesco, and Andrea Passarella. Energy conservation in wireless sensor networks: A survey. *Ad Hoc Networks*, 7 (3):537–568, 2009.

[9] Richard Zurawski. *Networked embedded systems*. CRC press Boca Raton, FL, USA, 2009.

[10] IEEE Computer Society. *IEEE Standard for Information technology, Part 15.4; Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LRWPANs)*, 2006.

[11] Alejandro Proaño and Loukas Lazos. Packet-Hiding Methods for Preventing Selective Jamming Attacks. *IEEE Transactions on Dependable and Secure Computing*, 9(1):101–114, January/February 2012.

[12] R. Daidone, G. Dini and M. Tiloca. A solution to the GTS-based selective jamming attack on IEEE 802.15.4 networks. *Wireless Networks*, 20(5):1223–1235, November 2013. IEEE 802.15.4, GTS, Security, Denial of service, Jamming, Wireless sensor networks.

[13] Matthew Gast. *802.11 wireless networks: the definitive guide*. " O'Reilly Media, Inc.", 2005.

[14] IEEE Computer Society. *IEEE Standard 802.15.1, Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Personal Area Networks (WPANs)*, 2005.

[15] Anwar Hithnawi, Hossein Shafagh, and Simon Duquennoy. Understanding the Impact of Cross Technology Interference on IEEE 802.15.4. In *Proceedings of the 9th ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization (WiNTECH'14)*, co-located with ACM MobiCom'14, Maui, HI, USA, September 2014.

[16] Thomas Watteyne, Steven Lanzisera, Ankur Mehta, and Kristofer SJ Pister. Mitigating multipath fading through channel hopping in wireless sensor networks.

In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–5. IEEE, 2010.

[17] Thomas Watteyne, Ankur Mehta, and Kris Pister. Reliability through frequency diversity: why channel hopping makes sense. In *Proceedings of the 6th ACM symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, pages 116–123. ACM, 2009.

[18] International Society of Automation (ISA). *Standard ISA -100.11a, Wireless Systems for Industrial Automation: Process Control nad Related Applications*, 2009.

[19] HART Communication Foundation Std. *HART Field Communication Protocol Specification*, 2007.

[20] IEEE Computer Society. *IEEE Standard 802.15.4e, Part 15.4: Low-Rate Wireless Personal Area Networks (LRWPANs) Amendament 1: MAC sublayer*, 2012.

[21] Iyappan Ramachandran, Arindam K Das, and Sumit Roy. Analysis of the contention access period of ieee 802.15. 4 mac. *ACM Transactions on Sensor Networks (TOSN)*, 3(1):4, 2007.

[22] Pangun Park, Piergiuseppe Di Marco, Pablo Soldati, Carlo Fischione, and Karl Henrik Johansson. A generalized markov chain model for effective analysis of slotted ieee 802.15. 4. In *Mobile Adhoc and Sensor Systems, 2009. MASS'09. IEEE 6th International Conference on*, pages 130–139. IEEE, 2009.

[23] Gang Lu, Bhaskar Krishnamachari, and Cauligi S Raghavendra. Performance evaluation of the ieee 802.15. 4 mac for low-rate low-power wireless networks. In *Performance, Computing, and Communications, 2004 IEEE International Conference on*, pages 701–706. IEEE, 2004.

[24] Jianliang Zheng and Myung J Lee. A comprehensive performance study of ieee 802.15. 4, 2004.

[25] Anis Koubaa, Mário Alves, and Eduardo Tovar. A comprehensive simulation study of slotted csma/ca for ieee 802.15. 4 wireless sensor networks. *IEEE WFCS*, 6: 63–70, 2006.

[26] Chandramani Kishore Singh, Anurag Kumar, and PM Ameer. Performance evaluation of an ieee 802.15. 4 sensor network with a star topology. *Wireless Networks*, 14(4):543–568, 2008.

[27] Giuseppe Anastasi, Marco Conti, and Mario Di Francesco. A comprehensive analysis of the mac unreliability problem in ieee 802.15. 4 wireless sensor networks. *Industrial Informatics, IEEE Transactions on*, 7(1):52–65, 2011.

[28] Marina Petrova, Janne Riihijarvi, Petri Mahonen, and Saverio Labella. Performance study of ieee 802.15. 4 using measurements and simulations. In *Wireless communications and networking conference, 2006. WCNC 2006. IEEE*, volume 1, pages 487–492. IEEE, 2006.

[29] Pangun Park, Piergiuseppe Di Marco, Carlo Fischione, and Karl Henrik Johansson. Modeling and optimization of the ieee 802.15. 4 protocol for reliable and timely communications. *Parallel and Distributed Systems, IEEE Transactions on*, 24(3):550–564, 2013.

[30] Mario Di Francesco, Giuseppe Anastasi, Marco Conti, Sajal K Das, and Vincenzo Neri. Reliability and energy-efficiency inieee 802.15. 4/zigbee sensor networks: An adaptive and cross-layer approach. *Selected Areas in Communications, IEEE Journal on*, 29(8):1508–1524, 2011.

[31] Sami M Lasassmeh and James M Conrad. Time synchronization in wireless sensor networks: a survey. In *IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the*, pages 242–245. IEEE, 2010.

[32] Bharath Sundararaman, Ugo Buy, and Ajay D Kshemkalyani. Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks*, 3(3):281–323, 2005.

[33] Ozlem Durmaz Incel, Amitabha Ghosh, and Bhaskar Krishnamachari. Scheduling algorithms for tree-based data collection in wireless sensor networks. In *Theoretical aspects of distributed computing in sensor networks*, pages 407–445. Springer, 2011.

[34] A. Proano and L. Lazos. Selective Jamming Attacks in Wireless Networks. In *Proceedings of the 2010 IEEE International Conference on Communications (ICC 2010)*. IEEE Computer Society, 2010.

[35] Maria Rita Palattella, Nicola Accettura, Mischa Dohler, Luigi Alfredo Grieco, and Gennaro Boggia. Traffic aware scheduling algorithm for reliable low-power multi-hop ieee 802.15. 4e networks. In *Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on*, pages 327–332. IEEE, 2012.

[36] Andrew Tinka, Thomas Watteyne, and Kris Pister. A decentralized scheduling algorithm for time synchronized channel hopping. In *Ad Hoc Networks*, pages 201–216. Springer, 2010.

[37] Nicola Accettura, Maria Rita Palattella, Gennaro Boggia, Luigi Alfredo Grieco, and Mischa Dohler. Decentralized traffic aware scheduling for multi-hop low power lossy networks in the internet of things. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a*, pages 1–6. IEEE, 2013.

[38] Kamin Whitehouse, Alec Woo, Fred Jiang, Joseph Polastre, and David Culler. Exploiting the capture effect for collision detection and recovery. In *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors*, pages 45–52, 2005.

[39] Benoît Latré, Pieter De Mil, Ingrid Moerman, Bart Dhoedt, Piet Demeester, and Niek Van Dierdonck. Throughput and delay analysis of unslotted ieee 802.15. 4. *Journal of Networks*, 1(1):20–28, 2006.

[40] Tae Ok Kim, Hongjoong Kim, Junsoo Lee, Jin Soo Park, and Bong Dae Choi. Performance analysis of ieee 802.15. 4 with non-beacon-enabled csma/ca in non-saturated condition. In *Embedded and Ubiquitous Computing*, pages 884–893. Springer, 2006.

[41] Mukul Goyal, Dawn Rohm, Weigao Xie, Seyed H Hosseini, Kishor S Trivedi, Yusuf Bashir, and August Divjak. A stochastic model for beaconless ieee 802.15. 4 mac operation. *Computer Communications*, 34(12):1460–1474, 2011.

[42] Piergiuseppe Di Marco, Pangun Park, Carlo Fischione, and Karl Henrik Johansson. Analytical modeling of multi-hop ieee 802.15. 4 networks. *Vehicular Technology, IEEE Transactions on*, 61(7):3191–3208, 2012.

[43] Chiara Buratti and Roberto Verdone. Performance analysis of ieee 802.15. 4 non beacon-enabled mode. *Vehicular Technology, IEEE Transactions on*, 58(7):3480–3493, 2009.

[44] Marco Gribaudo, Daniele Manini, Alessandro Nordio, and C Chiasserini. Transient analysis of ieee 802.15. 4 sensor networks. *Wireless Communications, IEEE Transactions on*, 10(4):1165–1175, 2011.

[45] Brigitte Plateau and Karim Atif. Stochastic automata network of modeling parallel systems. *Software Engineering, IEEE Transactions on*, 17(10):1093–1108, 1991.

[46] Jelena MISIC, Shairmina Shafi, and Vojislav B Misic. Maintaining reliability through activity management in an 802.15. 4 sensor cluster. *IEEE Transactions on Vehicular Technology*, 55(3):779–788, 2006.

[47] Giuseppe Bianchi. Performance analysis of the ieee 802.11 distributed coordination function. *Selected Areas in Communications, IEEE Journal on*, 18(3):535–547, 2000.

[48] Zhijia Chen, Chuang Lin, Hao Wen, and Hao Yin. An analytical model for evaluating ieee 802.15. 4 csma/ca protocol in low-rate wireless application. In *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, volume 2, pages 899–904. IEEE, 2007.

[49] Jelena Mišić, Shairmina Shafi, and Vojislav B Mišić. The impact of mac parameters on the performance of 802.15. 4 pan. *Ad Hoc Networks*, 3(5):509–528, 2005.

[50] Jelena Mišić, Shairmina Shafi, and Vojislav B Mišić. Performance limitations of the mac layer in 802.15. 4 low rate wpan. *Computer communications*, 29(13): 2534–2541, 2006.

[51] Feng Shu, Taka Sakurai, Moshe Zukerman, and Hai L Vu. Packet loss analysis of the ieee 802.15. 4 mac without acknowledgements. *Communications Letters, IEEE*, 11(1):79–81, 2007.

[52] Kiran Yedavalli and Bhaskar Krishnamachari. Enhancement of the ieee 802.15. 4 mac protocol for scalable data collection in dense sensor networks. In *Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks and Workshops, 2008. WiOPT 2008. 6th International Symposium on*, pages 152–161. IEEE, 2008.

[53] Sofie Pollin, Mustafa Ergen, Sinem Ergen, Bruno Bougard, L Der Perre, Ingrid Moerman, Ahmad Bahai, Pravin Varaiya, and Francky Catthoor. Performance analysis of slotted carrier sense ieee 802.15. 4 medium access layer. *Wireless Communications, IEEE Transactions on*, 7(9):3359–3371, 2008.

[54] Network Simulator Ns2. http://www.isi.edu/nsnam/ns/.

[55] Texas Instruments. CC2420 2.4 GHz IEEE 802.15.4 / ZigBee ready RF Transceiver. http://focus.ti.com/lit/ds/symlink/cc2420.pdf, 2012.

[56] MATLAB software. http://www.mathworks.it/products/matlab/.

[57] Bruno Bougard, Francky Catthoor, Denis C Daly, Anantha Chandrakasan, and Wim Dehaene. Energy efficiency of the ieee 802.15. 4 standard in dense wireless microsensor networks: Modeling and improvement perspectives. In *Design, Automation, and Test in Europe*, pages 221–234. Springer, 2008.

[58] Moteiv Corporation. *Tmote iv Low Power Wireless Sensor Module*. Moteiv Corporation, San Francisco, CA, USA, 2006.

[59] TinyOS. http://www.tinyos.net/.

[60] Jan-Hinrich Hauer. Tkn15. 4: An ieee 802.15. 4 mac implementation for tinyos. 2009.

[61] Cesare Alippi. *Intelligence for Embedded Systems*. Springer, 2014.

[62] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri. Just-in-time classifiers for recurrent concepts. *IEEE transactions on neural networks and learning systems*, 24(4):620–634, 2013.

[63] Cesare Alippi, Stavros Ntalampiras, and Manuel Roveri. A cognitive fault diagnosis system for distributed sensor networks. *Neural Networks and Learning Systems, IEEE Transactions on*, 24(8):1213–1226, 2013.

[64] Cesare Alippi, Giuseppe Anastasi, Mario Di Francesco, and Manuel Roveri. An adaptive sampling algorithm for effective energy management in wireless sensor networks with energy-hungry sensors. *Instrumentation and Measurement, IEEE Transactions on*, 59(2):335–344, 2010.

[65] Michèle Basseville, Igor V Nikiforov, et al. *Detection of abrupt changes: theory and application*, volume 104. Prentice Hall Englewood Cliffs, 1993.

[66] Alexander G Tartakovsky, Boris L Rozovskii, Rudolf B Blažek, and Hongjoong Kim. Detection of intrusions in information systems by sequential change-point methods. *Statistical Methodology*, 3(3):252–293, 2006.

[67] Gordon J Ross, Dimitris K Tasoulis, and Niall M Adams. Nonparametric monitoring of data streams for changes in location and scale. *Technometrics*, 53(4): 379–389, 2011.

[68] Cesare Alippi and Manuel Roveri. Just-in-time adaptive classifiers—part i: Detecting nonstationary changes. *Neural Networks, IEEE Transactions on*, 19(7): 1145–1153, 2008.

[69] Yoshinobu Kawahara and Masashi Sugiyama. Sequential change-point detection based on direct density-ratio estimation. *Statistical Analysis and Data Mining*, 5 (2):114–127, 2012.

[70] Cesare Alippi, Giacomo Boracchi, and Manuel Roveri. A hierarchical, nonparametric, sequential change-detection test. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2889–2896. IEEE, 2011.

[71] Giacomo Boracchi, Michalis Michaelides, and Manuel Roveri. A cognitive monitoring system for contaminant detection in intelligent buildings.

[72] Giacomo Boracchi and Manuel Roveri. A reconfigurable and element-wise ici-based change-detection test for streaming data.

[73] Douglas M Hawkins, QIU PEIHUA, and WOOK KANG CHANG. The change-point model for statistical process control. *Journal of quality technology*, 35(4): 355–366, 2003.

[74] Henry B Mann and Donald R Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[75] Giuseppe Anastasi, Eleonora Borgia, Marco Conti, Enrico Gregori, and Andrea Passarella. Understanding the real behavior of mote and 802.11 ad hoc networks: an experimental approach. *Pervasive and Mobile Computing*, 1(2):237–256, 2005.

[76] Andreas Willig, Martin Kubisch, Christian Hoene, and Adam Wolisz. Measurements of a wireless link in an industrial environment using an ieee 802.11-compliant physical layer. *Industrial Electronics, IEEE Transactions on*, 49(6):1265–1282, 2002.

[77] Khaled Arisha, Moustafa Youssef, and Mohamed Younis. Energy-aware tdma-based mac for sensor networks. In *System-level power optimization for wireless multimedia communication*, pages 21–40. Springer, 2002.

[78] Lodewijk FW Van Hoesel and Paul JM Havinga. A lightweight medium access protocol (lmac) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches. 2004.

[79] Venkatesh Rajendran, Katia Obraczka, and Jose Joaquin Garcia-Luna-Aceves. Energy-efficient, collision-free medium access control for wireless sensor networks. *Wireless Networks*, 12(1):63–78, 2006.

[80] Injong Rhee, Ajit Warrier, Mahesh Aia, Jeongki Min, and Mihail L Sichitiu. Z-mac: a hybrid mac for wireless sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 16(3):511–524, 2008.

[81] Sinem Coleri Ergen and Pravin Varaiya. Tdma scheduling algorithms for wireless sensor networks. *Wireless Networks*, 16(4):985–997, 2010.

[82] Nai-Luen Lai, Chung-Ta King, and Chun-Han Lin. On maximizing the throughput of convergecast in wireless sensor networks. In *Advances in Grid and Pervasive Computing*, pages 396–408. Springer, 2008.

[83] Xiang-Yang Li and Yu Wang. Simple heuristics and ptass for intersection graphs in wireless ad hoc networks. In *Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 62–71. ACM, 2002.

[84] Cédric Florens and Robert McEliece. Packets distribution algorithms for sensor networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 1063–1072. IEEE, 2003.

[85] Shashidhar Gandham, Ying Zhang, and Qingfeng Huang. Distributed minimal time convergecast scheduling in wireless sensor networks. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 50–50. IEEE, 2006.

[86] Shashidhar Gandham, Ying Zhang, and Qingfeng Huang. Distributed time-optimal scheduling for convergecast in wireless sensor networks. *Computer Networks*, 52(3):610–629, 2008.

[87] Shuguang Cui, Ritesh Madan, Andrea Goldsmith, and Sanjay Lall. Energy-delay tradeoffs for data collection in tdma-based sensor networks. In *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, volume 5, pages 3278–3284. IEEE, 2005.

[88] Meng-Shiuan Pan and Yu-Chee Tseng. Quick convergecast in zigbee beacon-enabled tree-based wireless sensor networks. *Computer Communications*, 31(5): 999–1011, 2008.

[89] Yoram Revah and Michael Segal. Improved lower bounds for data-gathering time in sensor networks. In *Networking and Services, 2007. ICNS. Third International Conference on*, pages 76–76. IEEE, 2007.

[90] Gang Lu and Bhaskar Krishnamachari. Minimum latency joint scheduling and routing in wireless sensor networks. *Ad Hoc Networks*, 5(6):832–843, 2007.

[91] Konstantinos Kalpakis, Koustuv Dasgupta, and Parag Namjoshi. Efficient algorithms for maximum lifetime data gathering and aggregation in wireless sensor networks. *Computer Networks*, 42(6):697–716, 2003.

[92] Jianlin Mao, Zhiming Wu, and Xing Wu. A tdma scheduling scheme for many-to-one communications in wireless sensor networks. *Computer Communications*, 30 (4):863–872, 2007.

[93] Tamer ElBatt and Anthony Ephremides. Joint scheduling and power control for wireless ad hoc networks. *Wireless communications, IEEE Transactions on*, 3(1): 74–85, 2004.

[94] S Chatterjea, LFW Van Hoesel, and PJM Havinga. Ai-lmac: An adaptive, information-centric and lightweight mac protocol for wireless sensor networks.

In *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004. Proceedings of the 2004*, pages 381–388. IEEE, 2004.

[95] Avinash Sridharan and Bhaskar Krishnamachari. Max-min fair collision-free scheduling for wireless sensor networks. In *Performance, Computing, and Communications, 2004 IEEE International Conference on*, pages 585–590. IEEE, 2004.

[96] Mário Macedo, António Grilo, and Mário Nunes. Distributed latency-energy minimization and interference avoidance in tdma wireless sensor networks. *Computer Networks*, 53(5):569–582, 2009.

[97] Tao Wang, Zhiming Wu, and Jianlin Mao. A new method for multi-objective tdma scheduling in wireless sensor networks using pareto-based pso and fuzzy comprehensive judgement. In *High Performance Computing and Communications*, pages 144–155. Springer, 2007.

[98] Yang Yu, Bhaskar Krishnamachari, and Viktor K Prasanna. Energy-latency tradeoffs for data gathering in wireless sensor networks. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 1. IEEE, 2004.

[99] K Pister and Lance Doherty. Tsmp: Time synchronized mesh protocol. *IASTED Distributed Sensor Networks*, pages 391–398, 2008.

[100] John N Tsitsiklis and Kuang Xu. On the power of (even a little) centralization in distributed processing. *ACM SIGMETRICS Performance Evaluation Review*, 39 (1):121–132, 2011.

[101] G. Noubir and G. Lin. Low-power DoS attacks in data wireless LANs and countermeasures. *SIGMOBILE Mob. Comput. Commun. Rev.*, 7:29–30, July 2003.

[102] L. Lazos, S. Liu and M. Krunz. Mitigating control-channel jamming attacks in multi-channel ad hoc networks. In *Proceedings of the second ACM conference on Wireless network security*, WiSec '09, pages 169–180, New York, NY, USA, 2009. ACM.

[103] W. Xu, K. Ma, W. Trappe and Y. Zhang. Jamming sensor networks: attack and defense strategies. *IEEE Network*, 20(3):41–47, May-June 2006.

[104] W. Xu, T. Wood, W. Trappe and Y. Zhang. Channel surfing and spatial retreats: defenses against wireless denial of service. In *Proceedings of the 3rd ACM workshop on Wireless security*, WiSe '04, pages 80–89, New York, NY, USA, 2004. ACM.

[105] W. Xu, W. Trappe, Y. Zhang and T. Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '05, pages 46–57, New York, NY, USA, 2005. ACM.

[106] R. Sokullu, I. Korkmaz and O. Dagdeviren. GTS Attack: An IEEE 802.15.4 MAC Layer Attack in Wireless Sensor Networks. *International Journal On Advances in Internet Technologies*, 2(1):104–114, 2009.

[107] A. Mpitziopoulos, D. Gavalas, C. Konstantopoulos and G. Pantziou. A survey on jamming attacks and countermeasures in WSNs. *IEEE Communications Surveys Tutorials*, 11(4):42–56, 2009.

[108] R.L. Pickholtz, D.L. Schilling and L.B. Milstein. Theory of Spread-Spectrum Communications - A Tutorial. *IEEE Transactions on Communications*, 20(5): 855–884, May 1982.

[109] K. Pelechrinis, C. Koufogiannakis and S. V. Krishnamurthy. Gaming the jammer: is frequency hopping effective? In *Proceedings of the 7th international conference on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, WiOPT'09, pages 187–196, Piscataway, NJ, USA, June 2009. IEEE Press.

[110] D.R. Raymond and S.F. Midkiff. Denial-of-Service in Wireless Sensor Networks: Attacks and Defenses. *IEEE Pervasive Computing*, 7(1):74–81, January–March 2008.

[111] M. Cagalj, S. Capkun and J.-P. Hubaux. Wormhole-Based Antijamming Techniques in Sensor Networks. *IEEE Transactions on Mobile Computing*, 6:100–114, January 2007.

[112] W. Xu, W. Trappe and Y. Zhang. Channel surfing: defending wireless sensor networks from interference. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, pages 499–508, New York, NY, USA, 2007. ACM.

[113] A.D. Wood, J.A. Stankovic and G. Zhou. DEEJAM: Defeating Energy-Efficient Jamming in IEEE 802.15.4-based Wireless Networks. In *Proceedings of the 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 60–69, Washington, DC, USA, June 2007. IEEE Computer Society.

[114] Y. W. Law, M. Palaniswami, L. V. Hoesel, J. Doumen, P. Hartel and P. Havinga. Energy-efficient link-layer jamming attacks against wireless sensor network MAC protocols. *ACM Transactions on Sensor Networks*, 5(1):6:1–6:38, February 2009.

[115] Arik Motskin, Tim Roughgarden, Primoz Skraba, and Leonidas J Guibas. Lightweight coloring and desynchronization for networks. In *INFOCOM*, pages 2383–2391, 2009.

[116] Tom Verhoeff. Reward variance in markov chains: A calculational approach. In *Proc. Eindhoven FASTAR Days*, 2004.

[117] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[118] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998. ISBN 0-201-89685-0.

[119] R. Daidone, G. Dini and M. Tiloca. On experimentally evaluating the impact of security on IEEE 802.15.4 networks. In *Proceedings of the 2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS 2011)*, Washington, DC, USA, 2011. IEEE Computer Society.

[120] C. K. Wong, M. Gouda and S. S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, 2000.

[121] G. Dini and I. M. Savino. LARK: A Lightweight Authenticated ReKeying Scheme for Clustered Wireless Sensor Networks. *ACM Transactions on Embedded Computing Systems*, 2011.

[122] G. Dini and M. Tiloca. HISS: a HIghly Scalable Scheme for group rekeying. *The Computer Journal*, 2013.

[123] S. Rafaeli and D. Hutchison. A survey of key management for secure group communication. *ACM Computing Surveys*, 2003.

[124] Thomas Watteyne, Xavier Vilajosana, Branko Kerkez, Fabien Chraim, Kevin Weekly, Qin Wang, Steven Glaser, and Kris Pister. Openwsn: a standards-based low-power wireless development environment. *Transactions on Emerging Telecommunications Technologies*, 23(5):480–493, 2012.

[125] David Stanislowski, Xavier Vilajosana, Qin Wang, Thomas Watteyne, and Kristofer SJ Pister. Adaptive synchronization in ieee802. 15.4 e networks. *Industrial Informatics, IEEE Transactions on*, 10(1):795–802, 2014.

[126] Adam Dunkels, Bjorn Grönvall, and Thiemo Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462. IEEE, 2004.

[127] Fredrik Österlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-level sensor network simulation with cooja. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 641–648. IEEE, 2006.