AN ANALYTICAL MODEL FOR IEEE 802.11AH NETWORKS UNDER THE RESTRICTED ACCESS WINDOW MECHANISM AND RAYLEIGH FADING CHANNEL

STEPHANIE MIRANDA SOARES

TESE DE DOUTORADO
EM ENGENHARIA ELÉTRICA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Elétrica

An Analytical Model for IEEE 802.11ah Networks Under the Restricted Access Window Mechanism and Rayleigh Fading Channel

Stephanie Miranda Soares

**TESE DE DOUTORADO SUBMETIDA AO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA DA
UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR.**

APROVADA POR:

Brasília/DF, abril de 2024.

FICHA CATALOGRÁFICA

SOARES, STEPHANIE

An Analytical Model for IEEE 802.11ah Networks Under the Restricted Access Window Mechanism and Rayleigh Fading Channel. [Brasília/DF] 2024.

204, 2024., 210 x 297 mm (ENE/FT/UnB, Doutor, Tese de Doutorado, 2024).

Universidade de Brasília, Faculdade de Tecnologia, Departamento de Engenharia Elétrica.

Departamento de Engenharia Elétrica

1. IEEE 802.11ah	2. Internet of Things
3. Wireless networks	4. Analytical modeling
5. Non-ideal channel	6. Restricted access window
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

SOARES, STEPHANIE (2024). An Analytical Model for IEEE 802.11ah Networks Under the Restricted Access Window Mechanism and Rayleigh Fading Channel. Tese de Doutorado, Publicação PPGEE.204/2024, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 108p.

Stephanie Soares

Universidade de Brasília (UnB)

Campus Darcy Ribeiro

Faculdade de Tecnologia - FT

Departamento de Engenharia Elétrica(ENE)

Brasília - DF CEP 70919-970

To my grandfather Miranda.

ACKNOWLEDGEMENTS

I am grateful to God for giving me the good health and strength to reach this point. I would like to express my sincere gratitude to my advisor Marcelo Menezes de Carvalho for his valuable teachings, patience, dedication, and comprehension. Above all, I want to thank him for always believing in me and trusting me throughout all these years of work that began during my undergraduate studies. I am also grateful to my family, especially my mother, for her love, encouragement, and unconditional support. My grandfather, who is no longer with us physically, has always been a source of love and support. The University of Brasilia (UnB) has been my second home for 14 years, and I am thankful for all the professors and colleagues who have been part of my journey.

ABSTRACT

Many promising technologies have been developed since the conception of Internet of Things (IoT). In light of the growth of IoT, the IEEE 802.11ah is an amendment to the IEEE 802.11 standard to address IoT's key challenges, which are connectivity of many devices to a single access point (AP), longer transmission ranges and limited power resources. Thus, this standard introduces new features in the Physical and MAC layers to meet IoT requirements. In the Physical layer, there were modifications to permit the operation in sub-1GHz frequency bands and, in the MAC layer the standard introduces power saving mechanisms to perform better in dense networks. One of its main novelties is the restricted access window (RAW), which is a channel access feature designed to reduce channel contention by dividing stations into RAW groups. Each RAW group is further divided into RAW slots, and stations only attempt channel access during the RAW slot they were assigned to. In this dissertation, we first propose a discrete-time Markov chain model to evaluate the average aggregate throughput of IEEE 802.11ah networks using the RAW mechanism under saturated traffic and ideal channel conditions.

The proposed analytical model describes the behavior of an active station within its assigned RAW slot. A key aspect of the model is the consideration of the event of RAW slot time completion during a station's backoff operation. The numerical results derived from our analytical model are compared to computer simulations based on an IEEE 802.11ah model developed for the ns-3 simulator by other researchers. We study the average aggregate network throughput for various numbers of RAW slots and stations in the network, and we confirm the importance of the RAW slot completion probability in our analytical model. For this, we conducted this comparison in two cases: one where this probability was considered and another where it was not considered. Moreover, our analytical model is also compared to two other analytical models proposed in the literature. The presented results indicate that the proposed analytical model reaches the closest agreement with independently-derived computer simulations.

Furthermore, we extend the discrete-time Markov chain model to consider the impact of Rayleigh fading channel and large-scale path loss on the operation of IEEE 802.11ah networks.

We also validate the proposed analytical model via ns-3 simulations, and we study network throughput in different scenarios by varying the modulation and coding schemes (MCS), packet size, and distance to the access point (AP). Our proposed analytical model accurately predicted network performance in most scenarios. However, in some instances, we observed higher percentage errors between the analytical model and simulations, which we attributed to packet losses in the simulations that the analytical model did not predict. We propose an expression to adjust the RAW slot duration based on the distance to the AP. The efficiency of the grouping strategies is measured in terms of Jain's fairness. With our case studies, we demonstrate the necessity of adjusting the RAW slot duration based on the distance between the stations and the AP to attain throughput fairness among groups.

Keywords: IEEE 802.11ah, internet of things, wireless networks, analytical modeling, non-ideal channel, restricted access window.

RESUMO

Título: Modelo analítico para redes IEEE 802.11ah sob o mecanismo de janela de acesso restrito e canal de desvanecimento Rayleigh

Muitas tecnologias promissoras foram desenvolvidas desde a concepção da Internet das Coisas (IoT, do inglês *Internet of Things*). À luz do crescimento da IoT, o IEEE 802.11ah é uma alteração ao padrão IEEE 802.11 para enfrentar os principais desafios da IoT, que são a conectividade de muitos dispositivos a um único ponto de acesso (AP, do inglês *Access Point*), transmissões a longas distâncias e recursos de energia limitados. Deste modo, o IEEE 802.11ah introduz novos recursos nas camadas Física e MAC (do inglês *Medium Access Control*) para atender aos requisitos de IoT. Na camada Física, houve modificações para permitir a operação em faixas de frequência abaixo de 1GHz e, na camada MAC, o padrão introduz mecanismos de economia de energia para melhorar o desempenho em redes densas. Uma de suas principais inovações é a janela de acesso restrito (RAW, do inglês *Restricted Access Window*), que é um recurso de acesso ao canal projetado para reduzir a contenção de canais, dividindo as estações em grupos RAW. Cada grupo RAW é dividido em slots RAW, e somente as estações atribuidas a um dado slot RAW podem tentar o acesso ao canal durante o intervalo de tempo do slot RAW. Nesta tese, propomos primeiro um modelo de cadeia de Markov de tempo discreto para avaliar a vazão média agregada de redes IEEE 802.11ah que utilizam o mecanismo RAW, sob tráfego saturado e condições de canal ideal.

O modelo analítico proposto descreve o comportamento de uma estação ativa dentro do seu slot RAW atribuído. O principal aspecto do modelo é a consideração do evento de conclusão do tempo de slot RAW durante o processo de *backoff* de uma estação. Os resultados numéricos derivados do nosso modelo analítico são comparados com simulações computacionais independentes baseadas em um módulo do IEEE 802.11ah desenvolvido para o simulador ns-3 por outros pesquisadores. Estudamos a vazão média agregada da rede para vários números de slots e estações RAW na rede e confirmamos a importância da probabilidade de término do slot RAW em nosso modelo analítico. Para isso, realizamos esta comparação em dois casos: um em que esta probabilidade foi considerada e outro em ela não foi considerada no modelo analítico. Além disso, nosso modelo analítico também é comparado com outros dois modelos analíticos propostos na literatura. Os resultados apresentados indicam que o modelo analítico proposto atinge a maior concordância com simulações computacionais derivadas independentemente.

Além disso, estendemos o modelo de cadeia de Markov de tempo discreto para considerar o impacto do desvanecimento Rayleigh e da perda de caminho em grande escala na operação de redes IEEE 802.11ah. Também validamos o modelo analítico proposto por meio de simulações ns-3 e estudamos o desempenho da rede em termos de vazão em diferentes cenários, variando os esquemas de modulação e codificação (MCS, do inglês *Modulation and Coding Scheme*), tamanho do pacote e distância ao ponto de acesso (AP, do inglês *Access Point*). O modelo analítico proposto previu com precisão o desempenho da rede na maioria dos cenários. Porém, em alguns casos, observamos erros percentuais mais elevados entre o modelo analítico e as simulações computacionais. Com as simulações computacionais, percebemos que houve perdas de pacotes as quais o modelo analítico não prevê. Propomos uma expressão para ajustar a duração do slot RAW com base na distância das estações até o AP e a eficiência das estratégias de agrupamento foi medida em termos da justiça de Jain. Com nossos estudos de caso, demonstramos a necessidade de ajustar a duração do slot RAW com base na distância entre as estações e o AP para obter justiça de acesso ao canal entre os grupos.

Palavras-chave: IEEE 802.11ah, internet das coisas, redes sem fio, modelo analítico, canal não ideal, janela de acesso restrito.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

δ	Propagation delay
σ	Empty slot
τ	Steady-state frame transmission probability
$b_{i,j}$	Steady-state probability
q	Probability of RAW slot completion
p	Probability of failed transmission
B	Bandwidth of the transmitted signal
G_r	Receive gain
G_T	Transmit gain
N_0	Noise power spectral density
P_b	Probability of bit error
P_e	Probability of packet error
P_s	Probability of successful transmission
P_{tr}	Probability of a frame transmission
P_L	Path-loss power
P_r	Received power
P_T	Transmit power
S_{BEACON}	Aggregate throughput over a beacon interval
S_{DATAi}	Throughput over a RAW slot
S_{RAW_SLOTi}	Effective throughput over RAW slot in a beacon interval
T_g	Guard period
T_h	Holding period
T_c	Time the channel is busy with a collision
T_s	Time the channel is busy with a successful transmission

GLOSSARY

AP	Access Point
AID	Associated Identifier
CSB	Cross Slot Boundary
DCF	Distributed Coordination Function
EDCA	Enhanced Distributed Channel Access
IFS	Interframe Space
EIFS	Extended Interframe Space
SIFS	Short Interframe Space
DIFS	DCF Interframe Space
PIFS	PCF Interframe Space
AIFS	Arbitration Interframe Space
IoT	Internet of Things
LPWAN	Low-Power Wide Area Networks
MCS	Modulation and Coding Schemes
MIMO	Multiple Input Multiple Output
RAW	Restricted Access Window
RPS	RAW Parameter Set
TIM	Traffic Indication Map
TWT	Target Wake Time
WPAN	Wireless Personal Area Networks

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

Nowadays, wireless networks are becoming increasingly pervasive, extending their reach to uncharted territories such as farms, manufacturing plants, retail stores, warehouses, and autonomous vehicles, to name a few, aiming to provide access to a multitude of objects and unleash the full potential of the *Internet of Things* (IoT) (AL-FUQAHA *et al.*, 2015; MINOLI *et al.*, 2017; YAQOOB *et al.*, 2017; SISINNI *et al.*, 2018; BELLINI *et al.*, 2022; QUY *et al.*, 2022; MISRA *et al.*, 2022). This unprecedented number of over-the-air connections poses significant challenges to the design of next-generation wireless networks due to the need to support *i*) a massive amount of concurrent connections and data traffic; *ii*) highly heterogeneous and stringent quality-of-service (QoS) requirements, and *iii*) efficient and fair use of scarce network resources (e.g., energy and bandwidth) (TIAN *et al.*, 2021).

In this effort, it is envisaged that Wireless Local Area Networks (WLANs) will play an essential role in the deployment and dissemination of IoT applications because Wi-Fi devices have been widely adopted and can operate in unlicensed spectrum (AHMED *et al.*, 2022). However, current Wi-Fi networks (e.g., IEEE 802.11ac/ax) face many challenges in IoT scenarios, consisting of large-scale networks deployed over areas wider than traditional WLANs. Moreover, as competing stations increase, traditional IEEE 802.11 networks become less efficient due to a higher probability of data packet collisions, leading to significant performance degradation. Additionally, due to the growth of the number of IoT devices and the fact that they are primarily devices with limited power resources, one of the significant concerns in IoT implementations is to extend the lifetime of the batteries in the devices. Powering IoT devices with conventional batteries requires frequent replacements, and billions of batteries will be discarded every year. The limited lifetime of conventional batteries can become a significant problem in networks with many connected devices because it increases maintenance and operation costs and has a negative impact on the environment. Given that medium access control (MAC) layer operations have a significant energy consumption in wireless communication networks, techniques for reducing the energy consumption of these operations are widely studied.

Currently, there are two existing low-power IoT communication technologies: *Wireless Personal Area Networks* (WPAN) and *Low-Power Wide Area Networks* (LPWAN). The WPAN technologies (such as Zig-Bee and Bluetooth Low Energy) provide medium data rates of up to a few hundred kilobits per second in a short range (tens of meters). On the other hand, LPWAN technologies (such as LoRa, SigFox, NB-IoT, eMTC, Wi-SUN, and IEEE 802.11ah) focus on long-range communications up to tens of kilometers and support low or medium data rates, from a few hundred bits per second to a few megabits per second. Regarding WPAN, Zig-Bee is developed based on IEEE 802.15.4 and supports many devices and extensive coverage using a mesh topology, while Bluetooth Low Energy consumes less energy. Regarding LWPAN, NB-IoT and eMTC are 5G technologies designed for IoT and operate in licensed frequency bands, while others work in the *Industrial, Scientific and Medical* (ISM) bands. LoRa and e-MTC support high mobility and long-range transmissions. The e-MTC supports critical service due to the high reliability and low latency, while SigFox has the most extended transmission range. The WPAN and LPWAN technologies have some limitations. WPAN has a short transmission range, and both WPAN and LPWAN have low data rates. Thus, they are only useful in limited IoT scenarios. Therefore, there is still a need for a low-power IoT communication technology that offers sufficient throughput up to tens of megabits per second over medium transmission ranges. The IEEE 802.11ah, marked as Wi-Fi HaLow, was introduced as a LPWAN technology to fill this gap, as it has the highest data rate and medium transmission range between WPAN and most of the LPWAN technologies.

Motivated by such IoT features, the IEEE 802.11ah amendment (IEEE STD 802.11AH-2016, 2016) was developed to operate on sub-GHz frequency channels in order to support higher network coverage. In addition, it introduces a number of other features that allow up to 8,191 stations to be associated with a single *access point* (AP) under higher energy efficiency, which makes it an attractive wireless access technology for IoT applications. The IEEE 802.11ah standard inherited the physical layer of the IEEE 802.11ac, with adaptations to operate at frequencies below 1 GHz, and channel bandwidths ranging from 1 MHz to 16 MHz, with the 1 MHz and 2 MHz bands being widely adopted. In particular, many new features were incorporated into the MAC sub-layer in order to provide better support for energy consumption and channel access under dense network scenarios, such as the mechanisms of *restricted access window* (RAW), *traffic indication map* (TIM), and *target wake time* (TWT), along with reduced frame headers, and faster association and authentication procedures. In fact, the RAW mechanism is one of the main innovations to handle dense scenarios, by which stations are grouped into "RAW groups," which are further divided into RAW slots. Only stations assigned to a given RAW slot can compete for channel access during the occurrence of this specific time slot. The assignment of stations to RAW groups and RAW slots, as well as their number and time duration, are all specified by the base station in the so-called *RAW Parameter Set* (RPS) transmitted in the beginning of each beacon interval (whose time duration can also be changed dynamically). Additionally, The AP creates the groups of stations using *associated identifier* (AID) numbers, which are first assigned to stations during association. However, the AID can also be changed dynamically, either by request of the station itself or by initiative of the AP.

The development of analytical models to analyze the performance of these protocols is crucial in ensuring the efficiency and effectiveness of communication in such networks. It is possible to analyze network behavior by varying parameters and initial conditions using an analytical model instead of real devices. Therefore, it is important to develop analytical models that predict the behavior of networks with high accuracy. Due to the importance and growing interest in this standard, a number of works have proposed analytical models to evaluate the performance of IEEE 802.11ah networks, especially with respect to the operation of the RAW mechanism. However, in general, previous analytical models have either included features that are not specified in the IEEE 802.11ah standard (ZHENG *et al.*, 2014; SANGEETHA; BABU, 2019), or they have not been validated against some independently-developed simulator that is widely adopted by the networking community. In fact, many previous works have validated their results based on their own customized simulators, whose accuracy and compliance to standard specifications have not been demonstrated (KHOROV *et al.*, 2019; BANKOV *et al.*, 2020; TARAMIT *et al.*, 2022a; DONG *et al.*, 2016). In this sense, the validation of an analytical model against a standard-compliant well-known simulator is key to establish its prediction accuracy. In addition, few works have addressed the impact of channel errors in the operation of IEEE 802.11ah networks and, more importantly, on its implications in the creation of RAW groups (ZHENG *et al.*, 2014; SANGEETHA; BABU, 2019; KHOROV *et al.*, 2019; BANKOV *et al.*, 2020; DONG *et al.*, 2016).

Another line of research related to IEEE 802.11ah is station grouping ideas to optimize the use of the RAW mechanism to improve network performance. Recently, the study of multi-rate IEEE 802.11ah networks has received attention due to the "performance anomaly" issue (HEUSSE *et al.*, 2003): co-located low data rate stations deteriorate throughput of high data rate ones due to unfair airtime usage. Some previous works mainly focused on grouping stations according to the modulation and coding scheme (MCS) reported to the AP during association (MAHESH *et al.*, 2020; SANGEETHA; BABU, 2020; BADARLA; HARIGOVINDAN, 2021). However, the proposed solutions do not address some issues such as: some devices may be mobile (or be relocated) during operation and data rates announced during association may change after some time; they did not consider grouping stations with similar channel conditions as a parameter, since the successful reception of frames depends on the received signal strength at the access point (AP), which impacts retransmissions; other traffic patterns (e.g., data frame lenght, interarrival time distribution, periodicity) should be taken into account when grouping stations, even if the announced MCS modes are the same. Therefore, there is still a need for station grouping proposals in order to improve airtime fairness. Thus, to deal with these issues, the AP needs to learn the dynamics of the network in real time, so it can group (or regroup) stations according to some criteria. To allow the dynamic grouping of stations, we have implemented the dynamic AID feature into the ns-3 simulator and studied its use based on a K-means approach to group stations on-the-fly that re-allocate stations to new RAW groups according to the dynamic changes in the network. Part of this work is presented in (OLIVEIRA *et al.*, 2022).

Given the importance of filling these gaps, in our previous work (SOARES; CARVALHO, 2019), we modeled the behavior of an IEEE 802.11ah station inside its assigned RAW slot based on a discrete-time Markov chain for ideal channel conditions and saturated traffic scenario. The main innovation of the proposed model is the modeling of the probability of RAW slot completion time during the activity of a station, which aims to capture the moment that the station needs to stop its activity and wait for the next assigned RAW slot in the subsequent beacon interval. For that, we presented two empirical proposals for this probability of RAW slot completion. The first expression considered that this probability depends only on the number of retransmission attempts, and the second expression also considered the impact of the number of stations within the RAW slot. Based on the proposed Markov chain, we derived an expression for the aggregate average throughput (i.e., considering all RAW slots within a RAW group), whose numerical results were compared to simulation results using the ns-3 simulator based on the 802.11ah module developed by Le Tian et al. (TIAN *et al.*, 2016). The model numerical results showed similar behavior compared to the independent ns-3 simulations with few discrepancies.

Afterwards, we realized that the duration of a RAW slot can greatly influence the probability of RAW slot completion time. As the RAW slot duration decreases, the chances of a slot ending during a transmission is greater or it may not even be enough for a transmission. Therefore, the probability of RAW slot completion time should be greater when the RAW slot duration is shorter. In this dissertation, we propose a new expression for this probability including the impact of RAW slot duration. We used the same Markov chain model and throughput computation, however, we modify the probability of RAW slot completion time expression to characterize how the RAW slot duration influences this probability. We investigate the impact of the probability of RAW slot time completion in numerical results by comparing them with the case when such probability is not taken into account (i.e., removed from the model). We also compare our analytical model with two other analytical models proposed in literature. The results obtained in this work were published (SOARES; CARVALHO, 2022).

Additionally, we extend the model to account for non-ideal channel conditions under Rayleigh fading. We have validated this model using ns-3 simulations and evaluated channel effects in different scenarios. We consider various modulation and coding schemes (MCS), packet sizes, and distances from the access point (AP) to assess their impact on the system's performance. Furthermore, we propose an expression for the minimum duration of the RAW slot of each group to ensure that stations in the furthest groups have sufficient time to transmit their packets. This is particularly important since these stations are more likely to spend more time in backoff or re-transmissions due to channel errors. Finally, we evaluate the grouping strategies in terms of Jain's Fairness to identify the most effective strategies in order to improve the required performance metrics, such as throughput and fair channel allocation for stations with worse channel conditions.

1.2 OBJECTIVES

As objectives of this dissertation, we model the behavior of a station based on the IEEE 802.11ah standard based on its backoff process, according to the distributed coordination function (DCF) for saturated traffic conditions, and propose an analytical model using Markov chain for ideal and non-ideal channel conditions. In addition, we introduce the moment of the RAW slot time completion to the Markovian model and derive expressions for the average aggregate throughput of the network.

Additionally, we evaluate channel effects in different scenarios by varying the coding and modulation scheme (MCS), packet size and distance from the access point (AP), and show the importance of assigning access time to the channel according to the distance of the station from the AP. Finally, we validate the throughput performance predicted by the analytical model with computer simulations ((TIAN *et al.*, 2016)).

1.3 CONTRIBUTIONS

The main contributions of this dissertation are the following:

- The analytical model we previously introduced (SOARES; CARVALHO, 2019) is extended and detailed to foster studies on IEEE 802.11ah network planning, optimization, and performance evaluation by other researchers without the need to rely on lengthy simulations;

- An improved heuristic expression for the probability of RAW slot time completion is introduced;

- The key role of the probability of "RAW slot time completion" is demonstrated via comparison with the case when it is not considered in the model;

- Numerical results of the analytical model are compared with ns-3 simulations based on the well-known IEEE 802.11ah module independently developed by Le Tian et al. (TIAN *et al.*, 2016), (TIAN *et al.*, 2018);

- The prediction accuracy of the proposed analytical model is shown to be significantly better than two other analytical models available in literature.
- The analytical model is extended to the case of Rayleigh fading channel;
- Numerical results of the analytical model are compared to ns-3 simulations;
- The grouping of stations is studied according to MCS modes, data frame length (payload), and distance from the AP;
- An expression to define the duration of the RAW slot in each group is proposed according to the distance of stations to the AP;
- Heterogeneous allocation of RAW slot duration is evaluated in terms of throughput fairness per group.

1.3.1 Publications

- SOARES, Stephanie M.; CARVALHO, Marcelo M. An analytical model for the aggregate throughput of IEEE 802.11ah networks under the restricted access window mechanism. Sensors, v. 22, n. 15, p. 5561, 2022. (SOARES; CARVALHO, 2022)
- OLIVEIRA, Eduardo C.; SOARES, Stephanie M.; CARVALHO, Marcelo M. K-Means Based Grouping of Stations with Dynamic AID Assignment in IEEE 802.11ah Networks. In: 2022 18th International Conference on Mobility, Sensing and Networking (MSN). IEEE, 2022. p. 134-141. (OLIVEIRA *et al.*, 2022)

1.4 ORGANIZATION OF THE DISSERTATION

Chapter 2 provides an overview of the main features of the IEEE 802.11ah used in this work, while Chapter 3 discusses related works. Chapter 4 contains our analytical model, in which we considered saturated traffic, ideal channel conditions, and, later, non-ideal channel conditions. In Chapter 5, we present the numerical results. Finally, the conclusions are in Chapter 5.3.

CHAPTER 2

OVERVIEW OF THE IEEE 802.11AH

2.1 INTRODUCTION

The release of the IEEE 802.11ah standard has introduced several new features that aim to facilitate the widespread deployment of IoT applications. The standard defines the Physical and MAC layers, which characteristics are a combination of the IEEE 802.11 standard with low-power communication technologies features, such as ZigBee and Bluetooth. It was developed to achieve communication with a range up to 1Km, while keeping the data throughput around 150Kbps, offering greater area coverage with a throughput considerably higher than the throughput obtained in low power technologies. In the MAC layer, several characteristics were introduced in order to meet IoT conditions, among them, reduced header, faster association and authentication, restricted access window (RAW), traffic indication map (TIM) and target wake time (TWT). As in traditional IEEE 802.11 power save mode, time is divided into beacon intervals, which can have one or more RAW groups, and each RAW group also has its time divided into one or more RAW slots, which are assigned to stations. Only the stations assigned to a given RAW slot can attempt to access the channel during that RAW slot. The beacon intervals can also have periods that are not occupied by RAW groups, in which all stations in the network can attempt channel access. Each beacon interval is preceded by the *RAW Parameter Set* (RPS) beacon that carries RAW information. The RPS specifies characteristics such as which stations belong to the RAW group, the RAW duration, and the number of RAW slots. In addition, the standard defines the *Cross Slot Boundary* (CSB) that indicates if a data packet transmission may exceed the RAW slot duration or not. If the CSB is disabled, a data packet transmission time cannot cross the RAW slot limit. Otherwise, a data packet transmission is permitted to cross the current RAW slot boundary even if it occupies the channel for the next RAW slot period. With the CSB disabled, the RAW slot is divided into two periods: the free access period, in which the stations can attempt to access the channel, and the holding period

T_h, which is a period that is unavailable for stations to contend for channel access because there is not enough time for a packet transmission. The standard also defines a guard period T_g between RAW slots to prevent a transmission from one RAW slot to overlap the following RAW slot due to the propagation delay. In this chapter it will be presented the main IEEE 802.11ah features used in this work.

2.2 THE IEEE 802.11AH STANDARD

Thinking about the future scenarios of wireless communication, IEEE 802.11ah was developed to support higher network coverage (up to 1 km) and a larger number of devices (up to 8,191 stations) associated to a single AP. In addition, it is more efficient in terms of energy consumption, what makes it an attractive communication protocol for IoT applications. In comparison to the IEEE 802.15.4 protocol (AHMED *et al.*, 2016), the current protocol for low-power devices with restrictions on power and memory consumption, the IEEE 802.11ah performs better in scenarios with many connected devices. The IEEE 802.11ah inherited the physical layer from IEEE 802.11ac, with adaptations to operate at frequencies below 1GHz (which differs according to country regulations $863 - 868$MHz in Europe and $902 - 928$MHz in North America, for example)(ADAME *et al.*, 2014). The bandwidth ranges from 1MHz to 16MHz, with the 1MHz and 2MHz bands being widely adopted. By operating at lower frequencies and a narrower band, IEEE 802.11ah achieves greater distances with lower power consumption than traditional Wi-Fi, which uses a frequency of 2.4GHz and bandwidth of 5MHz. The 802.11ah inherited 10 Modulation and Coding Schemes (MCSs) configurations with different data rates and reliability according to Table 2.1. Like 802.11ac, 802.11ah uses OFDM (Orthogonal Frequency Division Multiplexing), MIMO (Multiple Input Multiple Output) and DL MU-MIMO (Dowklink Multi-User MIMO) (KHOROV *et al.*, 2015). Additionally, the data frames should be encoded using Binary Convolutional Coding (BCC) or Low Density Parity Check (LDPC) coding.

The release of the IEEE 802.11ah standard has introduced several new features that aim to facilitate the widespread deployment of IoT applications. In the MAC layer, several characteristics were introduced in order to meet IoT conditions, among them, reduced header, faster association and authentication, dynamic AID, restricted access window (RAW), traffic

Table 2.1. IEEE 802.11ah MCSs for 1MHz and 2MHz with guard interval $GI = 8$ μs and data rate in Kbps

MCS Index	**Modulation**	**Coding Rate**	**Data Rate (1 MHz)**	**Data Rate (2 MHz)**
0	BPSK	1/2	300	650
1	QPSK	1/2	600	1300
2	QPSK	3/4	900	1950
3	16-QAM	1/2	1200	2600
4	16-QAM	2/3	1800	3900
5	64-QAM	1/2	2400	5200
6	64-QAM	3/4	2700	5850
7	64-QAM	5/6	3000	6500
8	256-QAM	3/4	3600	7800
9	256-QAM	5/6	4000	Not valid
10	BPSK	1/2 with $2\times$ repetition	150	Not valid

indication map (TIM) and target wake time (TWT). For channel access, in order to avoid collision and obtain a higher throughput in dense networks, the IEEE 802.11ah introduces the restricted access window, RAW, which divides stations into RAW groups. The time is divided into intervals, each of which is assigned to a RAW group, and only stations belonging to the RAW group can access the channel at the given interval. More details on how RAW works will be given in section 2.3. There is also the TIM segmentation mechanism which divides the information into several segments to transmit them separately and, along with the TIM beacon, there is also the *Delivery Traffic Indication Map (DTIM)* beacon . The access point sends the DTIM beacon in broadcast to the stations which have TIM segments with pending data, in this way, the stations wake up only to hear their corresponding TIM beacon, staying longer in mode of energy saving. The reduction of power consumption can be even greater for stations that rarely have packets to send using the TWT mechanism. With TWT, stations can negotiate with the access point when they will send their packet, so they remain in energy saving mode for a longer period.

2.3 RESTRICTED ACCESS WINDOW (RAW)

In order to reduce the number of collisions when many stations compete for the channel simultaneously, in RAW mechanism, stations are divided into RAW groups and only stations that belong to the group can access the channel within the RAW specified time interval. As well as traditional IEEE 802.11 power save mode, time is divided into beacon intervals and each beacon interval can have more than one or more RAW groups with different parameters. Moreover, each RAW group also has its time divided into one or more RAW slots, which are assigned to stations. Only the stations assigned to a given RAW slot can attempt to access the channel during that RAW slot. The beacon intervals can also have periods that are not occupied by RAW groups, in which all stations in the network can attempt channel access. Each beacon interval is preceded by the RAW Parameter Set (RPS) beacon that carries RAW information. The RPS specifies characteristics such as which stations belong to the RAW group, the RAW duration, and the number of RAW slots. The RAW mechanism is shown in Fig. 2.1, which shows that a beacon interval can have one or more RAW groups, and each RAW group can be divided into one or more RAW slots.

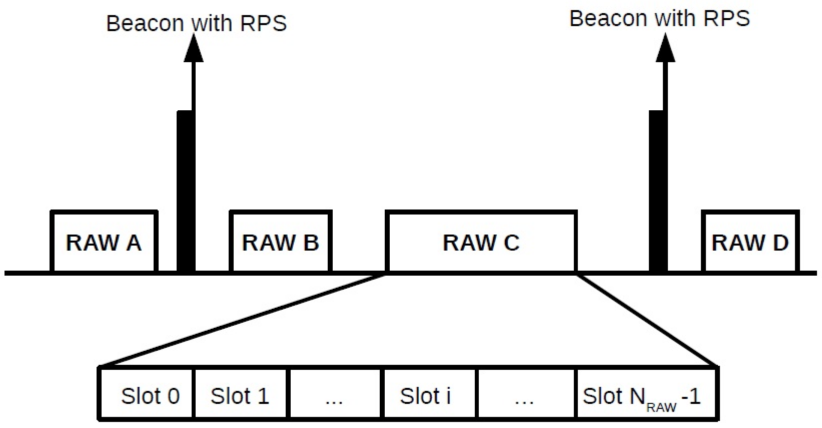

Figure 2.1. RAW mechanism of IEEE 802.11ah. Each beacon interval is preceded by a RPS beacon. The beacon interval can have one or more RAW groups, and each RAW group can be divided into one or more RAW slots.

Stations that belong to a group are required to have sequential AIDs, defined by a start AID and end AID. Moreover, the stations in a RAW group are evenly split (using round robin assignment) in the RAW slots. The stations are mapped to RAW slots as follows:

$$i_{slot} = (x + N_{\text{offset}}) \bmod N_{RAW}, \tag{2.1}$$

where i_{slot} is the index of the RAW slot to which the station is assigned, x is the AID of the station, N_{offset} is the offset value in the mapping function to improve fairness, and N_{RAW} is the number of RAW slots in one RAW group. The RPS beacon also contains the slot format duration count and slot format information, which are necessary to determine the RAW slot duration D, given by

$$D = 500\mu s + C \times 120\mu s, \tag{2.2}$$

where C represents the slot duration count subfield, which is $y = 11$ bits long when the slot format subfield is set to 1 or $y = 8$ when the slot format is set to 0. In addition, the number of slots field is $14 - y$ bits long. If $y = 11$, each RAW group consists of at most 8 RAW slots, and the maximum value of C is $2^{11} - 1 = 2047$. When $y = 8$, each RAW group consists of up to 64 RAW slots and the maximum value that C can assume is $2^8 - 1$.

Unlike the other IEEE 802.11 standards, each station uses two backoff functions according to the Enhanced Distributed Channel Access (EDCA). The first backoff function is used in periods which are not occupied by RAW groups, and all stations in the network are allowed to freely access the channel. The second backoff function is activated inside the RAW slot the station was assigned to. The stations execute the first backoff function during the period of free access to the channel, and suspend it in the beginning of the occurrence of a RAW group. When a RAW group period starts, the stations that are not assigned to this RAW group go to sleep state and save the state at which their first backoff function has stopped in, so they can resume it at the end of the time allocated to this RAW group. The stations assigned to the RAW group wake up and start the second backoff function within their assigned RAW slot. When the assigned RAW slot ends, the stations go back to sleep for the rest of the RAW group period. If the RAW slot ends during execution of the second backoff process, the stations must discard the state in which they were stopped and start a new backoff function in the next assigned RAW slot. Inside its assigned RAW slot, the station selects a random backoff interval according to the binary exponential backoff (BEB) algorithm.

Fig. 2.2 shows an example of when the two different backoff functions are activated. Station 1 belongs to the single RAW group shown in the figure, between beacons, where it is assigned to RAW slot 1. Suppose that there is a second station not assigned to this RAW group. Then, when the occurrence of the RAW group begins, both stations stop the first backoff counter and go to sleep. During the occurrence of the RAW group period, station 1 wakes up to perform the second backoff within RAW slot 1, and sleeps during the remaining RAW period, while station 2 remains in sleep state during the whole RAW group period because it does not belong to this

RAW group. Both stations return the execution of the first backoff function at the end of the time allocated to the RAW group.

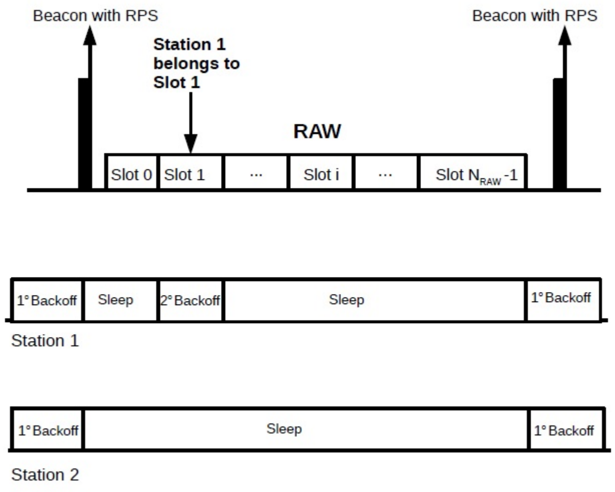

Figure 2.2. Backoff procedure inside a RAW group. Station 1 belongs to the single RAW group, and it is assigned to RAW slot 1, while station 2 does not belong to this RAW group. Both stations execute the first backoff outside the RAW group period. When the occurrence of the RAW group begins, both stations go to sleep state. Station 1 wakes up and performs the second backoff within RAW slot 1, and sleeps during the remaining RAW group period. Station 2 remains in sleep state during all RAW group period.

According to the standard (IEEE STD 802.11AH-2016, 2016), a transmission that occurs in a RAW slot may or may not exceed the RAW slot limit. In the case where transmission is allowed to cross the slot, the CSB (Cross Slot Boundary) is enabled, otherwise the CSB is disabled. When the CSB is disabled, the station will only start its transmission if there is enough time to successful transmit its packet and receive an ACK, which we call the holding time T_h. In addition to the holding time, there is also the guard interval T_g between RAW slots to prevent a transmission from one RAW slot to overlap the following RAW slot due to the propagation delay and synchronization errors.

2.4 ENHANCED DISTRIBUTED CHANNEL ACCESS (EDCA)

The IEEE 802.11ah MAC layer provides EDCA services through DCF (Distributed Coordination Function) services. As an extension of DCF, the EDCA assigns traffic priority to stations and the station with higher traffic priority is more likely to transmit than a station with lower traffic priority. The EDCA supports up to eight priorities on a station, which are mapped into four different access categories. Thus, the minimum and maximum content windows depend on each access category and the station must wait for an AIFS time, which is shorter than the DIFS of the traditional DCF. In the model and simulations, the stations have the same traffic priority, so the stations operate under the traditional IEEE 802.11 DCF.

The DCF is based on the CSMA/CA (Carrier Sense Multiple Access Protocol with Collision Avoidance) protocol where the station must wait an IFS (Interframe Space) period of time to transmit its packet after realizing that the channel is idle. The IEEE 802.11 Standard classifies these time intervals between frames according to their purpose, as specified below.

- SIFS (Short Interframe Space): Time interval between frames used for control frames, which have higher priority,i.e., a control frame is transmitted after the end of the SIFS;
- PIFS (PCF Interframe Space): Used in Point Coordination Function (PCF) networks in which there is no contention to access the channel. The coordinator (AP) defines in which time slot the station will transmit;
- DIFS (DCF Interframe Space): Minimum time interval between frames in a network with contention based medium access, in the distributed coordination function. A station will only transmit its frame if the channel is idle for a time interval equal to or greater than DIFS;
- EIFS (Extended Interframe Space): It does not have a fixed size and is used when transmission errors occur;
- AIFS (Arbitration Interframe Space): Time interval used by stations that have traffic priority in the EDCA.

Therefore, a station that has a packet to transmit checks the channel activity, if the channel is idle for a time interval greater than or equal to DIFS, the station will transmit its message and wait for a period equal to SIFS to receive a confirmation that its message arrived correctly at the receiver. If the channel is busy, the station waits for a random backoff interval chosen according to the exponential backoff algorithm when it notices that the channel is idle again. In this way, the protocol minimizes the probability of collision with packets transmitted by other stations. The station also waits for the random backoff between the transmission of two consecutive packets even if the channel is idle for a period equal to or greater than DIFS.

2.4.1 Carrier Sense Multiple Access with Collision Avoidance - CSMA/CA

The CSMA-CA allows packets from multiple devices to be transmitted on the same channel. Before starting a transmission, the station checks the channel activity. If the channel is idle, the station must wait for a DIFS time interval to start its transmission. However, if the channel is busy, the station will choose a random value of backoff and count down as soon as channel is idle. If the station notice that the channel is busy during the backoff process, the backoff counter will be frozen until the channel is idle again. Finally, when the counter reaches zero, the station will transmit its entire frame. If a collision occurs, the station can try to retransmit the packet until it succeeds or until it reaches the limit of retransmission attempts defined in the standard. Figure 2.3 describes the CSMA/CA operation.

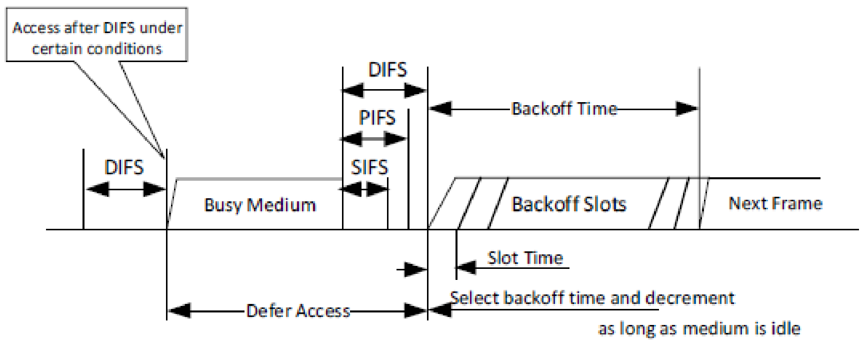
Access after DIFS under certain conditions
DIFS
PIFS
SIFS
DIFS
Backoff Time
Busy Medium
Backoff Slots
Next Frame
Slot Time
Defer Access
Select backoff time and decrement as long as medium is idle

Figure 2.3. CSMA/CA operation (IEEE STD 802.11AH-2016, 2016)

The CSMA/CA also can use short control frames to avoid collision between hidden terminals packets. These control frames are the RTS (Request to Send), request to send, and the CTS (Clear to Send), ready to send. Before transmitting a frame, the station transmits a broadcast RTS to reserve the channel and the access point responds with a CTS informing that a transmission will occur in this channel. The CTS frame is heard by all stations within range of the access point, so one station transmits its data frame and all other stations do not transmit even if the transmitting station is not within their range. The RTS/CTS frames helps to reduce collisions between packets, improving performance mainly in the transmission of longer frames. If a collision between RTS/CTS frames occurs, the time interval which the channel will be occupied by a collision will be shorter. However, the use of RTS/CTS may increase packet delay and use more device resources, which can reduce throughput and consume more energy. For this reason, RTS/CTS is most commonly used to reserve the channel for longer frames transmission.

2.4.2 Binary Exponential Backoff (BEB) Algorithm

As mentioned before, the DCF uses the binary exponential backoff (BEB) algorithm which sets it according to

$$\textit{Backoff time} = \textit{Random()} \times \textit{Slot time}, \tag{2.3}$$

where $Random()$ generates an integer value uniformly distributed in the interval $[0, CW - 1]$, where $CW_{min} \leq CW \leq CW_{max}$, and CW_{min} and CW_{max} are the minimum and maximum contention window sizes, respectively. According to the BEB algorithm, when the channel is idle, the backoff counter is decremented and, if the channel is busy, this counter freezes until the channel is idle again for a time interval greater than or equal to DIFS. Initially, the contention window size is set to the minimum value and, for each unsuccessful transmission attempt, the minimum contention window increases exponentially according to $CW = 2^i CW_{min}$, where $i \in [0, m]$ and m is the maximum number of re-transmission attempts, i.e., $CW_{max} = 2^m CW_{min}$. Finally, when the backoff counter reaches the value zero, the station can transmit its data packet.

2.5 DYNAMIC AID

In the association procedure, the IEEE 802.11ah AP assigns the Associated Identifier (AID) to the station with a value between 1 and 8191 (TIAN *et al.*, 2021). The AID in IEEE 802.1ah is a 13-bit long number that is uniquely assigned to identify each associated station and it follows an hierarchical organization in a four-level structure with 2-bit pages, 5-bit blocks, 3-bit subblocks and 3-bit stations. Stations are divided into N_p pages with N_b blocks each. Each block is divided into 8 subblocks and each subblock has 8 stations. N_p and N_b can be configured, with the maximum of $N_p = 4$ and $N_b = 32$. An example of the Hierarchical AID structure is depicted in Figure 2.4.

A station is in a RAW group if its AID is between the *RAW Start AID* and the *RAW End AID* from the RAW Group subfield in the RAW assignment of the RPS element. Therefore, all stations in the same RAW group must have the AID in the same range. Once all stations

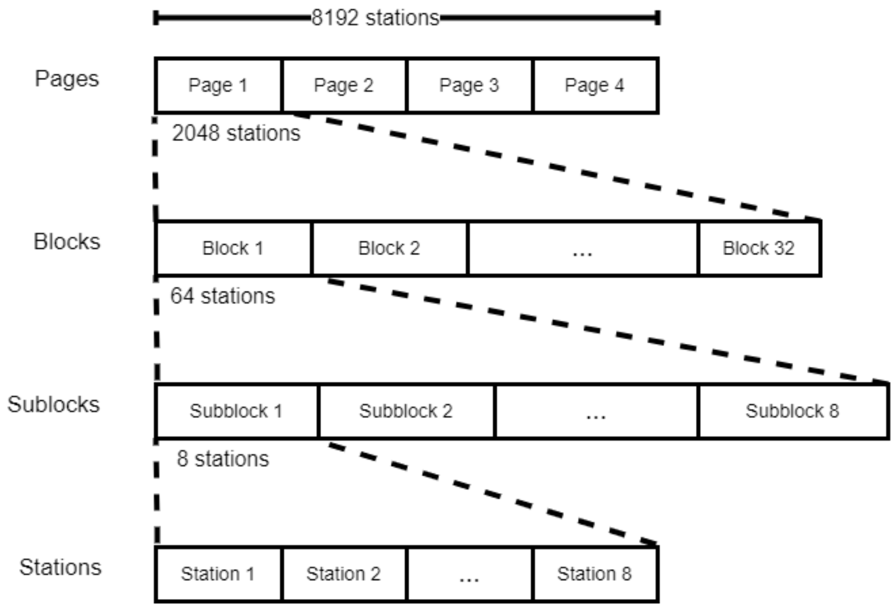

Figure 2.4. Example of hierarchical AID in IEEE 802.11 ah networks.

have their AIDs assigned in the association, it is not possible to shuffle stations between different RAW groups due to the aforementioned AID requirement. To overcome this limitation, the IEEE 802.11ah amendment supports dynamic AID assignment. A station that supports dynamic AID may send an *AID Switch Request* frame to the AP to request a new AID to change it to another group of stations when it notices changes in its traffic load or service characteristic (IEEE STD 802.11AH-2016, 2016). An IEEE 802.11ah access point may send an *AID Switch Response* frame to stations with no need of receiving an AID Switch Request. It is only possible if the station supports Unsolicited Dynamic AID, and it must set the Unsolicited Dynamic AID subfield to 1 in the S1G Capabilities.

2.6 CONCLUSIONS

The IEEE 802.11ah standard was developed to be used in Internet of Things networks. The standard operates in sub-1GHz frequencies to be able to transmit over long distances and supports up to 8191 stations associated with a single AP. The IEEE 802.11ah standard proposes a set of new features for the MAC layer to improve energy efficiency, and reduce latency and packet collision. One of the main features is the RAW mechanism, which divides stations into groups to access the channel. This chapter presented the main theoretical concepts of the IEEE 802.11ah standard used in the development of this work.

CHAPTER 3

RELATED WORK

Recently, there have been several works on different aspects of IEEE 802.11ah. The analytical modeling of IEEE 802.11ah networks using the RAW mechanism and station grouping strategies have been topics of growing interest in the past few years. As far as its analytical modeling is concerned, there are some different approaches. Some works used mean value approach, others used discrete-time Markov chain approach, and there is a large amount of works that proposed stations grouping strategies to optimize the IEEE 802.11ah performance. In the following, we present the main analytical models proposed to date.

Zheng et al. (ZHENG *et al.*, 2014) proposed an analytical model based on mean value analysis (MVA) to estimate the average network throughput under saturated traffic conditions. They considered both the cases for the *cross slot boundary* (CSB) mechanism: if enabled, it allows a station to proceed with a packet transmission that can exceed the station's allocated RAW slot time. Otherwise, a Markov chain model was proposed to estimate the number of mini-slots occupied by the last data transmission in the RAW slot, so that the duration of the next RAW slot can be calculated. They validated their analytical model with simulation results drawn from their own customized simulator. Qutab-ud-din et al. (DIN MUHAMMAD *et al.*, 2015) analyzed the performance of the RAW mechanism when the CSB is disabled and proposed four possible backoff schemes in the holding period to improve the throughput and energy efficiency of saturated IEEE 802.11ah networks. They used Zheng's (ZHENG *et al.*, 2014) analytical model as a benchmark to validate their simulation model developed in the OMNET++ simulator. Therefore, no new analytical model was actually proposed.

Mahesh and Harigovindan (MAHESH; HARIGOVINDAN, 2019) also used Zheng's model to propose a grouping scheme based on the average transmission time requirements of devices with the assignment of a priority level to each defined group. Then, they extended Zheng's analytical model to deal with the average throughput of a single group. They evaluated the average aggregate data transfer by comparing their proposed grouping scheme with uniform grouping using the ns-3 simulator. Taramiti et al. (TARAMIT *et al.*, 2022a) developed an analytical model which consider Rayleigh-fading channel with the presence of capture effect. As Zheng et al. (ZHENG *et al.*, 2014), their model based on Mean Value Analysis (MVA) and they also developed a process to estimate the number of transmissions that occurred within a RAW slot. The capture model considers the station's distance to the AP and the power attenuation of the received packets. They evaluated the impact of different parameters, without proposing a strategy to define RAW parameters or grouping methods to improve network efficiency. Furthermore, their results is based on MATLAB, which does not implement asynchronous and individual operations of nodes in a network as in the ns-3.

Raeesi et al. (RAEESI *et al.*, 2014b) proposed an analytical model that assumes saturated traffic conditions to estimate throughput and energy consumption for both basic and four-way handshake mechanisms. They investigated the performance gains obtained with the RAW mechanism compared to the regular operation of the distributed coordination function (DCF). However, their model does not consider the number of nodes as a parameter, and the probability of packet collisions is an *input* value, not a function of the number of nodes and channel contention. Moreover, the RAW mechanism is not represented in the model. Raeesi et al. (RAEESI *et al.*, 2014a) also worked on the scenario of multiple IEEE 802.11ah access points with a relatively high number of associated stations. Unfortunately, the proposed analytical model only deals with a theoretical maximum throughput that assumes no packet collisions (i.e., the backoff contention window never increases) and, again, the RAW mechanism is not incorporated in the model.

Park et al. (PARK *et al.*, 2014) developed an algorithm to estimate the number of devices in the uplink access of IEEE 802.11ah networks in order to determine the optimal duration of the RAW slot. However, they used the original analytical model for the IEEE 802.11 DCF provided by Bianchi (BIANCHI, 2000) to represent the behavior of traffic-saturated stations in IEEE 802.11ah network. Therefore, the analytical model does not take into account the RAW mechanism explicitly. Moreover, they compared the performance of the uplink total success probability delivered by their algorithm with the legacy (fixed) scheme via analysis and simulations, based on their own simulator.

Sangeetha et al. (SANGEETHA; BABU, 2019) adopted a discrete-time Markov chain model for the backoff operation of a station in its assigned RAW slot under saturated and non-saturated traffic conditions, using Bianchi's approach (BIANCHI, 2000) to compute the average throughput. Their model focuses on the case when the CSB is disabled, and assumes that the backoff counter stops at the end of the RAW slot to resumes its operation in the next assigned RAW slot, starting from the *frozen* backoff counter value. Thus, when the backoff counter reaches the value 1, the station checks whether the remaining time within the RAW slot is enough to transmit the whole data packet. If not, the station goes to a "defer state" and waits for its next assigned RAW slot in the following beacon interval. However, the described freezing of the backoff counter value is not defined in the standard. Instead, the station is supposed to discard the backoff counter value at the end of a RAW slot and start a new one in the beginning of the next RAW slot. The proposed analytical model was validated based on the ns-3 simulator. Later, they have extended their analytical model to address the grouping of stations according to traffic priority (SANGEETHA; BABU, 2021).

Ali et al. (ALI *et al.*, 2019) proposed a discrete-time Markov chain model for IEEE 802.11ah to evaluate the performance of heterogeneous IoT networks with different QoS traffic demands under non-ideal channel. They model channel effects regarding Bit Error Rate (BER) but did not consider Rayleigh fading. Moreover, they considered the case of a disabled CSB, and used the same idea as Sangeetha et al. (SANGEETHA; BABU, 2019) regarding a "defer state" (alternatively named as "delay state") in case the remaining time within a RAW slot is not enough for data packet transmission. Nevertheless, unlike Sangeetha's work, the backoff counter is *reset* in the beginning of every RAW slot. In addition, similar to our previous work (SOARES; CARVALHO, 2019), they defined a state transition probability to capture the ending of a RAW slot at any given moment during a station's backoff operation, which makes it return to an initial state for the following RAW slot. Unfortunately, their work did not provide an expression to compute this probability or a technique to estimate it, which makes their work unsuitable for comparison.

Khorov et al. (KHOROV *et al.*, 2015) proposed a discrete-time Markov chain model to find the probability distribution of the time needed for an arbitrary station to successfully transmit its frame. Based on that, they developed the probability distribution for the time needed by all stations to transmit their packets successfully, in order to determine the minimum duration of a RAW slot to improve channel efficiency. Later, they extended their work to a more general scenario (KHOROV *et al.*, 2019), with various traffic patterns and RAW configurations, and analyzed network performance in terms of throughput, energy consumption, and packet loss ratio. For that, they proposed a Markov chain model based on the time slot status, i.e., whether it is occupied by a successful transmission, packet collision, or idle state, and defined absorbing conditions to model the completion time of a RAW slot. They validated their model based on a simulator designed by themselves, i.e., not widely adopted by the networking community.

Nawaz et al. (NAWAZ *et al.*, 2017) presented a model where a RAW group is divided into two sub-groups and the duration of RAW slots in each sub-group is chosen according to the size of the group to improve network throughput. They showed that overall throughput is improved by assigning a relatively smaller RAW slot duration to a larger size group. They also used Bianchi's model (BIANCHI, 2000) to compute the network throughput, and validated the model with their own simulator. Finally, Kai et al. (KAI *et al.*, 2019) used Bianchi's approach to formulate throughput and energy efficiency expressions, and designed a traffic distribution-based grouping scheme to balance the energy efficiency and fairness guarantees among groups in heterogeneous IEEE 802.11ah networks. They proposed the Optimal Traffic Grouping Algorithm (OTGA) based on an integer nonlinear programming problem and validated their solution with simulations carried out on a simulator designed by themselves.

With regard to station grouping ideas, optimization proposals for the protocol performance are also being extensively explored in IEEE 802.11ah related works. In this dissertation, we focus on those who consider heterogeneous networks. Chang et al. (CHANG *et al.*, 2015) argued that the grouping mechanism performance is related to the different traffic demands of the devices, so the groups must be adapted to the demands of traffic. They proposed a greedy algorithm to add the sensor to the best group that can produce the maximal incremental gain of channel utilization to deliver data frames. They considered the payload size and sampling rate as parameters of their grouping scheme and they did not consider any change of these parameters during network operation.

Sangeetha and Babu (SANGEETHA; BABU, 2020) addressed heterogeneous data rates in IEEE 802.11ah networks by proposing a data rate grouping algorithm for the initialization phase, when stations associate with the AP. The AP gathers data rate information included in the PLCP header of association request frames sent by stations and associates the stations to specific arrays according to the data rate value. In addition to consider only a single parameter in their grouping scheme (data rate), the approach is static and it does not allow AID modifications on the fly if stations change their MCS parameters due to mobility or channel conditions. Moreover, their analytical model and performance evaluation do not explicitly take into account the impact of the physical layer and, consequently, the location of nodes in the terrain.

In a similar approach, Mahesh et al. (MAHESH *et al.*, 2020) proposed a grouping scheme to resolve the performance anomaly in multi-rate IEEE 802.11ah networks. For that, they proposed a data rate-based algorithm where the AP gathers the data rate of each station from the PLCP header of association request frames, and place the stations in a matrix of data rates, where each row in the matrix represents a group of stations with the same data rate. They assume the AP regroups stations if a station changes data rate. However, once assigned to a group, the station is not allowed to change groups without changing its AID. Furthermore, their grouping scheme does not consider other parameters, such as received signal power and payload size.

Badarla and Harigovindan (BADARLA; HARIGOVINDAN, 2021) also presented a solution for multi-rate IEEE 802.11ah networks based on a matrix where each row represents a group of stations with the same data rate. Similar to (SANGEETHA; BABU, 2020), the matrix is populated only once, during the association phase, and, therefore, their solution does not allow changes in the rows during operation in case the stations change their MCS parameters. The AID is fixed for the whole operation of the network. The main difference is the proportional allocation of channel time to stations with higher data rates. To accomplish that, they allocate more RAW slots to groups with higher data rates. However, they assume that stations can choose the RAW slot, which is not allowed in the standard.

Kai et al. (KAI *et al.*, 2019) designed a traffic distribution-based grouping scheme to balance energy efficiency and fairness guarantees among groups in heterogeneous IEEE 802.11ah networks. Their proposal consists of grouping stations according to each traffic demand (packet size and sampling rate) in order to ensure fairness in terms of energy efficiency. For that, they developed a heuristic traffic mapping algorithm to achieve sub-optimal energy efficiency with max-min fairness among groups. Mosavat-Jahromi et al. (MOSAVAT-JAHROMI *et al.*, 2019) also proposed a grouping scheme based on a max-min optimization problem to control the transmission and collision probabilities corresponding to each group in order to maximize the minimum per-station throughput. As this is an NP-hard problem, and it is difficult to find an optimal solution, they used the Ant Colony Optimization method to find a sub-optimal solution faster and with less complexity. However, the proposals (KAI *et al.*, 2019) and (MOSAVAT-JAHROMI *et al.*, 2019) do not allow regrouping of stations if traffic conditions change during the operation.

Considering a sensor network scenario with heterogeneous packet transmission intervals under certain periodicity, Tian et al. (TIAN *et al.*, 2017) designed the Traffic-Aware RAW Optimization Algorithm (TAROA) to adapt RAW parameters, such as number of stations and slot duration, in real time, according to traffic conditions. In every beacon interval, TAROA regroups the stations based on the estimated packet transmission interval of each station using packet transmission information obtained by the AP in the past beacon interval, and the optimal number of stations of a RAW group in order to improve network throughput. The optimal number is derived from simulations based on the ns-3 simulator under saturated state for different data rates and payload sizes of the stations. They considered in their evaluation only stations with the same MCS and payload size. Although the algorithm allows RAW configuration dynamically, the AID of the stations remains fixed throughout network operation and only stations with consecutive AID numbers can stay in the same group. Their follow-up work (TIAN *et al.*, 2018) allows non-saturated conditions and stations with different MCS and average packet sizes.

Taramit et al. (TARAMIT *et al.*, 2022b) proposed an algorithm to define the optimal duration of a RAW slot, considering the number of stations, in order to maximize the gains of the RAW mechanism in terms of throughput and energy efficiency and avoid channel waste. They considered saturated traffic and Rayleigh fading channel with capture effect. In (TARAMIT *et al.*, 2023), they proposed an algorithm to define the minimum RAW slot duration to guarantee that all stations assigned to this RAW slot deliver their packets. For this, they considered that the stations have only one packet to transmit, and they also considered a Rayleigh fading channel with capture effects. Additionally, they studied the impact of the spatial distribution of the stations around the AP in the capture effect and, thus, in the minimum RAW slot duration. However, they did not propose a grouping strategy. In both works they used MATLAB for the simulations.

Lakshmi and Sikdar (LAKSHMI; SIKDAR, 2019) proposed a fair scheduling grouping scheme with different traffic patterns. Their proposal assigns weights to the groups according to the amount of service, i.e., the amount of data the stations in the group generate per second. This way, the groups that receive higher weights get more RAW slots than the groups with lower weights. Moreover, to ensure fairness between stations in a group, they propose the use of a smaller backoff contention window than the normal one. However, the proposed grouping scheme does not consider dynamic changes in traffic conditions and does not allow change of AID.

Dong et al. (DONG *et al.*, 2016) developed a grouping scheme based on the geographical location of devices to avoid collisions related to the hidden terminal problem by defining a minimum distance between the stations in a RAW slot. They considered that the packet arrival follows a Poisson distribution, and their simulations were done in MATLAB. Yoon et al. (YOON *et al.*, 2016) proposed the Hidden Matrix-based Regrouping (HMR) algorithm to address the performance degradation due to hidden node problem in 802.11ah networks. The algorithm detects hidden node pairs using the time difference between the PS-Poll transmission, and moves nodes experiencing the hidden node problem into another group. They developed an analytical model for saturated traffic conditions. However, they considered ideal channel conditions. Their simulations were done in a C++ simulator developed by themselves.

Huang and Huang (HUANG; HUANG, 2023) adopted the Registered Backoff Time (RBT) mechanism to re-schedule those STAs from the overloaded slots to the slots that are underloaded to achieve load balance between slots, in which the AP reallocate stations according to the time they spend performing backoff. The AP re-schedule those STAs from the overloaded slots to the slots that are underloaded to achieve load balance between the slots. Their results are based on computer simulations in a ns-3 simulator like developed by themselves. Mallok et al. (MALOOK *et al.*, 2023) proposed an algorithm where the AP calculates the packet transmissions and the number of transmitting stations to modify the RAW slot duration dynamically every three beacon intervals. They proposed an expression to define the minimum RAW slot duration. However, they did not develop an analytical model to model the behavior of the network. They carried out their simulations in an ns-3 simulator, and they considered unsaturated traffic. Finally, Oliveira et al. (**??**) investigated the use of AID Switch Response frames to implement dynamic grouping according to changes in the scenario. Their objective is to enhance throughput fairness among stations that are geographically distributed over some area.

Finally, in this dissertation, we model the behavior of a station within its assigned RAW slot with a discrete-time Markov chain. The main innovation of our model is the proposal of an expression to calculate the probability of the RAW slot time completion during the backoff operation. We considered saturated traffic and ideal channel, and we compared the numerical results with independent computer simulations in the ns-3 simulator using the IEEE 802.11ah module developed by Le Tian et al. (TIAN *et al.*, 2016). We published this part of the work in (SOARES; CARVALHO, 2022). Additionally, we extend the model for the case of non-ideal channels, with Rayleigh fading and large-scale path loss. We also validate our analytical model with computer simulations in an ns-3 simulator, and we study the network throughput performance in different scenarios by varying the MCS, packet size, and distance to the access point. Furthermore, we propose an expression to adjust the RAW slot duration based on the distance to the access point. We measure the efficiency of the grouping strategies in terms of Jain's fairness.

In Table 3.1, we summarize the existing works on the RAW mechanism. We describe the performance parameter that the work aims to improve, the traffic, whether it is saturated or unsaturated, if it considers a nonideal channel, and if it considers backoff reset,i.e., if it initiates a new backoff counter in each RAW slot. Finally, we describe which simulator they used for validation. Additionally, in Table 3.2, we summarize the main works that proposed grouping methods and which parameters they consider to group stations

Table 3.1. Existing research on RAW mechanism.

Reference	**Objective**	**Traffic**	**Non-ideal Channel**	**Backoff Reset**	**Simulation**
(ZHENG *et al.*, 2014)	Throughput	Saturated			OMNET++
(DIN MUHAMMAD *et al.*, 2015)	Throughput Energy efficiency	Saturated			OMNET++
(MAHESH; HARIGOVINDAN, 2019)	Throughput	Saturated			NS-3
(TARAMIT *et al.*, 2022a) (TARAMIT *et al.*, 2022b) (TARAMIT *et al.*, 2023)	Throughput Energy efficiency	Saturated Unsaturated	✓	✓	MATLAB
(RAEESI *et al.*, 2014b)	Throughput Energy consumption	Saturated			Own Simulator
(PARK *et al.*, 2014)	Successful probability	Saturated		✓	Own simulator
(SANGEETHA; BABU, 2019) (SANGEETHA; BABU, 2020) (SANGEETHA; BABU, 2021)	Throughput	Saturated			NS-3
(ALI *et al.*, 2019)	Throughput QoS	Unsaturated	✓	✓	MATLAB
(KHOROV *et al.*, 2015) (KHOROV *et al.*, 2019)	Throughput Energy Consumption Packet loss ratio	Unsaturated		✓	Own simulator
(NAWAZ *et al.*, 2017)	Throughput	Saturated		✓	Own simulator
(TIAN *et al.*, 2017) (TIAN *et al.*, 2018)	Throughput	Saturated Unsaturated	✓		NS-3
(KAI *et al.*, 2019)	Throughput Energy consumption Fairness	Unsaturated		✓	Own simulator
(CHANG *et al.*, 2015)	Channel utilization	Saturated Unsaturated	✓		Own simulator
(MOSAVAT-JAHROMI *et al.*, 2019)	Throughput Fairness	Saturated	✓	✓	Own simulator
(LAKSHMI; SIKDAR, 2019)	Throughput Fairness	Saturated			NS-3
(BADARLA; HARIGOVINDAN, 2021)	Throughput Energy consumption	Unsaturated	✓	✓	NS-3
(YOON *et al.*, 2016)	Hidden terminal	Saturated		✓	Own simulator
(DONG *et al.*, 2016)	Hidden terminal	Unsaturated	✓		MATLAB
(HUANG; HUANG, 2023)	Throughput Load balance	Saturated	Not specified	Not specified	Own simulator
(MALOOK *et al.*, 2023)	Throughput Average delay Jitter Packet delivery ratio	Unsaturated	✓	Not specified	NS-3
(SOARES; CARVALHO, 2022)	Throughput	Saturated		✓	NS-3
(OLIVEIRA *et al.*, 2022)	Throughput Fairness	Saturated	✓	✓	NS-3

Table 3.2. Existing grouping proposals.

Reference	**Grouping Parameter**
(CHANG *et al.*, 2015)	Payload size Sampling rate
(SANGEETHA; BABU, 2020)	Data rate
(MAHESH *et al.*, 2020)	Data rate
(BADARLA; HARIGOVINDAN, 2021)	Data rate
(KAI *et al.*, 2019)	Payload size Sampling rate
(MOSAVAT-JAHROMI *et al.*, 2019)	Transmission and Collision probabilities
(TIAN *et al.*, 2017)	Packet transmission interval
(LAKSHMI; SIKDAR, 2019)	Sampling rate
(DONG *et al.*, 2016)	Geographical location
(YOON *et al.*, 2016)	Geographical location
(HUANG; HUANG, 2023)	Backoff time
(OLIVEIRA *et al.*, 2022)	Data rate Geografical location Payload size

3.1 CONCLUSIONS

In this chapter, we presented the main analytical models of the IEEE 802.11ah developed in the literature. A number of works have proposed analytical models to evaluate the performance of IEEE 802.11ah networks, especially with respect to the operation of the RAW mechanism. From this Chapter, we can identify two main lines of work: those that follow a mean value analysis, as originally used by Zheng et al. (ZHENG *et al.*, 2014), and others that propose Markov chain models. In this second group, however, there are works that adopt Bianchi's (BIANCHI, 2000) original model without considering any specific features of the IEEE 802.11ah standard. These works make different assumptions regarding protocol behavior, and they treat the event of RAW slot time completion differently, which is key for comparison with our model. In general, these previous analytical models have not been validated against some independently-developed simulator that is widely adopted by the networking community. They have validated their results based on their own customized simulators, whose accuracy and compliance to standard specifications have not been demonstrated. In addition, we presented the main works which propose station grouping strategies in heterogeneous networks.

CHAPTER 4

ANALYTICAL MODEL

In this Chapter, we present an analytical model to evaluate the average aggregate throughput of a single-hop IEEE 802.11ah network that is based on a discrete-time Markov chain model. The model describes the backoff operation of a station within its assigned RAW slot. It is assumed that there are n stations in the network operating in the basic access mode, i.e., without RTS/CTS control frames, which are evenly distributed among a certain number of RAW slots. Furthermore, it is assumed ideal channel conditions (i.e., no channel errors) and saturated traffic, i.e., all stations always have a data frame ready for transmission in their buffers. The beacon interval has fixed length, and contains only one RAW group with a fixed duration, i.e., a single RAW group occupies the whole beacon interval. Therefore, the model does not consider the periods when all stations can contend for channel access altogether (in-between RAW groups). The Cross Slot Boundary (CSB) is assumed to be disabled, and a station can transmit multiple data frames in its assigned RAW slot as long as their transmission time fits into the RAW slot. Finally, based on the analytical model, we derive an expression to compute the average aggregate throughput to evaluate network throughput performance.

4.1 MARKOV MODEL FOR BACKOFF OPERATION

We model the backoff operation of a station when it is active within its assigned RAW slot according to the two-dimensional discrete-time Markov model shown in Figure 4.1, where each state represents the backoff counter value, and each line represents a backoff stage, i.e., the number of transmission attempts. Similar to the basic IEEE 802.11 DCF, in the beginning of the Markov chain, the station chooses a random backoff value if the channel is sensed idle for a time period greater than or equal to a DIFS time interval, defined in the standard. While the channel is perceived idle, the backoff counter is decremented. Otherwise, the counter is frozen while the channel is sensed busy. The station transmits its data packet when the backoff counter reaches the value 0. If a packet collision occurs, the contention window size is doubled, and the station selects another random value according to Eq. (2.3), moving to the next backoff stage, as long as the number of re-transmission attempts does not exceed the maximum number m. If the RAW slot ends in any state, the station returns to the beginning of the Markov chain.

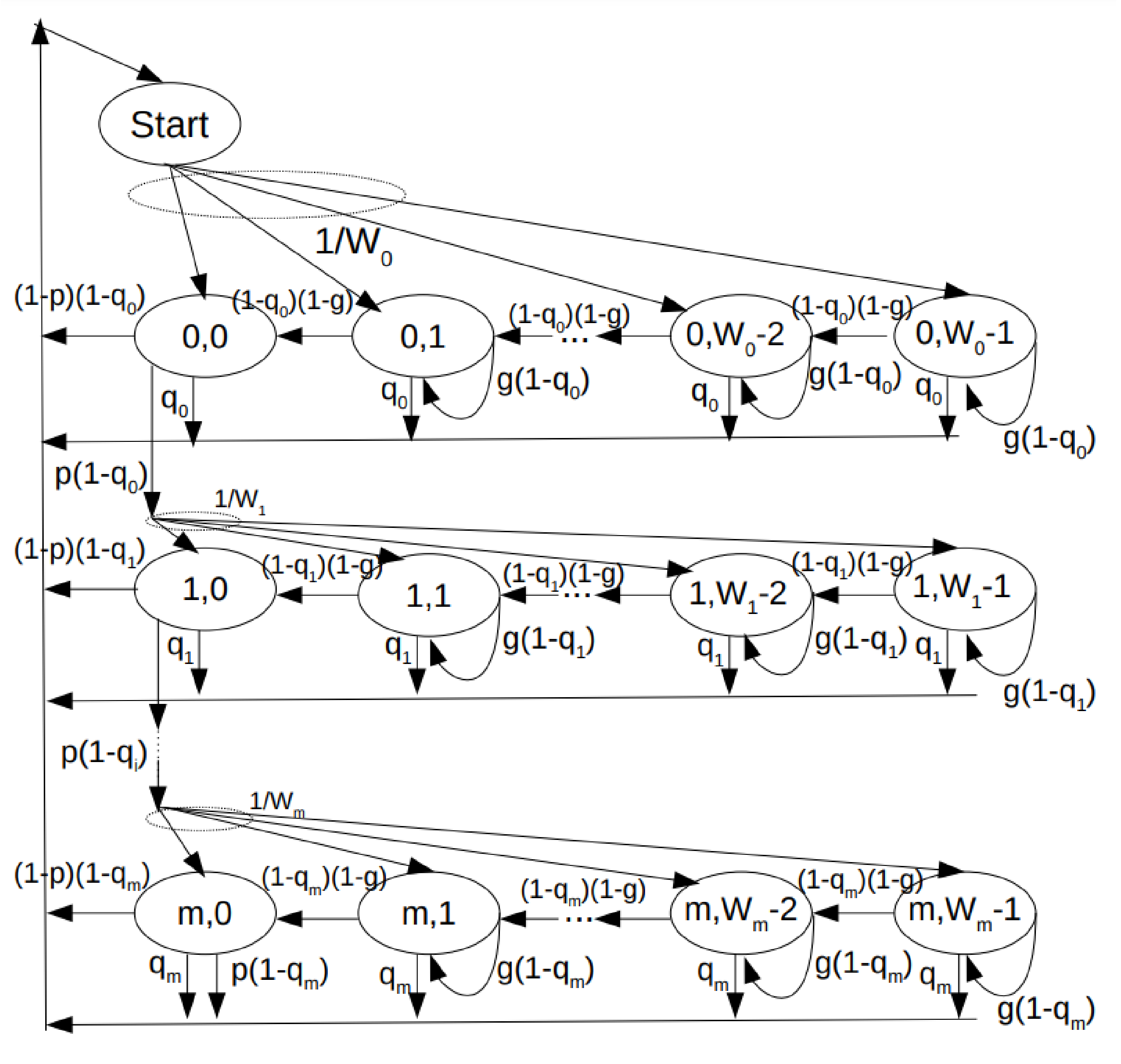

Figure 4.1. Markov chain model for station operating within its assigned RAW slot according to the IEEE 802.11ah.

Let $b(t)$ denote the stochastic process that represents the backoff counter value at time t for a station assigned to a RAW slot, and $s(t)$ denote the stochastic process that represents the backoff stage, i.e., the number of packet transmission attempts so far. Let p denote the probability of data packet collision, which is assumed to be constant and independent of the number of transmission attempts. Let g denote the “freezing” probability of the backoff counter when the channel is perceived to be busy, according to the DCF mechanism. We assume that it is constant and independent of the backoff stage. Since the RAW slot time can finish at any time during the backoff operation of a station, we represent the occurrence of this event with a transition probability q_i departing from every state $(i, j), 0 \leq i \leq m, 0 \leq j \leq Wi - 1$, and returning to the "start" state. We assume that q_i is constant at each stage i and the stages are independent of each other. We also assume that the event of RAW slot completion time is independent of the events of freezing the backoff counter value and the occurrence of packet collisions. Finally, let W_0 denote the minimum contention window size and $W_i = 2^i W_0$, $0 \leq i \leq m$. Hence, the one-step transition probabilities $P\{(i_1, j_1)|(i_0, j_0)\} = P\{s(t+1) = i_1, b(t+1) = j_1|s(t) = i_0, b(t) = j_0\}$ are given by

$$
\begin{array}{lll}
P\{(i,j)|(i,j+1)\} = (1-q_i)(1-g), & i \in [0,m], & j \in [0, W_i - 1] \\
P\{(i,j)|(i,j)\} = g(1-q_i), & i \in [0,m], & j \in [1, W_i - 1] \\
P\{(0,j)|(i,j)\} = \dfrac{q_i}{W_0}, & i \in [0,m], & j \in [0, W_i - 1] \\
P\{(0,j)|(i,0)\} = \dfrac{(1-p)(1-q_i)}{W_0}, & i \in [0,m], & j \in [0, W_0 - 1] \\
P\{(i+1,j)|(i,0)\} = \dfrac{p(1-q_i)}{W_i}, & i \in [0,m-1], & j \in [0, W_i - 1] \\
P\{(0,j)|(m,0)\} = \dfrac{p(1-q_i)}{W_0}, & i = m, & j \in [0, W_0 - 1]
\end{array}
$$

The first equation indicates that the backoff counter advances if the channel is idle and the RAW slot time is not over; the second equation indicates that the backoff counter remains in the same state (frozen) if the channel is busy and the RAW slot time is not over; the third equation contains the probability of ending the RAW slot and return to a new backoff process in the following RAW slot; the fourth equation represents a successful data frame transmission, which leads to the beginning of a new backoff operation for the next data frame in queue. The fifth equation indicates that a collision has occurred, and the station goes to the next backoff stage; the sixth equation describes the transition probability due to a failed data frame transmission in the last backoff stage: the data frame is discarded and a new backoff operation is initiated for the following data frame in the queue.

4.2 MARKOV CHAIN SOLUTION

4.2.1 Ideal Channel

Let $b_{i,j}$ denote the steady-state probability of state (i,j) in the Markov chain, i.e., $b_{i,j} = \lim_{t\to\infty} P\{s(t) = i, b(t) = j\}$, $i \in [0,m]$, $j \in [0, W_i - 1]$. Using the transition probabilities defined previously, we get

$$b_{i,j} = \begin{cases} B, & i = 0, j = W_0 - 1 \\ B \times \sum\limits_{l=0}^{W_0-(j+1)} \left[\frac{(1-g)(1-q_i)}{1-g(1-q_i)}\right]^l, & i = 0, j \in [0, W_0 - 2] \\ C_i, & i \in [1,m], j = W_i - 1 \\ C_i \times \sum\limits_{l=0}^{W_i-(j+1)} \left[\frac{(1-g)(1-q_i)}{1-g(1-q_i)}\right]^l, & i \in [1,m], j \in [1, W_i - 2] \\ \frac{p(1-q_{i-1})}{W_i} \times b_{i-1,0} \times \sum\limits_{l=0}^{W_i-1} \left[\frac{(1-g)(1-q_i)}{1-g(1-q_i)}\right]^l, & i \in [1,m], j = 0, \end{cases} \tag{4.1}$$

where

$$B = \frac{M}{W_0(1 - g(1 - q_i))}, \tag{4.2}$$

$$C_i = \frac{p(1 - q_{i-1})}{W_i(1 - g(1 - q_i))} \times b_{i-1,0}, \tag{4.3}$$

and

$$M = \sum_{i=0}^{m} \sum_{j=0}^{W_i-1} q_i b_{i,j} + \left[(1-p)\sum_{i=0}^{m}(1 - q_i) b_{i,0}\right] + p(1 - q_i) b_{m,0}. \tag{4.4}$$

Since a station transmits a packet when it reaches any state $b_{i,0}$, for $i \in [0,m]$, the steady-state packet transmission probability is given by

$$\tau = \sum_{i=0}^{m} b_{i,0} = \sum_{i=0}^{m} \frac{p(1 - q_{i-1})}{W_i} \left\{ \sum_{l=0}^{W_i-1} \left[\frac{(1-g)(1-q_i)}{1 - g(1-q_i)}\right]^l \times b_{i-1,0} \right\}. \tag{4.5}$$

If n denotes the number of stations competing for the channel in a RAW slot, then a packet collision occurs if two or more of the $n-1$ remaining stations transmit a packet at the same time. As per existing literature, we assume that all stations transmit packets independently of each other. Thus, the packet collision probability p will be given by

$$p = 1 - (1 - \tau)^{(n-1)}. \tag{4.6}$$

Similarly, from the station's point of view, we assume that the probability g of a busy channel is also the probability of having at least another node transmitting over the channel, i.e.,

$$g \cong p = 1 - (1 - \tau)^{(n-1)}. \tag{4.7}$$

The steady-state probabilities of the states of the Markov chain can then be computed by numerical calculation using the normalization condition

$$\sum_{i=0}^{m} \sum_{j=0}^{W_i - 1} b_{i,j} = 1, \tag{4.8}$$

and Eq. (4.5). Finally, we need to compute the probability q_i of RAW slot time completion, upon which the station cancels its backoff operation and waits for the next RAW slot in the following beacon interval. Given the complexity in deriving a closed-form expression for the probability of RAW slot time completion q_i, we extend our previous work (SOARES; CARVALHO, 2019) by proposing a new heuristic expression for the probability q_i, which aims to represent this probability more realistically and obtain a more accurate model. The development for computation of the probability q_i is introduced next.

4.2.2 Under Rayleigh Fading Channel

In this Section, we extend our model to consider the impact of large-scale path loss and small-scale fading propagation effects. In particular, we assume Rayleigh fading and a path-loss propagation model for outdoor macro deployments adopted for IEEE 802.11ah networks (DIN MUHAMMAD *et al.*, 2015). We maintain the assumption on saturated traffic at nodes, i.e., all stations always have a packet ready for transmission in their buffers. Furthermore, the model does not consider free access periods in beacon intervals, i.e., the beacon interval is fully occupied by RAW groups. We assume that the CSB is disabled, and a station can transmit multiple packets in its RAW slot as long as it is not over.

To incorporate physical layer effects into the Markov chain, we modify the probability p, originally defined as collision probability. Here, we consider the same Markov chain, but we refer to p as the probability of failed transmission. Assuming there is no packet capture effect, a packet is successfully transmitted if only one node transmits over the channel and no bit errors occur as a result of noise or channel propagation effects (DANESHGARAN *et al.*, 2008).

Hence, if p_s denotes the probability of successful transmission of a frame, then

$$\begin{aligned} p_s &= P\{\text{no bit errors, only 1 node TX}\} \\ &= P\{\text{no bit errors}|\text{only 1 node TX}\}P\{\text{only 1 node TX}\}. \end{aligned} \tag{4.9}$$

But, $P\{\text{only 1 node TX}\} = (1-\tau)^{(n-1)}$, where τ is the transmission probability in the Markov chain and, as usual, it is assumed that nodes transmit independently from each other. Now, if we denote $P_e = 1-P\{\text{no bit errors}|\text{only 1 node TX}\}$ then, the probability of failed transmission is given by

$$p = 1 - p_s = 1 - (1-P_e)(1-\tau)^{(n-1)}. \tag{4.10}$$

To evaluate P_e, we need the *signal-to-noise ratio* (SNR)

$$SNR = \frac{P_r}{N_o \times B}, \tag{4.11}$$

where P_r (in Watts) is the received power, N_0 is the noise power spectral density, and B is the bandwidth of the transmitted signal. The average received power P_r (in dBm) at a given distance d from the source is given by $P_r = P_T + G_T + G_r - P_L(d)$, where P_T is the transmit power, G_T and G_r are the transmit and receive antenna gains, respectively, and $P_L(d)$ is the large scale path-loss for a station in a distance d from the AP. In this dissertation, we adopt the path loss for outdoor macro deployments, which is given by (DIN MUHAMMAD *et al.*, 2015)

$$P_L(d) = 8 + 37.6\log_{10}(d) + 21\log_{10}\left(\frac{f}{900\text{ MHz}}\right), \tag{4.12}$$

where f is the carrier frequency. From the average SNR, we obtain the bit error probabilities for BPSK and M-QAM modulations (see Table 2.1) under Rayleigh fading, which are given by (GOLDSMITH, 2005)

$$P_b = \frac{1}{4\gamma_b}\ \text{(BPSK)} \quad \text{and} \quad P_b = \frac{M-1}{3\gamma_b \log_2 M}\ \text{(M-QAM)}, \tag{4.13}$$

where $\gamma_b = \frac{E_b}{N_0} = SNR \times \frac{B}{R}$, E_b is the energy per bit and R is the bit rate transmission mode.

For every data packet with a payload length of L, we can determine the probability P_e that a bit error occurs in the packet of length L. Assuming an Additive White Gaussian Noise Channel, Binary Convolutional Coding, and Hard-Decision Viterbi decoding, we can calculate P_e as

$$P_e = 1 - (1-P_u)^{8\times L}, \tag{4.14}$$

where P_u is the union bound of the first event error probability, $P_u = \sum_{d=d_{free}}^{\infty} a_d P_d$, d_{free} is the free distance of the convolutional code, a_d is the total number of error events of weight d, and P_d is the probability that the incorrect path from the correct path is chosen by the Viterbi decoder is defined by (LACAGE; HENDERSON, 2006)

$$P_d = \begin{cases} \sum_{i=(d+1)/2}^{d} \binom{d}{i} {P_b}^i (1-P_b)^{d-i} & \text{if d is odd} \\ \frac{1}{2}\binom{d}{d/2} \sum_{i=(d/2)+1}^{d} \binom{d}{i} {P_b}^i (1-P_b)^{d-i} & \text{otherwise} \end{cases} \tag{4.15}$$

4.3 RAW SLOT TIME COMPLETION PROBABILITY

The derivation of the probability of RAW slot time completion is not a trivial task because it depends on a number of events whose characterizations are not easily obtained. Because of that, we adopt a heuristic approach for its computation by considering some key aspects and assumptions. First, we notice that the higher the number of transmission attempts for a given data packet, the longer the time a station spends in backoff and, consequently, the higher the chances of RAW slot time completion during the transmission attempt of that specific data packet. Therefore, for any data packet a station attempts to transmit, we assume that the chances of RAW slot time completion increases proportionally to the number of stages traversed during backoff.

However, during execution of a given RAW slot, and under traffic saturation, the station may transmit multiple data packets while contending for channel access with other stations assigned to the same RAW slot. Hence, if we assume that the IEEE 802.11ah operation is fair, each of the n contending stations should receive an equal "share" of channel access time within a given RAW slot, i.e., each station should get a fraction $1/n$ of the total RAW slot time for channel access. In this sense, a station is unable to transmit more data packets once its "share" of channel access time is over. As a result, while the station may have performed the backoff algorithm for a number of consecutive data packets, the RAW slot will finish during backoff operation in the transmission attempt of a specific data packet. In other words, such an event determines in which packet transmission attempt the completion of the RAW slot time actually occurs.

Finally, we note that, the RAW slot to which the station was assigned occupies a certain amount of time within the beacon interval. Therefore, the ratio between the RAW slot time and the beacon time interval provides an estimate for the probability of finding the station within its RAW slot at any given time. Based on such observations and assumptions, we are interested in the probability of the event of RAW slot time completion during execution of the i-th backoff stage for a given packet, i.e., we are interested in the probability of the joint conditional event $E_i = \{$RAW slot ends, STA's fraction of RAW ends, RAW ends in this backoff stage | stage $i\}$, where STA refers to the station performing backoff. From the Markov chain in Figure 4.1, the probability $q_i = P\{E_i\}$ of RAW slot time completion during backoff stage $i \in \{0, \ldots, m\}$ of attempting a given packet transmission will be given by

$$q_i = P\{\text{RAW slot ends, STA's fraction of RAW ends, RAW ends in this stage}|\text{stage } i\}, \tag{4.16}$$

which can be rewritten as

$$\begin{aligned} q_i = {} & P\{\text{RAW slot ends}|\text{STA's fraction of RAW ends, RAW ends in this stage, stage } i\} \\ & \times P\{\text{STA's fraction of RAW ends}|\text{RAW ends in this stage, state } i\} \\ & \times P\{\text{RAWends in this stage}|\text{stage } i\}. \end{aligned} \tag{4.17}$$

Since the end of a RAW slot does not depend on the operation of any given station (its duration is defined by the access point), we have

$$\begin{aligned} & P\{\text{RAW slot ends}|\text{STA's fraction of RAW ends, RAW ends in this stage, stage } i\} = \\ & = P\{\text{RAW slot ends}\} = 1 - \frac{(T_{slot} - T_h - T_g)}{T_{BI}}, \end{aligned} \tag{4.18}$$

where T_{slot}, T_h, T_g, and T_{BI} are the RAW slot time, the holding time (assuming the CSB is disabled), the guard interval between RAW slots, and the beacon interval, respectively. Note that, the shorter the RAW slot time T_{slot} is, the higher the chances of RAW slot time completion. Now, assuming that the end of a station's fair share of channel access does not dependent on the specific backoff stage the station is currently performing, we have

$$\begin{aligned} & P\{\text{STA's fraction of RAW ends}|\text{RAW ends in this stage, stage } i\} \\ & = P\{\text{STA's fraction of RAW ends}\} = 1 - \frac{1}{n}, \end{aligned} \tag{4.19}$$

since it is assumed that each station occupies $1/n$ of the RAW slot time T_{slot}. At last, for any data packet a station attempts to transmit, the chances of RAW slot time completion are assumed to increase proportionally to the number of backoff stages traversed. Therefore,

$$P\{\text{RAW ends in this stage}|\text{stage } i\} = \frac{i}{m+1}. \tag{4.20}$$

Finally, from Eqs. (4.17), (4.18), (4.19) and (4.20), we obtain

$$q_i = \left[1 - \frac{(T_{slot} - T_h - T_g)}{T_{BI}}\right]\left(1 - \frac{1}{n}\right)\left(\frac{i}{m+1}\right). \tag{4.21}$$

4.4 THROUGHPUT COMPUTATION

4.4.1 Ideal Channel

In this section, we compute the average aggregate throughput within a beacon interval, i.e., the sum average throughput across all RAW slots contained in all RAW groups defined within the beacon interval. We assume that only the basic access mechanism is used (i.e., no RTS/CTS frames). Let us first consider the average throughput over a single RAW slot i. Using Bianchi's approach (BIANCHI, 2000), let n_i denote the number of stations within RAW slot i, and P_{tr_i} denote the probability that a frame is transmitted over the channel during the RAW slot i, i.e.,

$$P_{tr_i} = 1 - (1 - \tau_i)^{n_i}, \tag{4.22}$$

and let P_s^i denote the conditional probability of a successful DATA frame transmission within the RAW slot i, given by

$$P_{s_i} = \frac{n_i \tau_i (1 - \tau_i)^{(n_i - 1)}}{P_{tr_i}}. \tag{4.23}$$

Hence, the probability of a successful DATA frame transmission is $P_{s_i}P_{tr_i}$, the probability that a slot is empty is $(1 - P_{tr_i})$, and the probability of DATA collision is $(1 - P_{s_i})P_{tr_i}$. If $E[P]$ denotes the average length of a DATA frame payload, the throughput over a given RAW slot i is given by

$$S_{DATA_i} = \frac{P_{s_i}P_{tr_i}E[P]}{(1 - P_{tr_i})\sigma + P_{s_i}P_{tr_i}T_s + (1 - P_{s_i})P_{tr_i}T_c}, \tag{4.24}$$

where σ is the duration of a mini-slot (as defined by the standard), T_s is the time the channel is busy due to a successful DATA frame transmission, and T_c is the time interval the channel is busy due to collisions. The values of T_s and T_c are calculated according to

$$T_s = \text{DIFS} + \frac{H}{R_h} + \frac{E[P]}{R_d} + 2\delta + \text{SIFS} + \text{ACK} \tag{4.25}$$

and

$$T_c = \text{DIFS} + \frac{H}{R_h} + \frac{E[P]}{R_d} + \text{SIFS} + \text{ACK}_{TO}, \tag{4.26}$$

where H is the length of the header of a DATA frame, which contains PHY and MAC layer headers, δ is the propagation delay, R_h and R_d are the transmission rates for the header and payload fields, respectively, ACK and ACK_{TO} refer to the duration of the ACK and ACK timeout, and $DIFS$ and $SIFS$ are specified in the IEEE 802.11ah standard.

Now, we are interested in obtaining the effective throughput within a beacon interval corresponding to the data received from stations allocated to RAW slot i. In this case, we must take into account the time the stations spend waiting for the end of other RAW slots to which they were not assigned to. Note that, from the point of view of the access point, it will receive an amount of data from stations allocated to different RAW slots over a period of time. Therefore, over the duration of a beacon interval, the AP will receive a certain amount of data from stations allocated to RAW slot i. Hence, the effective throughput $S_{RAW\_SLOT_i}$, over a beacon interval, for stations assigned to RAW slot i is

$$S_{RAW\_SLOT_i} = S_{DATA_i} \times \frac{(\text{RAW slot time})_i - T_h - T_g}{\text{Beacon Interval}}, \tag{4.27}$$

where $T_h = T_s$ and T_g refer to the holding and guard periods, respectively. The average aggregate throughput S_{BEACON}, in a given beacon interval, can be obtained by adding the effective throughput $S_{RAW\_SLOT_i}$ of each RAW slot i across all RAW groups l in the beacon interval, i.e.,

$$S_{BEACON} = \sum_{l=1}^{r} \sum_{i=0}^{s_l} S_{RAW\_SLOT_i}, \tag{4.28}$$

where s_l is the number of RAW slots within each RAW group l, and r is the number of RAW groups within the beacon interval. In this work, we consider $r = 1$, i.e., the scenario of a single RAW group in a beacon interval. As a final note, Eq. (4.28) assumes that there are no free access periods between RAW groups.

4.4.2 Non-ideal Channel

Finally, to include the effect of the PHY layer in the average aggregate throughput calculation in Eq. (4.24), which considers that the unsuccessful packets were only those that suffered a collision, we have to include also the packets that suffered bit errors. Hence, the new throughput expression is

$$S_{DATA_i} = \frac{P_{s_i} P_{tr_i} E[P](1 - P_e)}{(1 - P_{tr_i})\sigma + P_{s_i} P_{tr_i} T_s + (1 - P_{s_i}) P_{tr_i} T_c}, \tag{4.29}$$

where P_{s_i} is the probability of a successful transmission and P_{tr_i} that a data frame is transmitted over the channel in a RAW slot i, σ is the duration of a mini-slot (as defined by the standard), T_s is the time the channel is busy due to a successful DATA frame transmission, and T_c is the time interval the channel is busy due to collisions. To calculate the effective throughput for the data received from stations allocated in a specific RAW slot within a beacon interval, and the effective throughput of a RAW slot over all RAW groups of the beacon interval, we used Eqs.(4.27) and (4.28), respectively.

4.5 MINIMUM RAW SLOT DURATION

It is expected that stations further away from the AP spend more time backing off before data transmission and require more retransmissions due to higher chances of bit errors. To compensate for this, RAW groups of the farthest zones should receive larger RAW slots to transmit their packets. Hence, we need to define a RAW slot duration to ensure that stations in each RAW group have enough time to send their packets and improve network fairness in terms of throughput. Given that packet transmission attempts are assumed to be independent with probability of success P_{s_i} in RAW slot i, the number of transmission attempts until success can be approximated to a geometric random variable with mean $1/P_{s_i}$. Hence, to ensure that at least one successful transmission occurs in RAW slot i, the minimum duration could be set to

$$\text{RAW slot duration} = \left(\frac{1}{P_{s_i}} + 1\right) \times T_s. \tag{4.30}$$

We use this expression is Section 5.2.3 to define the RAW slots duration.

4.6 CONCLUSIONS

In this Chapter we presented a discrete-time Markov chain analytical model to describe the behavior of a station within its assigned RAW slot under ideal channel conditions. A particular novelty of the proposed model is the inclusion of a probability to express the completion of the RAW slot time during the backoff activity of a station, which aims to capture the moment when the station needs to wait for the next RAW slot in the following beacon interval in order to resume the transmission attempts for a given data frame. For this, we proposed an empirical expression which takes into account the duration of the RAW slot, number of stations within the RAW slot and the backoff stages traversed by the station. In addition, we extended the model for the case of non-ideal channel, which consider the impact of large-scale path loss and Rayleigh fading. Finally, we derived a general expression for the computation of the average aggregate throughput over a set of RAW groups and corresponding RAW slots.

CHAPTER 5

NUMERICAL RESULTS

In this Chapter, we present the numerical results derived from our analytical model and compare them with computer simulations carried out in the ns-3 simulator based on the IEEE 802.11ah module previously developed by Le Tian et al. (TIAN *et al.*, 2016). In Section 5.1, we present the results with ideal channel conditions. We present the impact of the proposed probability of RAW slot time completion on the accuracy of the numerical results, and we compare our model with two other proposed models in the literature. Section 5.2 presents the results of the Rayleigh fading channel's impact on the throughput performance and the fairness of the network in terms of throughput. We evaluate the throughput performance in different scenarios, varying the *Modulation and Coding Schemes* (MCS), packet size, and duration of the RAW slot.

5.1 IDEAL CHANNEL

In this section, we focus on single-hop networks under ideal channel conditions and saturated traffic (i.e., all nodes always have a data frame ready for transmission in their queue). In the proposed scenario, each beacon interval has a fixed duration and contains only one RAW group with a given number of RAW slots, each with the same and fixed time duration. The RAW group, with its associated set of RAW slots, occupies the whole beacon interval. Every beacon interval has the same time duration, and the stations are evenly distributed among a given number of RAW slots. All stations are configured with UDP traffic and transmit packets with a time interval small enough to ensure that stations have saturated traffic. In addition, 10 simulation runs are performed for each scenario, and all simulations correspond to 120 seconds of operation. We present results for the average aggregate throughput by considering different numbers of stations and RAW slots per beacon interval. Table 5.1 and Table 5.2 show the parameter values used in all scenarios. The parameter values were chosen based on the IEEE

Table 5.1. Values of PHY-layer parameters used in simulations and numerical results.

Parameter	**Value**
Basic rate	1 Mb/s
Data rate	7.8 Mb/s
Channel bandwidth	2 MHz
Modulation and coding scheme	MCS8
PHY layer header	192 μs
Slot duration (σ)	52 μs
Propagation delay (δ)	3.3 μs

Table 5.2. Values of MAC-layer and other parameters used in simulations and numerical results.

Parameter	**Value**
SIFS	160 μs
DIFS	264 μs
ACK_{TO}	$2 \times \delta + SIFS + ACK$
CW_{min} / CW_{max}	16 / 1024
Payload size	256 bytes
ACK	14 bytes + PHY Header
MAC layer header	34 bytes
Beacon interval	0.1s
Guard interval (T_g)	8 μs
Holding period (T_h)	T_s

802.11ah standard (IEEE STD 802.11AH-2016, 2016), and on the simulation study carried out by Le Tian et al. (TIAN *et al.*, 2016) to validate their IEEE 802.11ah module developed for the ns-3 simulator.

In Section 5.1.1, we first present an investigation about the impact of the proposed probability of RAW slot time completion (q_i in our analytical model) on the accuracy of the numerical results with respect to the average aggregate throughput obtained via discrete-event simulations on the ns-3 simulator. This set of results is important to establish the key role that this probability plays on the accuracy of the proposed analytical model. Then, in Section 5.1.2, we present a performance comparison of predicted average aggregate throughput between our analytical model and the two other analytical models previously discussed, adopting the results derived by simulations performed on the ns-3 simulator based on the IEEE 802.11ah module developed by Le Tian et al. (TIAN *et al.*, 2016) as a benchmark.

5.1.1 Impact of RAW Slot Time Completion Probability on Throughput

In order to understand the impact of the proposed probability of RAW slot time completion on the accuracy of the analytical model, we compare the average aggregate throughput computed in two cases: when the probability of RAW slot time completion is either *present* or *absent* in the Markov Chain model of Figure 4.1 (i.e., we consider the cases $q_i = 0$ or $q_i \neq 0$, $\forall i \in \{0, \ldots, m\}$). In this section, we compute the average aggregate throughput according to Eq. (4.28) for the cases of 2, 5, and 10 RAW slots per beacon interval, and vary the total number of stations in the network from 5 to 100 nodes, evenly distributed among the RAW slots (note that, depending on the number of stations, there can be some RAW slots with fewer or no stations).

Figure 5.1 contains the results for the case when there are only 2 RAW slots within the RAW group. The blue line with circles contains the results for the case when the probability of RAW slot time completion is absent from the model, while the black curve with squares shows the results for the case when the probability of RAW slot time completion is taken into account in the model. The results from ns-3 simulations are shown in red line with asterisks. As we can see, the results derived from the analytical model without considering the probability of RAW slot time completion differs significantly from simulations, presenting very optimistic values compared to the case when such probability is considered. For purposes of comparison, the *root-mean-square error* (RMSE) computed between simulation results and the values predicted by the analytical model if the probability of RAW slot time completion is *disregarded* is $RMSE = 0.4160$ Mb/s, whereas if such probability is considered we get $RMSE = 0.0471$ Mb/s.

Figure 5.2 shows the results when there are 5 RAW slots per beacon interval. Again, we observe the strong disagreement from simulation results when the probability of RAW slot time completion is disregarded. In this case, we obtain $RMSE = 0.2352$ Mb/s if $q_i = 0$, $\forall i \in \{0, \ldots, m\}$, while $RMSE = 0.0178$ Mb/s if the probability of RAW slot time completion is considered.

Figure 5.3 depicts the results for the case when there 10 RAW slots per beacon interval. As the number of stations per RAW slot decreases, we observe higher aggregate throughput values,

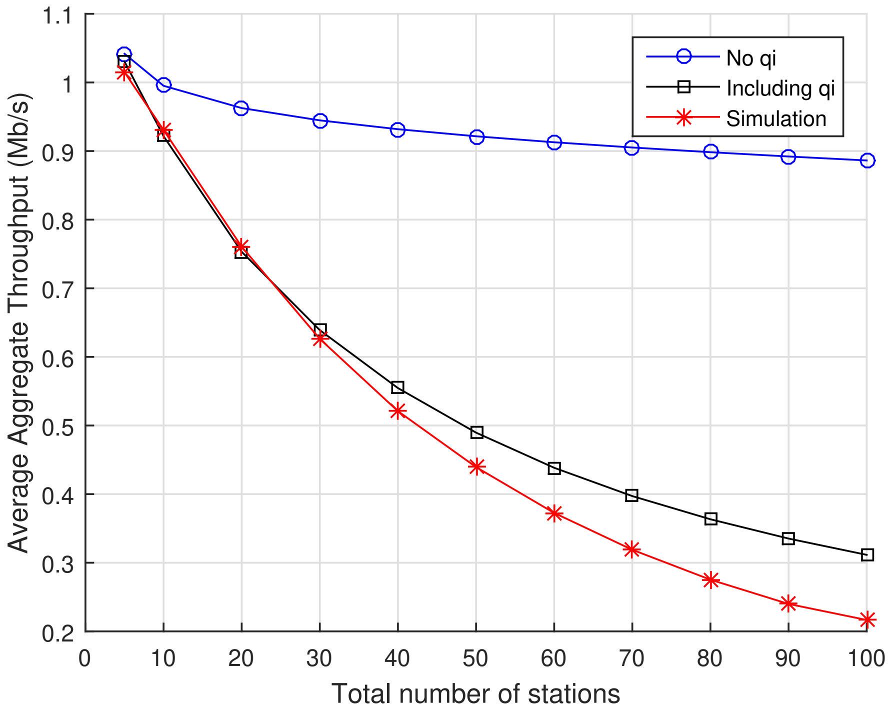

Figure 5.1. Average aggregate throughput as a function of the total number of stations divided into 2 RAW slots within a single RAW group per beacon interval. Numerical results for the analytical model are displayed for the cases when either Eq. (4.21) or $q = 0$ are considered for the probability of RAW slot time completion, along with the results obtained with ns-3 simulations.

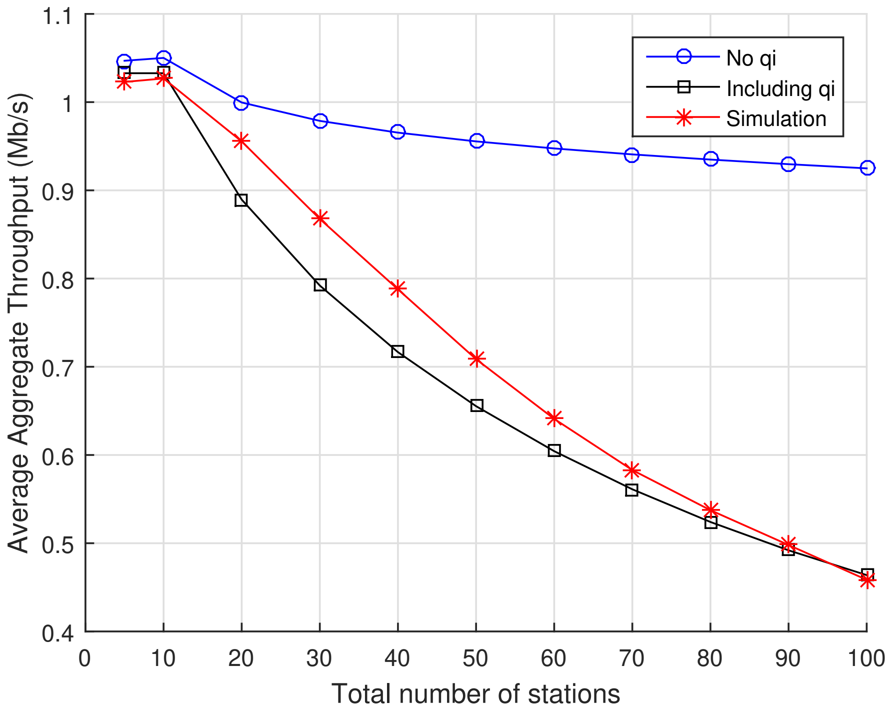

Figure 5.2. Average aggregate throughput as a function of the total number of stations divided into 5 RAW slots within a single RAW group per beacon interval. Numerical results for the analytical model are displayed for the cases when either Eq. (4.21) or $q = 0$ are considered for the probability of RAW slot time completion, along with the results obtained with ns-3 simulations.

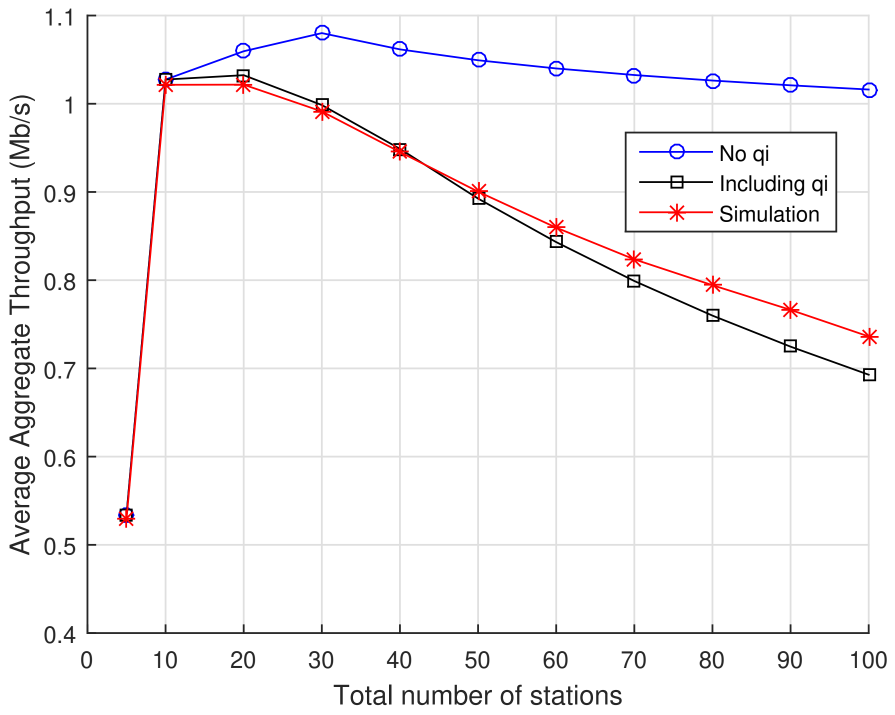

Figure 5.3. Average aggregate throughput as a function of the total number of stations divided into 10 RAW slots within a single RAW group per beacon interval. Numerical results for the analytical model are displayed for the cases when either Eq. (4.21) or $q = 0$ are considered for the probability of RAW slot completion time, along with the results obtained with ns-3 simulations.

since contention decreases at each RAW slot. In this case, $RMSE = 0.1416$ Mb/s for the case when there is no probability of RAW slot time completion, and $RMSE = 0.0124$ Mb/s if such probability is considered in the analytical model.

5.1.2 Comparison with other Analytical Models

In this section, we compare the average aggregate throughput predicted by our analytical model with the ones provided by Sangeetha (SANGEETHA; BABU, 2019) and Zheng (ZHENG *et al.*, 2014) models, considering ns-3 simulations as a benchmark. Considering the two main lines of work, mean value analysis and discrete-time Markov chain approaches, we chose Zheng et al. model because they were who originally used the mean value analysis, and we chose Sangeetha et al. model because they used discrete-time Markov chain as our model. In addition, Zheng's and Sangeetha's works make different assumptions regarding protocol behavior, and they treat the event of RAW slot time completion differently, which is key for comparison with our model. It is also worth mentioning that both analytical models make similar assumptions as ours, such as saturated traffic and ideal channel conditions, and they consider that stations are equally divided among RAW slots. Figure 5.4 depicts the results for the average aggregate throughput when there are only 2 RAW slots within a beacon interval. As we can observe, Zheng's and Sangeetha's models are in strong disagreement with the results obtained via ns-3 simulations. In particular, we observe that Sangeetha's model predicts almost no throughput decay as the total number of stations increases. In fact, this is in line with what is shown in their original work, which presents similar behavior. For Sangeetha's model we obtain $RMSE = 0.258$ Mb/s. In the case of Zheng's model, we observe a higher decay in throughput values as the number of stations increases, compared to Sangeetha's model. However, it does not show much agreement with simulations, with the exception of some specific results (e.g., when the number of stations is 40 and 50). For Zheng's model we obtain $RMSE = 0.198$ Mb/s. As discussed in Chapter 3, both models assume that the backoff process is not renewed at the beginning of a new RAW slot, i.e., the stations freeze their backoff counter values at the end of a RAW slot, and resume them in the beginning of the next RAW slot. Such feature has a major impact on their overall results. In contrast, our analytical model delivers $RMSE = 0.0471$ Mb/s as previously discussed.

Figure 5.5 contains the results for the case of 5 RAW slots within a single RAW group in the beacon interval. As expected, the average aggregate throughput increases with respect to the case with 2 RAW slots across all analytical models and simulations because channel contention decreases within each RAW slot and, therefore, more data packets can be transmitted per RAW slot. Sangeetha's model continues to present a behavior similar to that obtained for 2 RAW slots, but with higher average throughput values and $RMSE = 0.206$ Mb/s. Zheng's model continues to show the trending behavior of throughput decay as the total number of stations increases. However, it still shows considerable discrepancies with simulations, with $RMSE = 0.150$ Mb/s. For our model, the results are closer to simulations, but a bit more pessimistic in terms of throughput, especially when the number of stations is between 20 and 60. In this scenario, our model delivers $RMSE = 0.0178$ Mb/s, which means that Zheng 's RMSE is about 287% higher than ours.

Figure 5.6 contains the results for the case of 10 RAW slots within a single RAW group in each beacon interval. Note that, when $n = 5$, there are empty RAW slots in the beacon

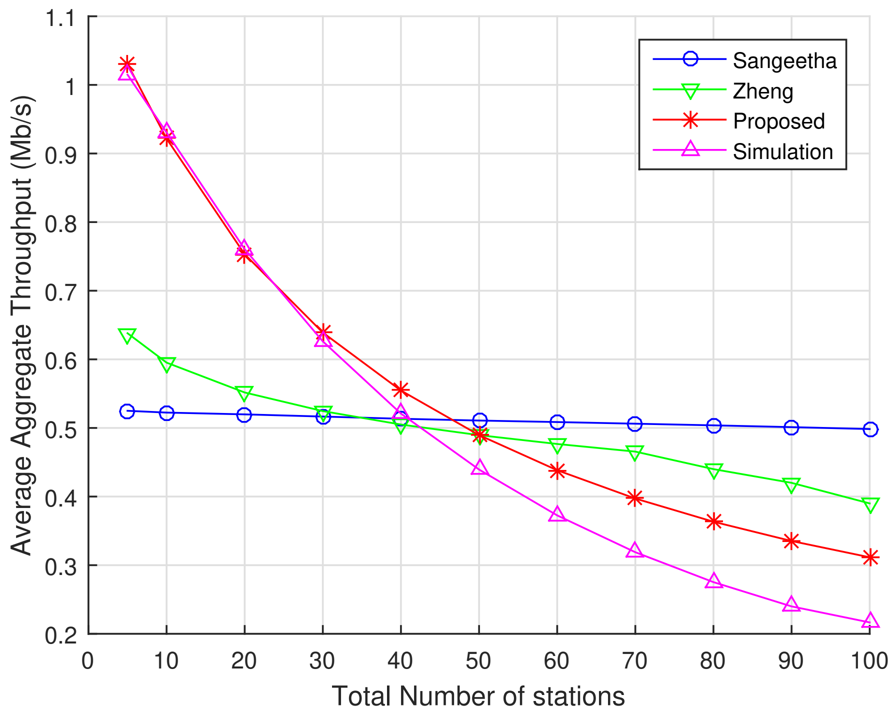

Figure 5.4. Average aggregate throughput as a function of the total number of stations divided into 2 RAW slots within a single RAW group per beacon interval. Numerical results for our proposed model, Zheng's model, and Sangeetha's model are displayed along with the results obtained with ns-3 simulations.

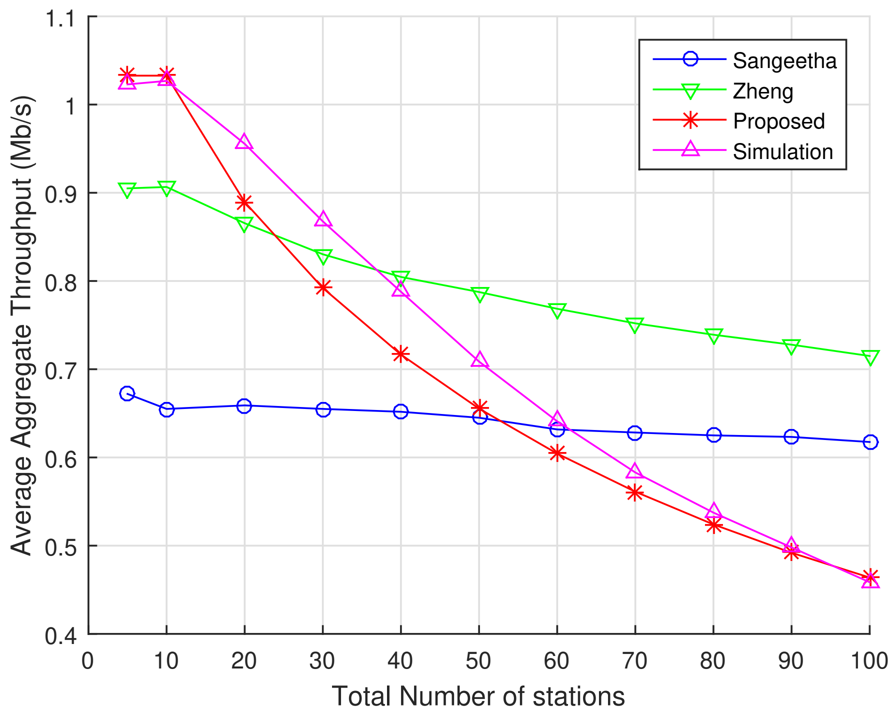

Figure 5.5. Average aggregate throughput as a function of the total number of stations divided into five RAW slots within a single RAW group per beacon interval. Numerical results for our proposed model, Zheng's model, and Sangeetha's model are displayed along with the results obtained with ns-3 simulations.

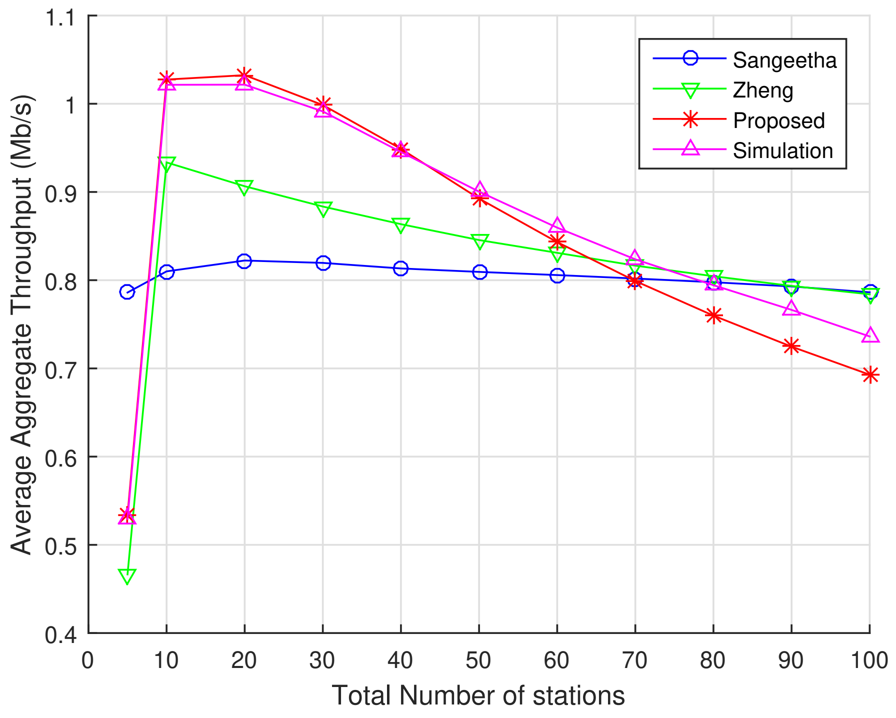

Figure 5.6. Average aggregate throughput as a function of the total number of stations divided into 10 RAW slots within a single RAW group per beacon interval. Numerical results for our proposed model, Zheng's model, and Sangeetha's model are displayed along with the results obtained with ns-3 simulations.

interval, which leads to low throughput values for the three analytical models and simulations. Similar to the other scenarios, our analytical model delivers the closest behavior to simulations, with $RSME = 0.0124$. In this case, however, the predicted throughput values are a bit more pessimistic than simulations, as the number of nodes increases. Zheng's model presents the general behavior of throughput decay as the number of stations increases. However, while it is more pessimistic than simulations for lower number of stations, it becomes more optimistic as the number of nodes increases, especially for $n \geq 80$. Zheng's model gives $RSME = 0.068$. Finally, Sangeetha's model presents the same subtle decrease in throughput as the number of stations increases, as observed in the numerical results in their work. In this case, we get $RMSE = 0.139$ Mb/s.

5.2 IMPACT OF RAYLEIGH FADING CHANNEL

In this section, we analyze the results for non-ideal channel conditions under Rayleigh fading. We evaluate the channel effects in different scenarios, by varying parameters such as MCS, packet size, and distance from the access point (AP). For this, we modify the scenario in order to evaluate the throughput performance of each RAW group and the fairness among the RAW groups in terms of throughput. All stations are in saturated traffic conditions. Additionally, we present three case studies to assess the throughput performance, while considering the distance of stations to the AP and other parameters, like transmission rate and packet size. In the three case studies, all stations in the network have packets of the same size and transmit under the same MCS. In the first and second case studies, the beacon interval is equally divided between the RAW groups, and each RAW slot has the same duration. In the third case, we define the RAW slot duration according to the distance of each RAW group.

It is expected that stations located further away from the AP spend more time in the backoff process due to path loss and Rayleigh fading. Consequently, these stations are harmed by stations closer to the AP and have more difficulty accessing the channel and sending their packets. With these three case studies, we evaluate the impact of path loss and Rayleigh fading on the throughput of each group and fairness between the stations when we randomly allocate the stations in the groups and when we allocate the stations according to their location. In addition, we show the importance of allocating different RAW slot durations for each group to ensure fairness for the stations farther away from the AP.

In all case studies, the stations are distributed over a circle around a single AP with radius 200 m, divided into four geographical zones R_i, $i \in \{1, 2, 3, 4\}$, that assume the shape of rings around the AP, as shown in Figure 5.7. This way, a station located in the region R_i is closer to the AP than a station located in region R_j if $i < j$. We consider the outdoor macro network

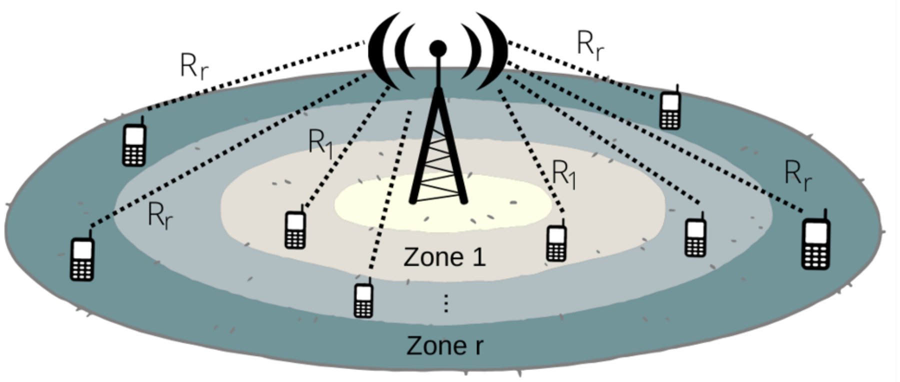

Figure 5.7. Network scenario: stations are divided into r zones around the AP.

scenario with the path-loss propagation model of Eq. (4.12), and adopt the *YansErrorModel* for the modulation and coding schemes (MCS) of IEEE 802.11ah. Table 5.3 contains the PHY-layer parameters used in simulations.

Table 5.3. Physical layer parameters used in ns-3 simulations numerical (analytical) computations.

Parameter	**Value (STA/AP)**
Frequency	900 MHz
Channel bandwidth	2 MHz
Transmit power	0 dBm / 0 dBm
Transmission gain	0 dB / 3 dB
Noise figure	6.8 dB
Propagation Loss model	Outdoor, macro
Error Rate Mode	YansErrorModel
MCS configurations	MCS0, MCS3

In the three scenarios, the beacon interval is fully occupied by RAW groups, i.e., there are no free access periods and the *cross slot boundary* (CSB) is disabled. Table 5.4 summarizes the MAC-layer parameters used in simulations numerical (analytical) computations. As in ideal channel simulations, the simulations correspond to 120 seconds of operation, and 10 simulation runs are performed for each scenario, and all stations are configured with UDP traffic.

Table 5.4. MAC layer parameters used in ns-3 simulations numerical (analytical) computations.

Parameter	**Value**
Beacon interval (ms)	199.84
RTS/CTS	not enabled
Cross slot boundary	not enabled
Total RAW time (ms)	199.84
Number of RAW groups	4
Number of RAW slots per group	2
Guard interval (T_g)	8 μs
Holding period (T_h)	T_s

In all scenarios, there are a total of 80 stations in the network, equally divided into each RAW group, and randomly placed in each region R_i, $i \in \{1, 2, 3, 4\}$. For the analytical model results, we consider the midpoint distance of each ring to compute pass-loss. We focus on the average throughput per RAW group and, for simulations, we also consider the *normalized* Jain's fairness index: if r_i and s_i denote the data rate (MCS mode) and the average throughput achieved by station i, respectively, then $u_i = s_i/r_i$ is the *normalized average throughput* achieved by station i. Therefore, for each RAW group containing $|\mathcal{C}|$ elements, the normalized Jain fairness index is given by

$$J_{\mathcal{C}} = \frac{\left(\sum_{i=1}^{|\mathcal{C}|} u_i\right)^2}{|\mathcal{C}| \sum_{i=1}^{|\mathcal{C}|} u_i^2}, \tag{5.1}$$

where $0 \leq J_{\mathcal{C}} \leq 1$, and values closer to 1 means higher fairness. As in Section 5.1, all stations are configured with a saturated UDP traffic, and are performed 10 simulation runs for each scenario with 120 seconds of operation.

5.2.1 Case 1

In this case, we present the impact of path loss and Rayleigh fading on the station-AP distance relationship for stations under the same MCS and data frame length when the stations are randomly grouped. The beacon interval was divided equally between the RAW groups; that is, the RAW groups have RAW slots with the same duration of $24.98ms$. We evaluate throughput performance and Jain´s Fairness of the network. As the stations are randomly located in the area around the AP and the groups have stations located in any of the four distance rings, and we have to choose a fixed distance to compute path loss in the analytical model, we chose the midpoint distance of 100 meters for the analytical model. In the first scenario, packet sizes are equal to 256 bytes and transmitted in MCS0 mode. Table 5.5 presents the numerical and simulation results of the average throughput, in Mb/s, per group, the standard deviation (Sta dev) of the simulations, and the Jain's Fairness.

Table 5.5. Comparison between analytical model and computer simulations of the throughput in Mb/s for MCS0 and 256 bytes with RAW slots of the same duration in each RAW group and random groups.

Group	Simulations	Sta dev	Analytical Model	Error %
1	0.2728	0.0287		11.36
2	0.2686	0.0341	0.2418	9.97
3	0.2659	0.0362		9.06
4	0.2756	0.0413		12.26
Jain's Fairness	0.5900	0.0408		

As expected, randomly assigning stations from the four regions to any of the four groups leads to similar throughput values across groups. From the results, random grouping is generally harmful to performance in this case: stations distant from the AP have less favorable channel conditions, making them more likely to undergo more retransmission attempts than stations close to the AP. As a result, the stations close to the AP are more likely to acquire the channel immediately after transmission of its data packet due to the binary exponential backoff (BEB) algorithm: a station farther away from the AP tends to spend more time in the backoff process due to more retransmission attempts. Thus, the random grouping of stations does not lead to a fair share of channel access, which we can observe from the Jain's Fairness (JF) index of $JF = 0.5900$ for this scenario. The analytical model presented a maximum percentage error of 12.26% with respect to the average results of simulations, which is acceptable since the range of distances of the stations from the AP is significant, and we chose only the distance of 100m to obtain the numerical results.

In Table 5.6, we have the results for the scenario where the stations are configured with MCS0 mode and packet length of 1024 bytes. Despite the larger packet size, we observe a slightly decrease in throughput if compared to the case where the packet size is 256 bytes and a Jain's fairness index of 0.3660. This scenario becomes less fair because, in addition to stations located further away from the AP being harmed by being allocated to the same group of stations located closer to the AP, longer packets have a greater probability of bit error. As a result, the stations farther away from the AP have more difficulty to send their packets successfully. Additionally, we notice that, in simulations, 10.88% of the stations (on average) did not transmit any packet. This explains the highest discrepancy between the analytical model and simulations.

Table 5.6. Comparison between analytical model and computer simulations of the throughput in Mb/s for MCS0 and 1024 bytes with RAW slots of the same duration in each RAW group and random groups.

Group	Simulations	Sta dev	Analytical Model	Error %
1	0.1910	0.0164		4.02
2	0.1745	0.0155	0.1990	12.31
3	0.1914	0.0067		3.81
4	0.2377	0.0167		16.28
Jain's Fairness	0.3660	0.0339		

For the next scenarios, we use MCS3, which has a higher data rate than MCS0. We use the same data frame lengths, 256 bytes and 1024 bytes. Table 5.7 shows the results for 256 bytes, while Table 5.8 shows the results for 1024 bytes.

As expected, throughput increases compared to the scenarios with MCS0 mode due to the increase in data rate. Additionally, there is a significant improvement of fairness in the scenario

Table 5.7. Comparison between analytical model and computer simulations of the throughput in Mb/s for MCS3 and 256 bytes with RAW slots of the same duration in each RAW group and random groups.

Group	Simulations	Sta dev	Analytical Model	Error %
1	0.5002	0.0430		3.10
2	0.5034	0.0492	0.5157	2.38
3	0.5362	0.0395		3.82
4	0.5133	0.0305		0.46
Jain's Fairness	0.6096	0.0338		

Table 5.8. Comparison between analytical model and computer simulations of the throughput in Mb/s for MCS3 and 1024 bytes with RAW slots of the same duration in each RAW group and random groups.

Group	Simulations	Sta dev	Analytical Model	Error %
1	0.9243	0.0115		0.88
2	0.8933	0.0133	0.9325	4.20
3	0.9621	0.0097		3.07
4	1.0378	0.0131		10.14
Jain's Fairness	0.6081	0.0333		

with the data frames of 1024 bytes. This is because increasing the data rate reduces the transmission duration, allowing more transmissions in a fixed-duration RAW slot. The analytical model shows a smaller discrepancy compared to simulations, with the highest percentage error under 11%. For the scenario with a data frame length of 256 bytes, $JF = 0.6096$, and for the scenario with data frame length of 1024 bytes, $JF = 0.6081$.

5.2.2 Case 2

In this case, we evaluate the throughput performance when each RAW group corresponds to one of the four regions R_i, and has the same duration, i.e., each RAW slot lasts 24.98 ms. For this case, the analytical model adopts the distance from the midpoint of each ring width to compute path loss. Like Section 5.2.1, packet sizes are set to 256 bytes and transmitted with MCS0 mode. Table 5.9 presents the results.

Table 5.9. Comparison between analytical model and computer simulations of the throughput in Mb/s for MCS0 and 256 bytes with RAW slots of the same duration in each RAW group.

Group	Simulations	Sta dev	Analytical Model	Error %
1	0.3074	0.0179	0.3152	2.47
2	0.2671	0.01735	0.2503	6.29
3	0.2563	0.0247	0.2344	8.54
4	0.2497	0.0195	0.2292	8.21
Jain's Fairness	0.6256	0.0267		

The results show that group 1, the closest to the access point (AP), has the highest throughput, while group 4 has the worst. This is because stations closer to the AP have better channel conditions, i.e., they are more likely to get access to the channel immediately after transmitting their data packet. In contrast, stations farther away from the AP have worse channel conditions and, therefore, are more likely to experience re-transmission attempts. Groups 3 and 4, which are the farthest groups away from the AP, present the highest percentage erros between model and simulation, and the model appears to be more pessimistic than the simulations. The model adopts the distance from the midpoint of the ring width to compute path loss, whereas, in simulations, stations are randomly distributed over the ring area and may be located in points of the ring closer to the AP. This leads to some discrepancies between the model and computer simulations. In this scenario, we have a slight improvement in fairness concerning the scenario where the stations are grouped randomly (Table 5.5), with $JF = 0.6256$, which means that the more distant stations have better channel access conditions when the stations are grouped according to their distance from the AP.

Now, let us consider fixed packet size of 1024 bytes under MCS0 mode. Table 5.10 shows the results. As observed in Section 5.2.1, there is a decrease in throughput if compared to the case with packet size of 256 bytes. Despite obtaining a slight improvement of fairness, $JF = 0.4096$, in comparison with the case of random groups, in this case, it is easier to observe how stations located farther away from the AP have more difficulty accessing the channel. We find that, in the latest case, a successful transmission lasts approximately 13.8 ms, without considering the backoff time, and the RAW slot of 24.98 ms is sufficient for only one complete transmission. Hence, the groups further away from the AP need more time to transmit their packets, since they spend more time in backoff and re-transmission attempts. Moreover, larger packets have a higher probability of bit errors. We also notice that, in simulations, 5.3% of stations (on average) did not transmit any packet. This explains why group 4 has the largest discrepancy between the analytical model and simulations, and the lowest Jain's Fairness.

For the next scenarios, we use MCS3, which has a higher data rate than MCS0. We use the same data frame lengths, 256 bytes and 1024 bytes. Table 5.11 shows the results for 256 bytes, while Table 5.12 shows the results for 1024 bytes.

As expected, the average throughput also increases compared to the scenarios with MCS0

Table 5.10. Comparison between analytical model and computer simulations of the throughput in Mb/s for MCS0 and 1024 bytes with RAW slot of the same duration in each RAW group.

Group	Simulations	Sta dev	Analytical Model	Error %
1	0.3146	0.0128	0.3236	2.78
2	0.1982	0.0744	0.2054	3.51
3	0.1856	0.0262	0.1978	6.16
4	0.0533	0.0328	0.0843	36.77
Jain's Fairness	0.4096	0.0628		

Table 5.11. Comparison between analytical model and computer simulations of the throughput in Mb/s for MCS3 and 256 bytes with RAW slot of the same duration in each RAW group.

Group	Simulations	Sta dev	Analytical Model	Error %
1	0.6133	0.0544	0.6291	2.51
2	0.5338	0.0483	0.5213	2.34
3	0.5093	0.0408	0.5133	0.78
4	0.5022	0.0189	0.5056	0.67
Jain's Fairness	0.6256	0.0418		

mode due to the increase in data rate. Additionally, there is a significant improvement of fairness in the scenario with the data frames of 1024 bytes. The analytical model shows a smaller discrepancy compared to simulations, with the highest percentage being 8.18%. For the scenario with a data frame length of 256 bytes, $JF = 0.6256$, and for the scenario with data frame length of 1024 bytes, $JF = 0.6246$. We can observe a slight improvement in fairness when comparing with the values in Tables 5.7 and 5.8, where the stations are grouped randomly.

5.2.3 Case 3

In this case study, we assign different RAW slot duration to each RAW group according to the zone in which the stations are located to improve fairness and throughput for those further away from the AP. We choose the RAW slot duration of each group according to Eqs. (2.2) and (4.30). First, we use Eq. (4.30) to define the minimum duration of the RAW slot required

Table 5.12. Comparison between analytical model and computer simulations of the throughput in Mb/s for MCS3 and 1024 bytes with RAW slot of the same duration in each RAW group.

Group	Simulations	Sta dev	Analytical Model	Error %
1	1.0773	0.0539	1.1734	8.18
2	0.9741	0.1069	0.9406	3.43
3	0.8812	0.0981	0.8867	0.62
4	0.8640	0.0309	0.8549	1.05
Jain's Fairness	0.6246	0.0421		

to obtain at least one successful transmission. Then, we use Eq. (2.2) to find the value closest to this minimum value to define the duration of the RAW slot according to the standard. Hence, maintaining the same beacon interval duration, we choose the RAW slots durations of 19.7 ms, 23.3 ms, 25.7 ms, and 29.3 ms for groups 1, 2, 3 and 4, respectively. Table 5.13 presents the new results for the scenario for MCS0 and a data frame of 256 bytes. Comparing with Table 5.9, we observe an increase in throughput in the furthest groups (groups 3 and 4). Despite slightly reducing the throughput of the closest groups, we obtained a significant gain in fairness, $JF = 0.8653$ instead of $JF = 0.6256$ IN Case 2.

Table 5.13. Comparison between analytical model and computer simulations of the throughput in Mb/s for MCS0 and 256 bytes with different Raw slot durations.

Group	**Simulations**	**Sta dev**	**Analytical Model**	**Error %**
1	0.3066	0.0087	0.3091	0.81
2	0.3040	0.0536	0.3001	1.28
3	0.3010	0.0138	0.2923	2.89
4	0.2806	0.0041	0.2784	0.78
Jain's Fairness	0.8653	0.0173		

Table 5.14 reports the results for MCS0 and packets with 1024 bytes. During simulations, we observed that all stations transmitted at least one packet when a longer-duration RAW slot was allocated to this group, unlike in Case 1 and Case 2, where some stations could not transmit their packets. Despite the throughput of the furthest group remaining low due to the low data rate and higher error probability, we obtained a significant improvement in network fairness, with a Jain's Fairness of $JF = 0.7655$ instead of $JF = 0.4096$ in Case 1, and higher accuracy of the analytical model.

Table 5.14. Comparison between analytical model and computer simulations of the throughput in Mb/s for MCS0 and 1024 bytes with different RAW slot durations.

Group	**Simulations**	**Sta dev**	**Analytical Model**	**Error %**
1	0.3091	0.0128	0.3047	1.42
2	0.2795	0.0744	0.2713	2.93
3	0.2621	0.0262	0.2498	4.69
4	0.1582	0.0328	0.1564	1.13
Jain's Fairness	0.7655	0.0403		

Tables 5.15 and 5.16 contain the results for MCS3 data frames of 256 bytes and 1024 bytes, respectively. In both scenarios, we achieve a significant improvement in network fairness, and the throughput of the furthest group is more similar to the throughput of the closest

group when compared to Case 2. Therefore, assigning larger RAW slots to the more distant groups can compensate for the higher bit error probabilities and provide more similar channel access conditions between the groups. The analytical model showed similar behavior to the simulations, with percentage errors lower than 6.32%, and the Jain's fairness are $JF = 0.8541$ and $JF = 0.8427$ for the cases with data frame lengths of 256 and 1024, respectively.

Table 5.15. Comparison between analytical model and computer simulations of the throughput in Mb/s for MCS3 and 256 bytes with different Raw slot durations.

Group	Simulations	Sta dev	Analytical Model	Error %
1	0.6084	0.0192	0.6122	0.62
2	0.5272	0.0130	0.5157	2.18
3	0.5709	0.0317	0.5928	3.69
4	0.5452	0.0112	0.5496	0.80
Jain's Fairness	0.8541	0.0152		

Table 5.16. Comparison between analytical model and computer simulations of the throughput in Mb/s for MCS3 and 1024 bytes with different Raw slot durations.

Group	Simulations	Sta dev	Analytical Model	Error %
1	1.0608	0.0539	1.1401	6.32
2	0.9674	0.1069	0.9298	3.89
3	1.0942	0.0981	1.0519	3.86
4	0.9917	0.0309	0.9833	0.85
Jain's Fairness	0.8427	0.0261		

5.2.4 Comparison between the three case studies

In this section, we show bar graphs comparing network performance across the three case studies for each scenario. In Figure 5.8 is possible to verify that, although the random grouping (Case 1) seems fairer, as the throughput in each group has similar values, the fairness is the lowest among the three case studies. This occurs because stations further away from the AP compete for the channel in the same group as stations closer to the AP. This way, the closer stations harm the more distant stations. In Case 2, where stations are grouped according to the distance from the rings in which they are located, but with RAW slots of the same duration, we can see a slight increase in Jain fairness, which means that the stations further away were able to access the channel better when competing with stations with similar channel conditions. Finally, in Case 3, we observe the importance of allocating channel resources heterogeneously to

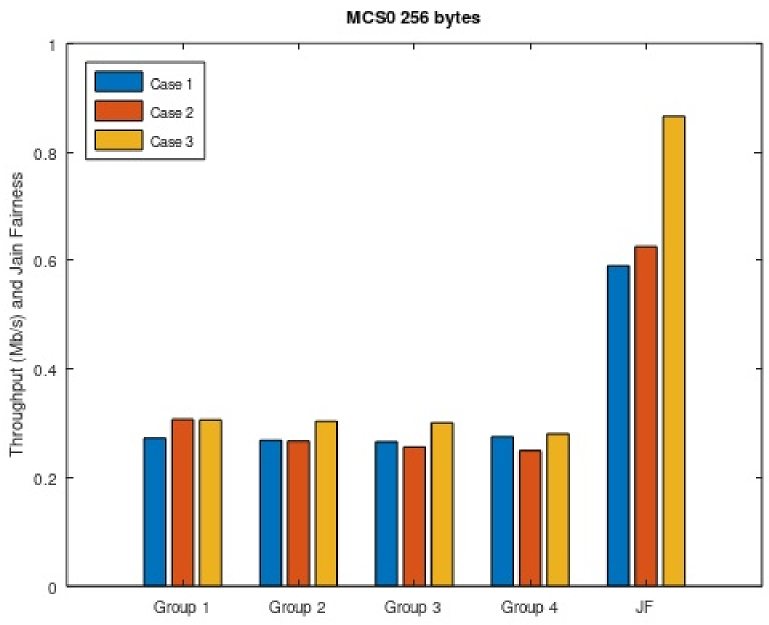

Figure 5.8. Comparison of the three case studies considering the throughput per group and Jain's fairness obtained in computational simulations for the MCS0 and packet size of 256 bytes scenario.

stations with different channel conditions. We see an increase in the throughput of the groups further away from the AP and a significant improvement in Jain's fairness.

We can observe a similar behavior in Figure 5.9. Case 1 seems to be fairer because the average throughput of each group has approximate values. However, this case had the most minor justice for this scenario. In Case 2, we have a slight improvement in fairness. At the same time, we have yet to obtain a significant increase in fairness because more distant stations need more channel time to transmit their packets, as they take longer in the backoff process and with retransmission attempts. In addition, this scenario has longer packet lengths and a low transmission rate, and we observe that some stations did not send their packets. Therefore, the duration of the RAW slot needs to be increased to guarantee that stations in the most distant group send their packets. In Case 3, we observe the biggest improvement in fairness when the RAW slot duration is distributed heterogeneously for each group.

We compare the outcomes for the scenario configured with MCS3 mode and packet lengths of 256 bytes and 1024 bytes in Figures 5.10 and 5.11, respectively. In both scenarios, Case 1 appears to be more equitable distributed as the average throughput of each group is approximately similar. In Case 2, we have a small improvement in fairness, but there is still no significant increase since more distant stations require more channel time to transmit their

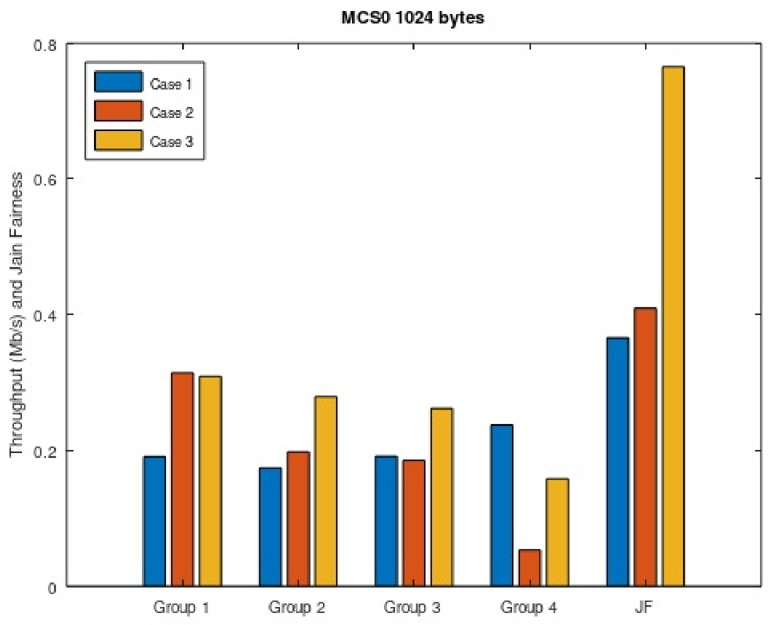

Figure 5.9. Comparison of the three case studies considering the throughput per group and Jain's fairness obtained in computational simulations for the MCS0 and packet size of 1024 bytes scenario.

packets. Finally, we also observe a significant improvement in network fairness in Case 3, when assigned different RAW slot duration for each group. Furthermore, the average throughput of the furthest group is more similar to the average throughput of the closest group when compared to Cases 1 and 2. Therefore, these results show that assigning larger RAW slots duration to the more distant groups could compensate for higher bit error probabilities and offer more equitable channel access conditions between groups.

5.3 CONCLUSIONS

In this Chapter, we presented the numerical results obtained with our analytical model and computer simulations. First, we validated the importance of the RAW slot completion probability to the performance of our analytical model with comparisons with computer simulations by considering two cases for our model: when such probability is taken into account, and when it is removed from the model. Additionally, we compared the average aggregate throughput predicted by our analytical model with results provided by two analytical models available in literature. The results showed that our analytical model predicts network performance with higher accuracy than the other models. Then, we extended the analysis by including the impact of Rayleigh fading channel and large-scale path loss, and the results showed that the proposed

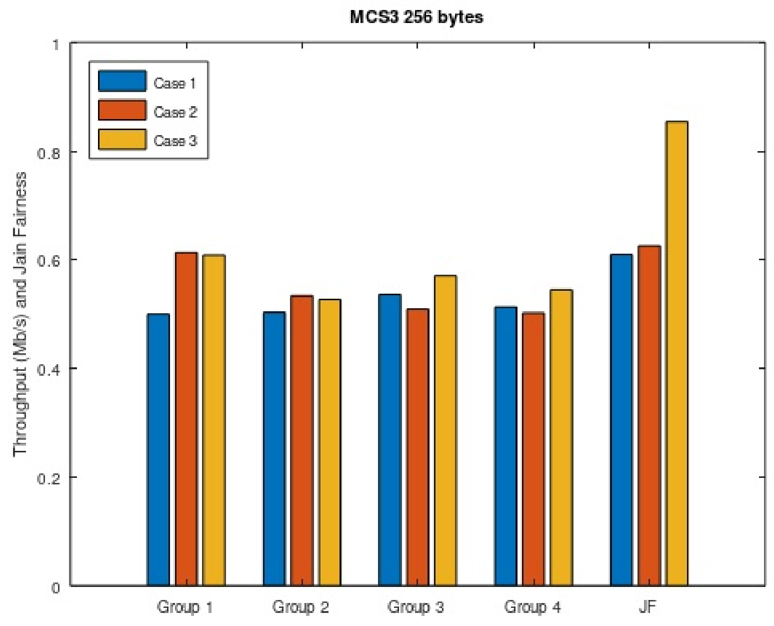

Figure 5.10. Comparison of the three case studies considering the throughput per group and Jain's fairness obtained in computational simulations for the MCS3 and packet size of 256 bytes scenario.

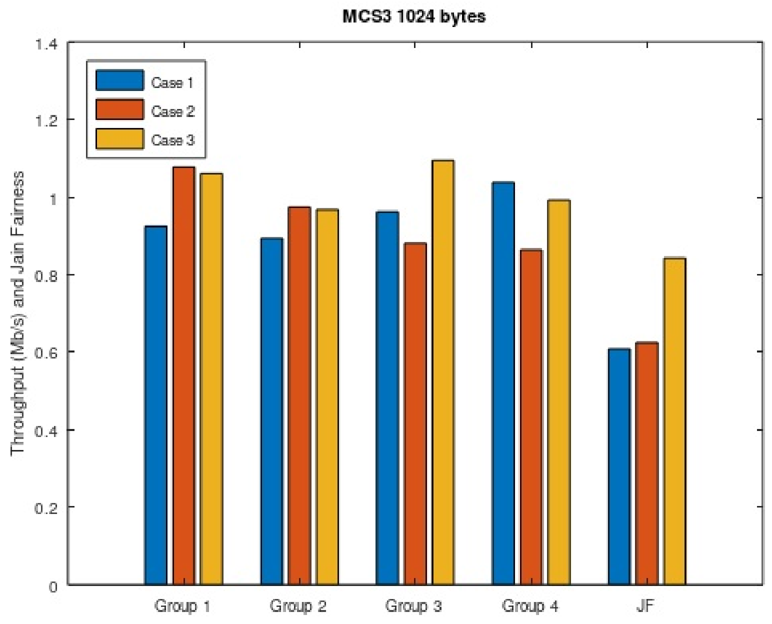

Figure 5.11. Comparison of the three case studies considering the throughput per group and Jain's fairness obtained in computational simulations for the MCS3 and packet size of 1024 bytes scenario.

analytical model predicts network performance with high accuracy in most scenarios. In the scenario in which we obtained a higher percentage error (36.77%) between the analytical model and simulations, we verified that there were packet losses in the simulations that the analytical model did not predict. Finally, we showed the need to adjust the RAW slot duration according to the distance of the stations to the AP to achieve throughput fairness among groups.

CONCLUSION

Currently, there is a growing use of objects inserted in the context of the Internet of Things, whose main challenges are networks with many connected devices, with limited power and memory resources, and geographically distributed over large areas. To deal with these challenges, a new Wi-Fi standard was developed, the IEEE 802.11ah, which supports long-range transmissions and many connected devices to a single access point. One of its main features is the restricted access window, RAW. With the RAW mechanism, the stations in the network can be divided into smaller groups, which can only try to access the channel within the interval time determined for the group to which they belong. In this work, we introduced a discrete-time Markov chain model to evaluate the average aggregate throughput performance of an IEEE 802.11ah network under ideal and non-ideal channel conditions and saturated traffic. The model addresses the dynamics of a station within its assigned RAW slot and includes the impact of RAW slot time completion as a key parameter, for which we present a heuristic approach for is derivation and showed its impact on model accuracy through comparisons with computer simulations.

For this, initially, we presented an analytical model consisting of a discrete-time Markov chain with two dimensions to describe the behavior of a station in its backoff process within a RAW slot. Each Markov chain state represents the backoff counter, while the lines represent the retransmission stages. The transition probabilities represent the events of backoff counter decrement, backoff counter freeze when the channel is busy, and RAW slot time completion. An empirical expression was provided for the probability of RAW slot time completion which takes into account the length of the RAW slot with respect to the beacon interval, the transmission attempts, and the number of stations within a RAW slot. We evaluated the importance such probability by comparing the numerical results of the model including the RAW slot completion probability and the model without including this probability. We verified that the model that considers the expression of RAW slot time completion was the closest to the computer simulations. Moreover, the performance of the proposed analytical model was compared to two other analytical models available in the literature, whose numerical results were also compared to computer simulations carried out independently in the ns-3 simulator. The results showed that the proposed analytical model predicts network performance with higher accuracy than the other models (expressed in terms of RMSE).

Furthermore, we extended the analytical model by including the impact of Rayleigh fading channel and large-scale path loss. The performance of the proposed analytical model after including these Physical layer parameters was compared to computer simulations carried out independently in the ns-3 simulator, and the results showed that the proposed analytical model predicts network performance with high accuracy in most scenarios. Additionally, we showed the need to adjust the RAW slot duration according to the distance of the stations to the AP to achieve throughput fairness among groups. We propose an expression to adjust the duration of the RAW slot according to the distance of the stations to ensure that the more distant stations, which tend to spend more time in the backoff process and with retransmission attempts due to poor channel conditions, have time enough to transmit their packets. The results showed a significant increase in fairness between groups when we allocated different RAW slot durations according to group distance.

The development of analytical models that describe communication protocols is important to evaluate the performance metrics of a network. From the analytical model, it is possible to evaluate how the network behaves with different initial conditions and with variations of its parameters, which allows the implementation of more efficient communication protocols with lower energy consumption. Thus, for future work, we plan to develop expressions to compute delay and energy consumption, and to extend this model to the scenario of non-saturated networks. Furthermore, we plan to apply the model in efficient station allocation in RAW groups and RAW slots studies. Additionally, we intend to explore heterogeneous scenarios with greater mobility of stations and extend our analytical model for these scenarios to formulate an optimization problem to derive a station grouping strategy.

REFERENCES

ADAME, T.; BEL, A.; BELLALTA, B.; BARCELO, J.; OLIVER, M. Ieee 802.11 ah: the wifi approach for m2m communications. *IEEE Wireless Communications*, IEEE, v. 21, n. 6, p. 144–152, 2014. Cited in page 9.

AHMED, N.; DE, D.; BARBHUIYA, F. A.; HUSSAIN, M. I. MAC protocols for IEEE 802.11ah-based internet of things: A survey. *IEEE Internet of Things Journal*, v. 9, n. 2, p. 916–938, 2022. Cited in page 1.

AHMED, N.; RAHMAN, H.; HUSSAIN, M. I. A comparison of 802.11 ah and 802.15. 4 for iot. *ICT Express*, Elsevier, v. 2, n. 3, p. 100–102, 2016. Cited in page 9.

AL-FUQAHA, A.; GUIZANI, M.; MOHAMMADI, M.; ALEDHARI, M.; AYYASH, M. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, v. 17, n. 4, p. 2347–2376, 2015. Cited in page 1.

ALI, M. Z.; MIŠIĆ, J.; MIŠIĆ, V. B. Performance Evaluation of Heterogeneous IoT Nodes with Differentiated QoS in IEEE 802.11ah RAW Mechanism. *IEEE Transactions on Vehicular Technology*, IEEE, v. 68, n. 4, p. 3905–3918, 2019. Cited 2 times in pages 20 and 26.

BADARLA, S. P.; HARIGOVINDAN, V. Restricted access window-based resource allocation scheme for performance enhancement of IEEE 802.11ah multi-rate IoT networks. *IEEE Access*, IEEE, v. 9, p. 136507–136519, 2021. Cited 4 times in pages 4, 22, 26, and 27.

BANKOV, D.; KHOROV, E.; LYAKHOV, A.; FAMAEY, J. Resource allocation for Machine-Type communication of Energy-Harvesting devices in Wi-Fi halow networks. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 20, n. 9, p. 2449, 2020. Cited 2 times in pages 3 and 4.

BELLINI, P.; NESI, P.; PANTALEO, G. IoT-enabled smart cities: A review of concepts, frameworks and key technologies. *Applied Sciences*, v. 12, n. 3, p. 1607, 2022. Cited in page 1.

BIANCHI, G. Performance Analysis of the IEEE 802.11 Distributed Coordination Function. *IEEE Journal on Selected Areas in Communications*, IEEE, v. 18, n. 3, p. 535–547, 2000. Cited 5 times in pages 19, 20, 21, 27, and 36.

CHANG, T.-C.; LIN, C.-H.; LIN, K. C.-J.; CHEN, W.-T. Load-balanced Sensor Grouping for IEEE 802.11ah Networks. In: IEEE. *IEEE Global Communications Conference*. [S.l.], 2015. p. 1–6. Cited 3 times in pages 21, 26, and 27.

DANESHGARAN, F.; LADDOMADA, M.; MESITI, F.; MONDIN, M.; ZANOLO, M. Saturation throughput analysis of IEEE 802.11 in the presence of non ideal transmission channel and capture effects. *IEEE transactions on Communications*, IEEE, v. 56, n. 7, p. 1178–1188, 2008. Cited in page 32.

DIN MUHAMMAD, H. A. Qutab-ud; BADIHI, B.; LARMO, A.; TORSNER, J.; VALKAMA, M. Performance Analysis of IoT-Enabling IEEE 802.11ah Technology and its RAW Mechanism with Non-cross Slot Boundary Holding Schemes. In: *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. [S.l.: s.n.], 2015. p. 1–6. Cited 4 times in pages 18, 26, 32, and 33.

DONG, M.; WU, Z.; GAO, X.; ZHAO, H. An Efficient Spatial Group Restricted Access Window Scheme for IEEE 802.11ah Networks. In: IEEE. *Sixth International Conference on Information Science and Technology*. [S.l.], 2016. p. 168–173. Cited 5 times in pages 3, 4, 24, 26, and 27.

GOLDSMITH, A. *Wireless communications*. [S.l.]: Cambridge university press, 2005. Cited in page 33.

HEUSSE, M.; ROUSSEAU, F.; BERGER-SABBATEL, G.; DUDA, A. Performance anomaly of 802.11b. In: *IEEE INFOCOM*. [S.l.: s.n.], 2003. p. 836–843. Cited in page 4.

HUANG, C.-M.; HUANG, S.-H. Traffic-aware re-grouping for load balance in ieee 802.11 ah iot network based on the registered backoff time mechanism. *The Computer Journal*, Oxford University Press, p. bxad027, 2023. Cited 3 times in pages 24, 26, and 27.

IEEE STD 802.11AH-2016. *IEEE Standard for Information technology–Telecommunications and information exchange between systems – Local and metropolitan area networks–Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: Sub 1 GHz License Exempt Operation*. [S.l.], 2016. Cited 6 times in pages iii, 2, 13, 15, 17, and 41.

KAI, C.; ZHANG, J.; ZHANG, X.; HUANG, W. Energy-Efficient Sensor Grouping for IEEE 802.11ah Networks with Max-Min Fairness Guarantees. *IEEE Access*, IEEE, v. 7, p. 102284–102294, 2019. Cited 5 times in pages 21, 22, 23, 26, and 27.

KHOROV, E.; KROTOV, A.; LYAKHOV, A. Modelling Machine Type Communication in IEEE 802.11ah Networks. In: IEEE. *IEEE International Conference on Communication Workshop*. [S.l.], 2015. p. 1149–1154. Cited 2 times in pages 20 and 26.

KHOROV, E.; KROTOV, A.; LYAKHOV, A.; YUSUPOV, R.; CONDOLUCI, M.; DOHLER, M.; AKYILDIZ, I. Enabling the Internet of Things With Wi-Fi Halow —Performance Evaluation of the Restricted Access Window. *IEEE Access*, IEEE, v. 7, p. 127402–127415, 2019. Cited 4 times in pages 3, 4, 21, and 26.

KHOROV, E.; LYAKHOV, A.; KROTOV, A.; GUSCHIN, A. A survey on ieee 802.11 ah: An enabling networking technology for smart cities. *Computer Communications*, Elsevier, v. 58, p. 53–69, 2015. Cited in page 9.

LACAGE, M.; HENDERSON, T. R. Yet another network simulator. In: *Proceedings of the 2006 Workshop on ns-3*. [S.l.: s.n.], 2006. p. 12–es. Cited in page 34.

LAKSHMI, L. R.; SIKDAR, B. Fair scheduling in ieee 802.11 ah networks for internet of things applications. In: IEEE. *IEEE GLOBECOM*. [S.l.], 2019. p. 1–7. Cited 3 times in pages 24, 26, and 27.

MAHESH, M.; HARIGOVINDAN, V. Restricted Access Window-Based Novel Service Differentiation Scheme for Group-Synchronized DCF. *IEEE Communications Letters*, IEEE, v. 23, n. 5, p. 900–903, 2019. Cited 2 times in pages 18 and 26.

MAHESH, M.; PAVAN, B. S.; HARIGOVINDAN, V. Data rate-based grouping to resolve performance anomaly of multi-rate IEEE 802.11 ah IoT networks. *IEEE Networking Letters*, IEEE, v. 2, n. 4, p. 166–170, 2020. Cited 3 times in pages 4, 22, and 27.

MALOOK, F.; MUJAHID, O.; ULLAH, Z.; FOUZDER, T. On enhancing the performance of ieee 802.11 ah by employing a dynamic raw approach in iot networks. *Wireless Personal Communications*, Springer, v. 129, n. 3, p. 1983–1997, 2023. Cited 2 times in pages 24 and 26.

MINOLI, D.; SOHRABY, K.; OCCHIOGROSSO, B. IoT considerations, requirements, and architectures for smart buildings—energy optimization and next-generation building management systems. *IEEE Internet of Things Journal*, v. 4, n. 1, p. 269–283, 2017. Cited in page 1.

MISRA, N. N.; DIXIT, Y.; AL-MALLAHI, A.; BHULLAR, M. S.; UPADHYAY, R.; MARTYNENKO, A. IoT, big data, and artificial intelligence in agriculture and food industry. *IEEE Internet of Things Journal*, v. 9, n. 9, p. 6305–6324, 2022. Cited in page 1.

MOSAVAT-JAHROMI, H.; LI, Y.; CAI, L. A throughput fairness-based grouping strategy for dense IEEE 802.11ah networks. In: IEEE. *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. [S.l.], 2019. p. 1–6. Cited 3 times in pages 23, 26, and 27.

NAWAZ, N.; HAFEEZ, M.; ZAIDI, S. A. R.; MCLERNON, D. C.; GHOGHO, M. Throughput Enhancement of Restricted Access Window for Uniform Grouping Scheme in IEEE 802.11ah. In: IEEE. *IEEE International Conference on Communications*. [S.l.], 2017. p. 1–7. Cited 2 times in pages 21 and 26.

OLIVEIRA, E. C.; SOARES, S. M.; CARVALHO, M. M. K-means Based Grouping of Stations with Dynamic AID Assignment in IEEE802.11ah Networks. In: IEEE. *18th International Conference on Mobility, Sensing and Networking (MSN)*. [S.l.], 2022. p. 1–8. Cited 4 times in pages 4, 7, 26, and 27.

PARK, C. W.; HWANG, D.; LEE, T.-J. Enhancement of IEEE 802.11ah MAC for M2M Communications. *IEEE Communications Letters*, IEEE, v. 18, n. 7, p. 1151–1154, 2014. Cited 2 times in pages 19 and 26.

QUY, V. K.; HAU, N. V.; ANH, D. V.; QUY, N. M.; BAN, N. T.; LANZA, S.; RANDAZZO, G.; MUZIRAFUTI, A. IoT-enabled smart agriculture: Architecture, applications, and challenges. *Applied Sciences*, v. 12, n. 7, p. 3396, 2022. Cited in page 1.

RAEESI, O.; PIRSKANEN, J.; HAZMI, A.; TALVITIE, J.; VALKAMA, M. Performance Enhancement and Evaluation of IEEE 802.11ah Multi-access Point Network Using Restricted Access Window Mechanism. In: IEEE. *IEEE International Conference on Distributed Computing in Sensor Systems*. [S.l.], 2014. p. 287–293. Cited in page 19.

RAEESI, O.; PIRSKANEN, J.; HAZMI, A.; LEVANEN, T.; VALKAMA, M. Performance Evaluation of IEEE 802.11 ah and its Restricted Access Window Mechanism. In: IEEE. *IEEE International Conference on Communications Workshops*. [S.l.], 2014. p. 460–466. Cited 2 times in pages 19 and 26.

SANGEETHA, U.; BABU, A. Performance Analysis of IEEE 802.11ah Wireless Local Area Network Under the Restricted Access Window-Based Mechanism. *International Journal of Communication Systems*, WILEY 111 RIVER ST, HOBOKEN 07030-5774, NJ USA, v. 32, n. 4, 2019. Cited 5 times in pages 3, 4, 20, 26, and 44.

SANGEETHA, U.; BABU, A. Fair and efficient resource allocation in IEEE 802.11ah WLAN with heterogeneous data rates. *Computer Communications*, Elsevier, v. 151, p. 154–164, 2020. Cited 5 times in pages 4, 21, 22, 26, and 27.

SANGEETHA, U.; BABU, A. Service Differentiation in IEEE 802.11 ah WLAN under Restricted Access Window based MAC protocol. *Computer Communications*, Elsevier, v. 172, p. 142–154, 2021. Cited 2 times in pages 20 and 26.

SISINNI, E.; SAIFULLAH, A.; HAN, S.; JENNEHAG, U.; GIDLUND, M. Industrial internet of things: Challenges, opportunities, and directions. *IEEE transactions on industrial informatics*, v. 14, n. 11, p. 4724–4734, 2018. Cited in page 1.

SOARES, S. M.; CARVALHO, M. M. Throughput Analytical Modeling of IEEE 802.11ah Wireless Networks. In: IEEE. *16th IEEE Annual Consumer Communications & Networking Conference*. [S.l.], 2019. p. 1–4. Cited 4 times in pages 4, 6, 20, and 32.

SOARES, S. M.; CARVALHO, M. M. An analytical model for the aggregate throughput of ieee 802.11 ah networks under the restricted access window mechanism. *Sensors*, MDPI, v. 22, n. 15, p. 5561, 2022. Cited 4 times in pages 5, 7, 25, and 26.

TARAMIT, H.; CAMACHO-ESCOTO, J. J.; GOMEZ, J.; OROZCO-BARBOSA, L.; HAQIQ, A. Accurate analytical model and evaluation of wi-fi halow based iot networks under a rayleigh-fading channel with capture. *Mathematics*, MDPI, v. 10, n. 6, p. 952, 2022. Cited 3 times in pages 3, 19, and 26.

TARAMIT, H.; OROZCO-BARBOSA, L.; HAQIQ, A. Resource and energy-efficient configuration of ieee 802.11 ah networks under rayleigh channels. In: IEEE. *2022 4th IEEE Middle East and North Africa COMMunications Conference (MENACOMM)*. [S.l.], 2022. p. 177–184. Cited 2 times in pages 23 and 26.

TARAMIT, H.; OROZCO-BARBOSA, L.; HAQIQ, A.; ESCOTO, J. J. C.; GOMEZ, J. Load-aware channel allocation for ieee 802.11 ah-based networks. *IEEE Access*, IEEE, v. 11, p. 24484–24496, 2023. Cited 2 times in pages 23 and 26.

TIAN, L.; DERONNE, S.; LATRÉ, S.; FAMAEY, J. Implementation and Validation of an IEEE 802.11ah Module for ns-3. In: ACM. *8th Workshop on ns-3*. [S.l.], 2016. p. 49–56. Cited 6 times in pages 5, 6, 7, 25, 40, and 41.

TIAN, L.; KHOROV, E.; LATRÉ, S.; FAMAEY, J. Real-time Station Grouping Under Dynamic Traffic for IEEE 802.11ah. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 17, n. 7, p. 1559, 2017. Cited 3 times in pages 23, 26, and 27.

TIAN, L.; MEHARI, M.; SANTI, S.; LATRÉ, S.; POORTER, E. D.; FAMAEY, J. IEEE 802.11ah Restricted Access Window Surrogate Model for Real-time Station Grouping. In: IEEE. *IEEE 19th International Symposium on"A World of Wireless, Mobile and Multimedia Networks"*. [S.l.], 2018. p. 14–22. Cited 2 times in pages 23 and 26.

TIAN, L.; SANTI, S.; SEFERAGIĆ, A.; LAN, J.; FAMAEY, J. Wi-Fi HaLow for the Internet of Things: An up-to-date survey on IEEE 802.11ah research. *Journal of Network and Computer Applications*, Elsevier, v. 182, p. 103036, 2021. Cited 2 times in pages 1 and 16.

TIAN, L.; ŠLJIVO, A.; SANTI, S.; POORTER, E. D.; HOEBEKE, J.; FAMAEY, J. Extension of the IEEE 802.11ah ns-3 Simulation Module. In: ACM. *10th Workshop on ns-3*. [S.l.], 2018. p. 53–60. Cited in page 7.

YAQOOB, I.; AHMED, E.; HASHEM, I. A. T.; AHMED, A. I. A.; GANI, A.; IMRAN, M.; GUIZANI, M. Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges. *IEEE Wireless Communications*, v. 24, n. 3, p. 10–16, 2017. Cited in page 1.

YOON, S.-G.; SEO, J.-O.; BAHK, S. Regrouping Algorithm to Alleviate the Hidden Node Problem in 802.11ah Networks. *Computer Networks*, Elsevier, v. 105, p. 22–32, 2016. Cited 3 times in pages 24, 26, and 27.

ZHENG, L.; NI, M.; CAI, L.; PAN, J.; GHOSH, C.; DOPPLER, K. Performance Analysis of Group-Synchronized DCF for Dense IEEE 802.11 Networks. *IEEE Transactions on Wireless Communications*, IEEE, v. 13, n. 11, p. 6180–6192, 2014. Cited 7 times in pages 3, 4, 18, 19, 26, 27, and 44.

APÊNDICE A

NS-3 PROGRAM

s1g-tcc-raw.cc

```
 #include "s1g-tcc-raw.h"

NS_LOG_COMPONENT_DEFINE("S1gTccRaw");

uint32_t AssocNum = 0;
int64_t AssocTime = 0;
uint32_t StaNum = 0;
NetDeviceContainer staDeviceCont;
const int MaxSta = 8000;

Configuration config;
Statistics stats;
SimulationEventManager eventManager;



class assoc_record {
public:
      assoc_record();
      bool GetAssoc();
      void SetAssoc(std::string context, Mac48Address address);
      void UnsetAssoc(std::string context, Mac48Address address);
      void setstaid(uint16_t id);
private:
      bool assoc;
      uint16_t staid;
};

assoc_record::assoc_record() {
      assoc = false;
      staid = 65535;
}

void assoc_record::setstaid(uint16_t id) {
      staid = id;
}

void assoc_record::SetAssoc(std::string context, Mac48Address address) {
      assoc = true;
}

void assoc_record::UnsetAssoc(std::string context, Mac48Address address) {
      assoc = false;
}
```

```
bool assoc_record::GetAssoc() {
      return assoc;
}

typedef std::vector<assoc_record *> assoc_recordVector;
assoc_recordVector assoc_vector;

uint32_t GetAssocNum() {
      AssocNum = 0;
      for (assoc_recordVector::const_iterator index = assoc_vector.begin();
                  index != assoc_vector.end(); index++) {
            if ((*index)->GetAssoc()) {
                  AssocNum++;
            }
      }
      return AssocNum;
}

void PopulateArpCache() {
      Ptr<ArpCache> arp = CreateObject<ArpCache>();
      arp->SetAliveTimeout(Seconds(3600 * 24 * 365));
      for (NodeList::Iterator i = NodeList::Begin(); i != NodeList::End(); ++i) {
            Ptr<Ipv4L3Protocol> ip = (*i)->GetObject<Ipv4L3Protocol>();
            NS_ASSERT(ip != 0);
            ObjectVectorValue interfaces;
            ip->GetAttribute("InterfaceList", interfaces);
            for (ObjectVectorValue::Iterator j = interfaces.Begin();
                        j != interfaces.End(); j++) {
                  Ptr<Ipv4Interface> ipIface =
                              (j->second)->GetObject<Ipv4Interface>();
                  NS_ASSERT(ipIface != 0);
                  Ptr<NetDevice> device = ipIface->GetDevice();
                  NS_ASSERT(device != 0);
                  Mac48Address addr = Mac48Address::ConvertFrom(device->GetAddress());
                  for (uint32_t k = 0; k < ipIface->GetNAddresses(); k++) {
                        Ipv4Address ipAddr = ipIface->GetAddress(k).GetLocal();
                        if (ipAddr == Ipv4Address::GetLoopback())
                              continue;
                        ArpCache::Entry * entry = arp->Add(ipAddr);
                        entry->MarkWaitReply(0);
                        entry->MarkAlive(addr);
                        std::cout << "Arp Cache: Adding the pair (" << addr << ","
                                    << ipAddr << ")" << std::endl;
                  }
            }
      }
      for (NodeList::Iterator i = NodeList::Begin(); i != NodeList::End(); ++i) {
            Ptr<Ipv4L3Protocol> ip = (*i)->GetObject<Ipv4L3Protocol>();
            NS_ASSERT(ip != 0);
            ObjectVectorValue interfaces;
            ip->GetAttribute("InterfaceList", interfaces);
            for (ObjectVectorValue::Iterator j = interfaces.Begin();
                        j != interfaces.End(); j++) {
                  Ptr<Ipv4Interface> ipIface =
                              (j->second)->GetObject<Ipv4Interface>();
                  ipIface->SetAttribute("ArpCache", PointerValue(arp));
            }
      }
}
```

```
void OnSignalArrival(Ptr< const Packet > packet, uint16_t channelFreqMhz,
uint16_t channelNumber, uint32_t rate, bool isShortPreamble, WifiTxVector txvector,
double signalDbm, double noiseDbm){
      Ptr< Packet > packetCopy = packet->Copy();
      uint32_t packetSize = packetCopy->GetSize();

      WifiMacHeader hdr;
      packetCopy->RemoveHeader (hdr);

      Mac48Address destAddr = hdr.GetAddr1();
      Mac48Address sourceAddr = hdr.GetAddr2();
      const char * typeString = hdr.GetTypeString();

      if (hdr.IsData()){
            // LlcSnapHeader llchdr;
            Ipv4Header ipv4hdr;
            // packetCopy->RemoveHeader (llchdr);
            packetCopy->RemoveHeader (ipv4hdr);

            Ipv4Address sourceip = ipv4hdr.GetSource();
            Ipv4Address destinationip = ipv4hdr.GetDestination();
            uint8_t tos = ipv4hdr.GetTos();

            cout<<"IPv4 source: "<<sourceip<<" ,";
            cout<<"IPv4 destination: "<<destinationip<<" ,";
            cout<<"Type of Service: "<<unsigned(tos);
      }

      cout<<endl;
      }


vector<string> dataModes = {"MCS2_3", "MCS2_2", "MCS2_1", "MCS2_0"};
string GetRandomDataMode ( uint16_t numModes )
{
Ptr<UniformRandomVariable> m_rv = CreateObject<UniformRandomVariable>();

uint16_t modeNum = m_rv->GetInteger(0, numModes-1);

return dataModes[modeNum];
}


string GetRoundRoubindDataMode ( uint16_t idx, uint16_t numModes )
{
return dataModes[idx%numModes];
}

void ConfigureTopology (string TopologyFile)
{
uint16_t numZones = 0;
uint16_t minVal = 0, maxVal = 0;
string dataMode;

ifstream topoFile (TopologyFile);

if (topoFile.is_open())
{
```

```
topoFile >> numZones;
for (uint16_t i = 0; i < numZones; i++)
{
Zone * zone = new Zone;
topoFile >> maxVal;
zone->SetNumSta (maxVal);

topoFile >> minVal;
topoFile >> maxVal;
zone->SetRadius(minVal, maxVal);

topoFile >> dataMode;
zone->SetDataMode(dataMode);

topoFile >> minVal;
topoFile >> maxVal;
zone->SetPayloadSize(minVal, maxVal);

topoFile >> minVal;
topoFile >> maxVal;
zone->SetTrafficInterval(minVal, maxVal);

topology.AddZone(zone);
}
}
else
{
throw::invalid_argument("Invalid topology file.");
}
}

RPSVector configureRAW(RPSVector rpslist, string RAWConfigFile)
{
      uint16_t NRPS = 0;
      uint16_t NRAWPERBEACON = 0;
      uint16_t Value = 0;
      uint32_t page = 0;
      uint32_t aid_start = 0;
      uint32_t aid_end = 0;
      uint32_t rawinfo = 0;

      uint16_t ngroup;
      uint16_t nslot;

      ifstream myfile(RAWConfigFile);
      //1. get info from config file

      uint16_t block = 0;

      //2. define RPS
      if (myfile.is_open()) {
            myfile >> NRPS;
            for (uint16_t kk = 0; kk < NRPS; kk++) // number of beacons covering all raw groups
            {
                  RPS *m_rps = new RPS;
                  myfile >> NRAWPERBEACON;
                  ngroup = NRAWPERBEACON;
                  std::cout<<"RPS "<<kk+1<<" number of groups: "<<ngroup<<std::endl;
                  for (uint16_t i = 0; i < NRAWPERBEACON; i++) // raw groups in one beacon
                  {
                        RPS::RawAssignment *m_raw = new RPS::RawAssignment;

                        myfile >> Value;
                        m_raw->SetRawControl(Value);      //support paged STA or not
                        myfile >> Value;
                        m_raw->SetSlotCrossBoundary(Value);
                        myfile >> Value;
                        m_raw->SetSlotFormat(Value);
                        myfile >> Value;
                        m_raw->SetSlotDurationCount(Value);
                        myfile >> Value;
                        nslot = Value;
                        m_raw->SetSlotNum(Value);
                        myfile >> page;
aid_start = block << 6;
if (aid_start == 0) aid_start = 1;
aid_end = block << 6 | 0x3f;
block++;
                        rawinfo = (aid_end << 13) | (aid_start << 2) | page;
m_raw->SetRawGroup(rawinfo);
                        m_rps->SetRawAssignment(*m_raw);
                        delete m_raw;
                  }
                  rpslist.rpsset.push_back(m_rps);
            }
            myfile.close();
      } else
            cout << "Unable to open RAW configuration file \n";

      return rpslist;
}

/*
pageslice element and TIM(DTIM) together accomplish page slicing.

Prior knowledge:
802.11ah support up to 8192 stations, they are constructed into: page, block,
 subblock, sta.
there are 13 bit represent the AID of stations.
 AID[11-12] represent page.
 AID[6-10] represent block.
 AID[3-5] represent subblock.
 AID[0-2] represent sta.

A TIM(DTIM) element only support one page
A Page slice element only support one page

 Concept of page slicing:
 Between two DTIM beacon, there are many TIM beacons, only allow a TIM beacon
include some blocks of one page is called page slice. One TIM beacon is called a
%page slice. Page slcie element specify number of page slice between two DTIM,
number of blocks in each page slice.
Page slice element only appears together with DTIM.

 Details:
 Page slice element also indicates AP has buffered data for which block,
 if a station is in that block, the station should first sleep,
 then wake up at corresponding
 page slice(TIM beacon) which includes that block.

 When station wake up at that block, it check whether AP has data for itself.
 If has, keep awake to receive packets and go to sleep in the next beacon.
 */

void configurePageSlice (void)
{
            config.pageS.SetPageindex (config.pageIndex);
            config.pageS.SetPagePeriod (config.pagePeriod);
            //2 TIM groups between DTIMs
            config.pageS.SetPageSliceLen (config.pageSliceLength);
            //each TIM group has 1 block (2 blocks in 2 TIM groups)
            config.pageS.SetPageSliceCount (config.pageSliceCount);
            config.pageS.SetBlockOffset (config.blockOffset);
            config.pageS.SetTIMOffset (config.timOffset);
            //std::cout << "pageIndex=" << (int)config.pageIndex <<
            ", pagePeriod=" << (int)config.pagePeriod << ", pageSliceLength=
            " << (int)config.pageSliceLength << ", pageSliceCount="
            << (int)config.pageSliceCount << ",
            blockOffset=" << (int)config.blockOffset
            << ", timOffset=" << (int)config.timOffset
            << std::endl;
            // page 0
            // 8 TIM(page slice) for one page
            // 4 block (each page)
            // 8 page slice
            // both offset are 0
}

void configureTIM (void)
{
            config.tim.SetPageIndex (config.pageIndex);
            if (config.pageSliceCount)
                  config.tim.SetDTIMPeriod (config.pageSliceCount); // not necessarily the same
            else
                  config.tim.SetDTIMPeriod (1);

            //std::cout << "DTIM period=" << (int)config.pagePeriod << std::endl;
}

void checkRawAndTimConfiguration (void)
{
std::cout << "Checking RAW and TIM configuration..." << std::endl;
bool configIsCorrect = true;
NS_ASSERT (config.rps.rpsset.size());
// Number of page slices in a single page has to equal number of different RPS elements because
// If #PS > #RPS, the same RPS will be used in more than 1 PS and that is wrong because
// each PS can accommodate different AIDs (same RPS means same stations in RAWs)
if(config.pageSliceCount)
{
//NS_ASSERT (config.pagePeriod == config.rps.rpsset.size());
}
for (uint32_t j = 0; j < config.rps.rpsset.size(); j++)
{
uint32_t totalRawTime = 0;
for (uint32_t i = 0; i < config.rps.rpsset[j]->GetNumberOfRawGroups(); i++)
{
totalRawTime += (120 * config.rps.rpsset[j]->GetRawAssigmentObj(i)
   .GetSlotDurationCount() + 500) * config.rps.rpsset[j]->GetRawAssigmentObj(i).GetSlotNum();
auto aidStart = config.rps.rpsset[j]->GetRawAssigmentObj(i).GetRawGroupAIDStart();
auto aidEnd = config.rps.rpsset[j]->GetRawAssigmentObj(i).GetRawGroupAIDEnd();
configIsCorrect = check (aidStart, j) && check (aidEnd, j);
// AIDs in each RPS must comply with TIM in the following way:
// TIM0: 1-63; TIM1: 64-127; TIM2: 128-191; ...; TIM32: 1983-2047
// If RPS that belongs to TIM0 includes other AIDs (other than range [1-63]) configuration
   is incorrect
NS_ASSERT (configIsCorrect);
}
std::cout<<"totalRawTime = "<<totalRawTime<<std::endl;
NS_ASSERT (totalRawTime <= config.BeaconInterval);
}
}
// assumes each TIM has its own beacon - doesn't need to be the case as there has to be only
PageSliceCount beacons between DTIMs
bool check (uint16_t aid, uint32_t index)
{
uint8_t block = (aid >> 6 ) & 0x001f;
NS_ASSERT (config.pageS.GetPageSliceLen() > 0);
if (index == config.pageS.GetPageSliceCount() - 1 && config.pageS.GetPageSliceCount() != 0)
{
// the last page slice has 32 - the rest blocks
return (block <= 31) && (block >= index * config.pageS.GetPageSliceLen());
}
else if (config.pageS.GetPageSliceCount() == 0)
return true;

return (block >= index *
 config.pageS.GetPageSliceLen()) && (block < (index + 1) * config.pageS.GetPageSliceLen());
}


void sendStatistics(bool schedule) {
      eventManager.onUpdateStatistics(stats);
      eventManager.onUpdateSlotStatistics(
                  transmissionsPerTIMGroupAndSlotFromAPSinceLastInterval,
                  transmissionsPerTIMGroupAndSlotFromSTASinceLastInterval);
      // reset
      std::fill(transmissionsPerTIMGroupAndSlotFromAPSinceLastInterval.begin(),
                  transmissionsPerTIMGroupAndSlotFromAPSinceLastInterval.end(), 0);
      std::fill(transmissionsPerTIMGroupAndSlotFromSTASinceLastInterval.begin(),
                  transmissionsPerTIMGroupAndSlotFromSTASinceLastInterval.end(), 0);

      if (schedule)
            Simulator::Schedule(Seconds(config.visualizerSamplingInterval), &sendStatistics, true);
}

void onSTADeassociated(int i) {
      eventManager.onNodeDeassociated(*nodes[i]);
}

void updateNodesQueueLength() {
      for (uint32_t i = 0; i < config.Nsta; i++) {
            nodes[i]->UpdateQueueLength();
            stats.get(i).EDCAQueueLength = nodes[i]->queueLength;
      }
      Simulator::Schedule(Seconds(0.5), &updateNodesQueueLength);
}
```

```
void onSTAAssociated(int i) {
      // cout << "Node " << std::to_string(i) << " is associated and has aid "
      //             << nodes[i]->aId << endl;

      for (int k = 0; k < config.rps.rpsset.size(); k++) {
            for (int j = 0; j < config.rps.rpsset[k]->GetNumberOfRawGroups(); j++) {
                  if (config.rps.rpsset[k]->GetRawAssigmentObj(j).GetRawGroupAIDStart()
                              <= i + 1
                              && i + 1
                                          <= config.rps.rpsset[k]->GetRawAssigmentObj(j)
                                          .GetRawGroupAIDEnd()) {
                        nodes[i]->rpsIndex = k + 1;
                        nodes[i]->rawGroupNumber = j + 1;
                        nodes[i]->rawSlotIndex =
                                    nodes[i]->aId
                        % config.rps.rpsset[k]->GetRawAssigmentObj(j).GetSlotNum()
                                                + 1;
        /*cout << "Node " << i << " with AID "
        << (int)nodes[i]->aId << " belongs to " << (int)nodes[i]->rawSlotIndex
        << " slot of RAW group "
        << (int)nodes[i]->rawGroupNumber << " within the " << (int)nodes[i]->rpsIndex
        << " RPS." << endl;
                         */
                  }
            }
      }

      eventManager.onNodeAssociated(*nodes[i]);

      // RPS, Raw group and RAW slot assignment

      if (GetAssocNum() == config.Nsta) {
            cout << "All " << AssocNum << " stations associated at "
            << Simulator::Now ().GetMicroSeconds () <<", configuring clients & server" << endl;

            // association complete, start sending packets
            stats.TimeWhenEverySTAIsAssociated = Simulator::Now();

            if (config.trafficType == "udp") {
                  configureUDPServer();
                  configureUDPClients();
            }
// else if (config.trafficType == "udpecho") {
            //       configureUDPEchoServer();
            //       configureUDPEchoClients();
            // }
// else if (config.trafficType == "tcpecho") {
            //       configureTCPEchoServer();
            //       configureTCPEchoClients();
            // }
// else if (config.trafficType == "tcppingpong") {
            //       configureTCPPingPongServer();
            //       configureTCPPingPongClients();
            // }
// else if (config.trafficType == "tcpipcamera") {
            //       configureTCPIPCameraServer();
            //       configureTCPIPCameraClients();
            // }
// else if (config.trafficType == "tcpfirmware") {
            //       configureTCPFirmwareServer();
            //       configureTCPFirmwareClients();
            // }
// else if (config.trafficType == "tcpsensor") {
            //       configureTCPSensorServer();
            //       configureTCPSensorClients();
            // }
            updateNodesQueueLength();
      }
}

void RpsIndexTrace(uint16_t oldValue, uint16_t newValue) {
      currentRps = newValue;
      //cout << "RPS: " << newValue << " at " << Simulator::Now().GetMicroSeconds() << endl;
}

void RawGroupTrace(uint8_t oldValue, uint8_t newValue) {
      currentRawGroup = newValue;
      // cout << "      group " << std::to_string(newValue) << " at "
      << Simulator::Now().GetMicroSeconds() << endl;
}

void RawSlotTrace(uint8_t oldValue, uint8_t newValue) {
      currentRawSlot = newValue;
      // cout << "            slot " << std::to_string(newValue) << " at "
      << Simulator::Now().GetMicroSeconds() << endl;
}

void S1gBeaconTrace(S1gBeaconHeader beacon, RPS::RawAssignment raw) {
beaconCount += 1;
}

void configureNodes(NodeContainer& wifiStaNode, NetDeviceContainer& staDevice) {
      cout << "Configuring STA Node trace sources..." << endl;

      for (uint32_t i = 0; i < config.Nsta; i++) {

            // cout << "Hooking up trace sources for STA " << i << endl;

            NodeEntry* n = new NodeEntry(i, &stats, wifiStaNode.Get(i),
                        staDevice.Get(i));

            n->SetAssociatedCallback([ = ] {onSTAAssociated(i);});
            n->SetDeassociatedCallback([ = ] {onSTADeassociated(i);});

            nodes.push_back(n);
            // hook up Associated and Deassociated events
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Mac
                         /$ns3::RegularWifiMac/$ns3::StaWifiMac/Assoc",
                        MakeCallback(&NodeEntry::SetAssociation, n));
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac
                                    /$ns3::StaWifiMac/DeAssoc",
                        MakeCallback(&NodeEntry::UnsetAssociation, n));
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac
                                    /BE_EdcaTxopN/NrOfTransmissionsDuringRaw",
                        MakeCallback(
                                    &NodeEntry::OnNrOfTransmissionsDuringRAWSlotChanged,
                                    n));      //not implem
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac
                                    /$ns3::StaWifiMac/TransmissioninRAWSlot",
                        MakeCallback(
                                    &NodeEntry::OnTransmissionInsideRAWSlot,
                                    n));
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac
                                    /$ns3::StaWifiMac/RawDuration",
                        MakeCallback(
                                    &NodeEntry::onRawDurationChanged,
                                    n));
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac
                                    /$ns3::StaWifiMac/RawSlotDuration",
                        MakeCallback(
                                    &NodeEntry::OnRAWSlotDurationChanged,
                                    n));
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac
                                    /$ns3::StaWifiMac/AID",
                        MakeCallback(
                                     &NodeEntry::OnAIDChanged,
                                     n));

            //Config::Connect("/NodeList/" + std::to_string(i) + "/DeviceList/0/$ns3::WifiNetDevice/
            /$ns3::StaWifiMac/S1gBeaconMissed", MakeCallback(&NodeEntry::OnS1gBeaconMissed, n));

            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac
                                    /$ns3::StaWifiMac/PacketDropped",
                        MakeCallback(&NodeEntry::OnMacPacketDropped, n));
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac
                                    /$ns3::StaWifiMac/Collision",
                        MakeCallback(&NodeEntry::OnCollision, n));
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0
                    /$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac
                                    /$ns3::StaWifiMac/TransmissionWillCrossRAWBoundary",
                        MakeCallback(&NodeEntry::OnTransmissionWillCrossRAWBoundary,
                                    n)); //?

            // hook up TX
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
```

```
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Phy/PhyTxBegin",
                        MakeCallback(&NodeEntry::OnPhyTxBegin, n));
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Phy/PhyTxEnd",
                        MakeCallback(&NodeEntry::OnPhyTxEnd, n));
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Phy/PhyTxDropWithReason",
                        MakeCallback(&NodeEntry::OnPhyTxDrop, n)); //?

            // hook up RX
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Phy/PhyRxBegin",
                        MakeCallback(&NodeEntry::OnPhyRxBegin, n));
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Phy/PhyRxEnd",
                        MakeCallback(&NodeEntry::OnPhyRxEnd, n));
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Phy/PhyRxDropWithReason",
                        MakeCallback(&NodeEntry::OnPhyRxDrop, n));

            // hook up MAC traces
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/RemoteStationManager
                                    /MacTxRtsFailed",
                        MakeCallback(&NodeEntry::OnMacTxRtsFailed, n)); //?
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/RemoteStationManager
                                    /MacTxDataFailed",
                        MakeCallback(&NodeEntry::OnMacTxDataFailed, n));
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/RemoteStationManager
                                    /MacTxFinalRtsFailed",
                        MakeCallback(&NodeEntry::OnMacTxFinalRtsFailed, n)); //?
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/RemoteStationManager
                                    /MacTxFinalDataFailed",
                        MakeCallback(&NodeEntry::OnMacTxFinalDataFailed, n)); //?

            // hook up PHY State change
            Config::Connect(
                        "/NodeList/" + std::to_string(i)
                                    + "/DeviceList/0/$ns3::WifiNetDevice/Phy/State/State",
                        MakeCallback(&NodeEntry::OnPhyStateChange, n));

      }
}

void StaAIDMonitor ( void )
{
ofstream aidFile(config.name+"-aid.txt", ios::app);
```

```
aidFile << Simulator::Now().GetSeconds() << ",";
for (uint16_t i = 0; i < nodes.size(); i++)
{
uint32_t aid = nodes[i]->aId;
aidFile << aid << ",";
}
aidFile << endl;
Simulator::Schedule(Seconds(1), &StaAIDMonitor);
}

int getBandwidth(string dataMode) {
      if (dataMode == "MCS1_0" || dataMode == "MCS1_1" || dataMode == "MCS1_2"
                  || dataMode == "MCS1_3" || dataMode == "MCS1_4"
                  || dataMode == "MCS1_5" || dataMode == "MCS1_6"
                  || dataMode == "MCS1_7" || dataMode == "MCS1_8"
                  || dataMode == "MCS1_9" || dataMode == "MCS1_10")
            return 1;

      else if (dataMode == "MCS2_0" || dataMode == "MCS2_1"
                  || dataMode == "MCS2_2" || dataMode == "MCS2_3"
                  || dataMode == "MCS2_4" || dataMode == "MCS2_5"
                  || dataMode == "MCS2_6" || dataMode == "MCS2_7"
                  || dataMode == "MCS2_8")
            return 2;

      return 0;
}

string getWifiMode(string dataMode) {
      if (dataMode == "MCS1_0")
            return "OfdmRate300KbpsBW1MHz";
      else if (dataMode == "MCS1_1")
            return "OfdmRate600KbpsBW1MHz";
      else if (dataMode == "MCS1_2")
            return "OfdmRate900KbpsBW1MHz";
      else if (dataMode == "MCS1_3")
            return "OfdmRate1_2MbpsBW1MHz";
      else if (dataMode == "MCS1_4")
            return "OfdmRate1_8MbpsBW1MHz";
      else if (dataMode == "MCS1_5")
            return "OfdmRate2_4MbpsBW1MHz";
      else if (dataMode == "MCS1_6")
            return "OfdmRate2_7MbpsBW1MHz";
      else if (dataMode == "MCS1_7")
            return "OfdmRate3MbpsBW1MHz";
      else if (dataMode == "MCS1_8")
            return "OfdmRate3_6MbpsBW1MHz";
      else if (dataMode == "MCS1_9")
            return "OfdmRate4MbpsBW1MHz";
      else if (dataMode == "MCS1_10")
            return "OfdmRate150KbpsBW1MHz";

      else if (dataMode == "MCS2_0")
            return "OfdmRate650KbpsBW2MHz";
      else if (dataMode == "MCS2_1")
            return "OfdmRate1_3MbpsBW2MHz";
      else if (dataMode == "MCS2_2")
            return "OfdmRate1_95MbpsBW2MHz";
      else if (dataMode == "MCS2_3")
```

```
            return "OfdmRate2_6MbpsBW2MHz";
      else if (dataMode == "MCS2_4")
            return "OfdmRate3_9MbpsBW2MHz";
      else if (dataMode == "MCS2_5")
            return "OfdmRate5_2MbpsBW2MHz";
      else if (dataMode == "MCS2_6")
            return "OfdmRate5_85MbpsBW2MHz";
      else if (dataMode == "MCS2_7")
            return "OfdmRate6_5MbpsBW2MHz";
      else if (dataMode == "MCS2_8")
            return "OfdmRate7_8MbpsBW2MHz";
      return "";
}


double getDataRate(string dataMode) {
      if (dataMode == "MCS1_0")
            return 0.3;
      else if (dataMode == "MCS1_1")
            return 0.6;
      else if (dataMode == "MCS1_2")
            return 0.9;
      else if (dataMode == "MCS1_3")
            return 1.2;
      else if (dataMode == "MCS1_4")
            return 1.8;
      else if (dataMode == "MCS1_5")
            return 2.4;
      else if (dataMode == "MCS1_6")
            return 7.0;
      else if (dataMode == "MCS1_7")
            return 3.0;
      else if (dataMode == "MCS1_8")
            return 3.6;
      else if (dataMode == "MCS1_9")
            return 4.0;
      else if (dataMode == "MCS1_10")
            return 0.15;

      else if (dataMode == "MCS2_0")
            return 0.65;
      else if (dataMode == "MCS2_1")
            return 1.3;
      else if (dataMode == "MCS2_2")
            return 1.95;
      else if (dataMode == "MCS2_3")
            return 2.6;
      else if (dataMode == "MCS2_4")
            return 3.9;
      else if (dataMode == "MCS2_5")
            return 5.2;
      else if (dataMode == "MCS2_6")
            return 5.85;
      else if (dataMode == "MCS2_7")
            return 6.5;
      else if (dataMode == "MCS2_8")
            return 7.8;
      return 0.0;
}
```

```
void OnAPPhyRxDrop(std::string context, Ptr<const Packet> packet,
            DropReason reason) {
      // THIS REQUIRES PACKET METADATA ENABLE!
      auto pCopy = packet->Copy();
      auto it = pCopy->BeginItem();
      while (it.HasNext()) {

            auto item = it.Next();
            Callback<ObjectBase *> constructor = item.tid.GetConstructor();

            ObjectBase *instance = constructor();
            Chunk *chunk = dynamic_cast<Chunk *>(instance);
            chunk->Deserialize(item.current);

            if (dynamic_cast<WifiMacHeader*>(chunk)) {
                  WifiMacHeader* hdr = (WifiMacHeader*) chunk;

                  int staId = -1;
                  if (!config.useV6) {
                        for (uint32_t i = 0; i < staNodeInterface.GetN(); i++) {
                              if (wifiStaNode.Get(i)->GetDevice(0)->GetAddress()
                                          == hdr->GetAddr2()) {
                                    staId = i;
                                    break;
                              }
                        }
                  } else {
                        for (uint32_t i = 0; i < staNodeInterface6.GetN(); i++) {
                              if (wifiStaNode.Get(i)->GetDevice(0)->GetAddress()
                                          == hdr->GetAddr2()) {
                                    staId = i;
                                    break;
                              }
                        }
                  }
                  if (staId != -1) {
                        stats.get(staId).NumberOfDropsByReasonAtAP[reason]++;
                  }
                  delete chunk;
                  break;
            } else
                  delete chunk;
      }

}

void OnAPPacketToTransmitReceived(string context, Ptr<const Packet> packet,
            Mac48Address to, bool isScheduled, bool isDuringSlotOfSTA,
            Time timeLeftInSlot) {
      int staId = -1;
      if (!config.useV6) {
            for (uint32_t i = 0; i < staNodeInterface.GetN(); i++) {
                  if (wifiStaNode.Get(i)->GetDevice(0)->GetAddress() == to) {
                        staId = i;
                        break;
                  }
            }
      } else {
            for (uint32_t i = 0; i < staNodeInterface6.GetN(); i++) {
                  if (wifiStaNode.Get(i)->GetDevice(0)->GetAddress() == to) {
                        staId = i;
                        break;
                  }
            }
      }
      if (staId != -1) {
            if (isScheduled)
                  stats.get(staId).NumberOfAPScheduledPacketForNodeInNextSlot++;
            else {
                  stats.get(staId).NumberOfAPSentPacketForNodeImmediately++;
                  stats.get(staId).APTotalTimeRemainingWhenSendingPacketInSameSlot +=
                              timeLeftInSlot;
            }
      }
}

void OnAPReceivedOnRawGroup(string context, const WifiMacHeader *hdr,
uint16_t rawGroup, uint32_t pktSize, uint32_t rate, double signalDbm) {
rawGroupsStats[rawGroup-1].numberOfReceivedPackets += 1;
rawGroupsStats[rawGroup-1].totalByteReceived += pktSize;


macToStaParams[hdr->GetAddr2()].packetSize = pktSize;
macToStaParams[hdr->GetAddr2()].rate = rate;
macToStaParams[hdr->GetAddr2()].signalDbm = signalDbm;
macToStaParams[hdr->GetAddr2()].rawGroup = rawGroup;
}

void onChannelTransmission(Ptr<NetDevice> senderDevice, Ptr<Packet> packet) {
      int rpsIndex = currentRps - 1;
      int rawGroup = currentRawGroup - 1;
      int slotIndex = currentRawSlot - 1;
      //cout << rpsIndex << "            " << rawGroup << "            " << slotIndex << "

      uint64_t iSlot = slotIndex;
      if (rpsIndex > 0)
            for (int r = rpsIndex - 1; r >= 0; r--)
                  for (int g = 0; g < config.rps.rpsset[r]->GetNumberOfRawGroups(); g++)
                        iSlot += config.rps.rpsset[r]->GetRawAssigmentObj(g).GetSlotNum();

      if (rawGroup > 0)
            for (int i = rawGroup - 1; i >= 0; i--)
                  iSlot += config.rps.rpsset[rpsIndex]->GetRawAssigmentObj(i).GetSlotNum();

      if (rpsIndex >= 0 && rawGroup >= 0 && slotIndex >= 0)
      {
            if (senderDevice->GetAddress() == apDevice.Get(0)->GetAddress())
            {
                  // from AP
                  transmissionsPerTIMGroupAndSlotFromAPSinceLastInterval[iSlot] +=
                  packet->GetSerializedSize();
            }
            else
            {
                  // from STA
                  transmissionsPerTIMGroupAndSlotFromSTASinceLastInterval[iSlot] +=
                  packet->GetSerializedSize();
            }
      }
      //std::cout << "------------- packetSerializedSize =
      " << packet->GetSerializedSize() << std::endl;
      //std::cout << "------------- txAP[" << iSlot <<"] =
      " << transmissionsPerTIMGroupAndSlotFromAPSinceLastInterval[iSlot] << std::endl;
      //std::cout << "------------- txSTA[" << iSlot <<"] =
      " << transmissionsPerTIMGroupAndSlotFromSTASinceLastInterval[iSlot] << std::endl;

}

int getSTAIdFromAddress(Ipv4Address from) {
      int staId = -1;
      for (int i = 0; i < staNodeInterface.GetN(); i++) {
            if (staNodeInterface.GetAddress(i) == from) {
                  staId = i;
                  break;
            }
      }
      return staId;
}

void udpPacketReceivedAtServer(Ptr<const Packet> packet, Address from) { //works
      int staId = getSTAIdFromAddress(
                  InetSocketAddress::ConvertFrom(from).GetIpv4());
      if (staId != -1)
            nodes[staId]->OnUdpPacketReceivedAtAP(packet);
      else
            cout << "*** Node could not be determined from received packet at AP "
                        << endl;
}

void tcpPacketReceivedAtServer(Ptr<const Packet> packet, Address from) {
int staId = getSTAIdFromAddress(
InetSocketAddress::ConvertFrom(from).GetIpv4());
if (staId != -1)
nodes[staId]->OnTcpPacketReceivedAtAP(packet);
else
cout << "*** Node could not be determined from received packet at AP "
<< endl;
}

void tcpRetransmissionAtServer(Address to) {
int staId = getSTAIdFromAddress(Ipv4Address::ConvertFrom(to));
if (staId != -1)
nodes[staId]->OnTcpRetransmissionAtAP();
else
cout << "*** Node could not be determined from received packet at AP "
<< endl;
}

void tcpPacketDroppedAtServer(Address to, Ptr<Packet> packet,
DropReason reason) {
int staId = getSTAIdFromAddress(Ipv4Address::ConvertFrom(to));
if (staId != -1) {
stats.get(staId).NumberOfDropsByReasonAtAP[reason]++;
}
}
```

```
void tcpStateChangeAtServer(TcpSocket::TcpStates_t oldState,
TcpSocket::TcpStates_t newState, Address to) {

int staId = getSTAIdFromAddress(
InetSocketAddress::ConvertFrom(to).GetIpv4());
if (staId != -1)
nodes[staId]->OnTcpStateChangedAtAP(oldState, newState);
else
cout << "*** Node could not be determined from received packet at AP "
<< endl;

//cout << Simulator::Now().GetMicroSeconds() << " ********** TCP SERVER SOCKET STATE CHANGED FROM "
}

void tcpIPCameraDataReceivedAtServer(Address from, uint16_t nrOfBytes) {
int staId = getSTAIdFromAddress(
InetSocketAddress::ConvertFrom(from).GetIpv4());
if (staId != -1)
nodes[staId]->OnTcpIPCameraDataReceivedAtAP(nrOfBytes);
else
cout << "*** Node could not be determined from received packet at AP "
                        << endl;
}

void wireTCPServer(ApplicationContainer serverApp) {
serverApp.Get(0)->TraceConnectWithoutContext("Rx",
MakeCallback(&tcpPacketReceivedAtServer));
serverApp.Get(0)->TraceConnectWithoutContext("Retransmission",
MakeCallback(&tcpRetransmissionAtServer));
serverApp.Get(0)->TraceConnectWithoutContext("PacketDropped",
MakeCallback(&tcpPacketDroppedAtServer));
serverApp.Get(0)->TraceConnectWithoutContext("TCPStateChanged",
MakeCallback(&tcpStateChangeAtServer));

if (config.trafficType == "tcpipcamera") {
serverApp.Get(0)->TraceConnectWithoutContext("DataReceived",
MakeCallback(&tcpIPCameraDataReceivedAtServer));
}
}

void wireTCPClient(ApplicationContainer clientApp, int i) {

clientApp.Get(0)->TraceConnectWithoutContext("Tx",
MakeCallback(&NodeEntry::OnTcpPacketSent, nodes[i]));
clientApp.Get(0)->TraceConnectWithoutContext("Rx",
MakeCallback(&NodeEntry::OnTcpEchoPacketReceived, nodes[i]));

clientApp.Get(0)->TraceConnectWithoutContext("CongestionWindow",
MakeCallback(&NodeEntry::OnTcpCongestionWindowChanged, nodes[i]));
clientApp.Get(0)->TraceConnectWithoutContext("RTO",
MakeCallback(&NodeEntry::OnTcpRTOChanged, nodes[i]));
clientApp.Get(0)->TraceConnectWithoutContext("RTT",
MakeCallback(&NodeEntry::OnTcpRTTChanged, nodes[i]));
clientApp.Get(0)->TraceConnectWithoutContext("SlowStartThreshold",
MakeCallback(&NodeEntry::OnTcpSlowStartThresholdChanged, nodes[i]));
clientApp.Get(0)->TraceConnectWithoutContext("EstimatedBW",
MakeCallback(&NodeEntry::OnTcpEstimatedBWChanged, nodes[i]));
```

```
clientApp.Get(0)->TraceConnectWithoutContext("TCPStateChanged",
MakeCallback(&NodeEntry::OnTcpStateChanged, nodes[i]));
clientApp.Get(0)->TraceConnectWithoutContext("Retransmission",
MakeCallback(&NodeEntry::OnTcpRetransmission, nodes[i]));

clientApp.Get(0)->TraceConnectWithoutContext("PacketDropped",
MakeCallback(&NodeEntry::OnTcpPacketDropped, nodes[i]));

if (config.trafficType == "tcpfirmware") {
clientApp.Get(0)->TraceConnectWithoutContext("FirmwareUpdated",
MakeCallback(&NodeEntry::OnTcpFirmwareUpdated, nodes[i]));
} else if (config.trafficType == "tcpipcamera") {
clientApp.Get(0)->TraceConnectWithoutContext("DataSent",
MakeCallback(&NodeEntry::OnTcpIPCameraDataSent, nodes[i]));
clientApp.Get(0)->TraceConnectWithoutContext("StreamStateChanged",
MakeCallback(&NodeEntry::OnTcpIPCameraStreamStateChanged,
nodes[i]));
}
}

void configureUDPServer() {
UdpServerHelper myServer(9);
serverApp = myServer.Install(wifiApNode);
serverApp.Get(0)->TraceConnectWithoutContext("Rx",
MakeCallback(&udpPacketReceivedAtServer));
serverApp.Start(Seconds(0));

}

void configureUDPClients() {
    //Application start time
    Ptr<UniformRandomVariable> m_rv = CreateObject<UniformRandomVariable>();

UdpClientHelper myClient(apNodeInterface.GetAddress(0), 9); //address of remote node
myClient.SetAttribute("MaxPackets",  UintegerValue(4294967295u));

cout << "Configure UDP Clients. " << endl;
for (uint16_t k = 0; k < topology.GetNumZones(); k++)
{
Zone * zone = topology.m_zones[k];
// uint32_t packetSize = zone->GetMaxPaySize();
uint32_t trafficInterval = zone->GetMaxTrafficInter();

myClient.SetAttribute("Interval",
TimeValue(MilliSeconds(trafficInterval)));

for (auto i: zone->m_nodes)
{
uint32_t packetSize = m_rv->GetInteger(zone->GetMinPaySize(), zone->GetMaxPaySize());
myClient.SetAttribute("PacketSize", UintegerValue(packetSize));
stats.get(i).SetPayLoadSize(packetSize);
ApplicationContainer clientApp = myClient.Install(
wifiStaNode.Get(i));
wireTCPClient(clientApp, i);

double random = m_rv->GetValue(0, trafficInterval);
clientApp.Start(MilliSeconds(5 + random));
clientApp.Stop(Seconds(config.simulationTime + 5));
}
```

```
}

}


Time timeIdleArray[MaxSta];
Time timeRxArray[MaxSta];
Time timeTxArray[MaxSta];
Time timeSleepArray[MaxSta];
Time timeCollisionArray[MaxSta];

Time timeIdleNotAssociated[MaxSta];
Time timeRxNotAssociated[MaxSta];
Time timeTxNotAssociated[MaxSta];
Time timeSleepNotAssociated[MaxSta];
Time timeCollisionNotAssociated[MaxSta];

double dist[MaxSta];

//it prints the information regarding the state of the device
void PhyStateTrace(std::string context, Time start, Time duration,
            enum WifiPhy::State state) {

/*Get the number of the node from the context*/
/*context = "/NodeList/"+strSTA+"/DeviceList/'*'/Phy/$ns3::YansWifiPhy/State/State"*/
unsigned first = context.find("t/");
unsigned last = context.find("/D");
string strNew = context.substr((first + 2), (last - first - 2));

int node = std::stoi(strNew);

if (nodes[node]->isAssociated)
{
switch (state)
{
case WifiPhy::State::SLEEP: //Sleep
timeSleepArray[node] = timeSleepArray[node] + duration;
//NS_LOG_UNCOND(to_string(node + 1) + ",SLEEP,"
    +to_string(start.GetMicroSeconds()) + " " + to_string(duration.GetMicroSeconds()));
break;
case WifiPhy::State::IDLE: //Idle
timeIdleArray[node] = timeIdleArray[node] + duration;
//NS_LOG_UNCOND(to_string(node + 1) + ",IDLE,"
    +to_string(start.GetMicroSeconds()) + " " + to_string(duration.GetMicroSeconds()));
break;
case WifiPhy::State::TX: //Tx
timeTxArray[node] = timeTxArray[node] + duration;
//NS_LOG_UNCOND (to_string(node+1) + ",TX",
    +to_string(start.GetMicroSeconds()) + " " + to_string(duration.GetMicroSeconds()));
break;
case WifiPhy::State::RX: //Rx
timeRxArray[node] = timeRxArray[node] + duration;
//NS_LOG_UNCOND (to_string(node+1) + ",RX,"
    +to_string(start.GetMicroSeconds()) + " " + to_string(duration.GetMicroSeconds()));
break;
case WifiPhy::State::CCA_BUSY: //CCA_BUSY
timeCollisionArray[node] = timeCollisionArray[node] + duration;
//NS_LOG_UNCOND (to_string(node+1) + ",CCA_BUSY,"
    + to_string(start.GetMicroSeconds()) + " " + to_string(duration.GetMicroSeconds()));
break;
}
}
else
{
switch (state)
{
case WifiPhy::State::SLEEP: //Sleep
timeSleepNotAssociated[node] = timeSleepNotAssociated[node] + duration;
//NS_LOG_UNCOND(to_string(node + 1) + ",SLEEP,"
   +
    to_string(start.GetMicroSeconds())+
    " "+ to_string(duration.GetMicroSeconds()));
break;
case WifiPhy::State::IDLE: //Idle
timeIdleNotAssociated[node] = timeIdleNotAssociated[node] + duration;
//NS_LOG_UNCOND(to_string(node + 1) + ",IDLE," + to_string(start.GetMicroSeconds()) + " "
    + to_string(duration.GetMicroSeconds()));
break;
case WifiPhy::State::TX: //Tx
timeTxNotAssociated[node] = timeTxNotAssociated[node] + duration;
//NS_LOG_UNCOND (to_string(node+1) + ",TX," +
    +to_string(start.GetMicroSeconds()) + " "
    + to_string(duration.GetMicroSeconds()));
break;
case WifiPhy::State::RX: //Rx
timeRxNotAssociated[node] = timeRxNotAssociated[node] + duration;
//NS_LOG_UNCOND (to_string(node+1) + ",RX," + to_string(start.GetMicroSeconds()) + " "
    + to_string(duration.GetMicroSeconds()));
break;
case WifiPhy::State::CCA_BUSY: //CCA_BUSY
timeCollisionNotAssociated[node] = timeCollisionNotAssociated[node] + duration;
//NS_LOG_UNCOND (to_string(node+1) + ",CCA_BUSY," + to_string(start.GetMicroSeconds()) + " "
    + to_string(duration.GetMicroSeconds()));
break;
}
}
}

std::vector<double>
GetPositionOnCircle(double cx, double cy, double minR, double maxR){
std::vector<double> coords(2); // {x, y}
Ptr<UniformRandomVariable> randVal = CreateObject<UniformRandomVariable> ();

double radius = randVal->GetValue(minR, maxR);

coords[0] = randVal->GetValue(cx-radius, cx+radius);
coords[1] = sqrt(pow(radius, 2) - pow((coords[0]-cx), 2));

uint32_t sum = randVal->GetInteger(0, 1);

if (sum) coords[1] = cy - coords[1];
else coords[1] = cy + coords[1];

return coords;
}

int main (int argc, char *argv[]) {
```

```
// Logging components

// LogComponentEnable ("UdpServer", LOG_INFO);
    // LogComponentEnable ("UdpClient", LOG_INFO);
// LogComponentEnable ("UdpEchoServerApplication", LOG_INFO);
// LogComponentEnable ("UdpEchoClientApplication", LOG_INFO);

// LogComponentEnable ("ApWifiMac", LOG_DEBUG);
// LogComponentEnable ("StaWifiMac", LOG_DEBUG);
// LogComponentEnable ("EdcaTxopN", LOG_DEBUG);
// LogComponentEnable ("S1gTccRaw", LOG_INFO);
// LogComponentEnable ("Kmeans", LOG_DEBUG);

// LogComponentEnableAll (LOG_DEBUG);


// Create a NetAnim animation for simulation visualization
AnimationInterface * pAnim = 0;

// If true, shows the current position of each node
bool OutputPosition = true;

// Parse configurations
config = Configuration(argc, argv);

// Rng Seed
RngSeedManager::SetSeed(config.seed);

ConfigureTopology(config.topologyFile);

// Setup RAW and TIM
config.rps = configureRAW(config.rps, config.RAWConfigFile);

config.Nsta = topology.GetTotalSta();
config.NRawSta = config.Nsta;

configurePageSlice ();
configureTIM ();
checkRawAndTimConfiguration ();

// What does a NSS file do?
config.NSSFile = config.trafficType + "_" + std::to_string(config.Nsta)
+ "sta_" + std::to_string(config.NGroup) + "Group_"
+ std::to_string(config.NRawSlotNum) + "slots_"
// + std::to_string(config.payloadSize) + "payload_"
+ std::to_string(config.BeaconInterval) + "BI" + ".nss";

// Simulation statistics
stats = Statistics(config.Nsta);
eventManager = SimulationEventManager(config.visualizerIP,
config.visualizerPort, config.NSSFile);


uint32_t totalRawGroups(0);
for (int i = 0; i < config.rps.rpsset.size(); i++) {
int nRaw = config.rps.rpsset[i]->GetNumberOfRawGroups();
totalRawGroups += nRaw;
for (int j = 0; j < nRaw; j++) {
config.totalRawSlots += config.rps.rpsset[i]->GetRawAssigmentObj(j).GetSlotNum();
}

}

for (uint16_t i = 0; i < totalRawGroups; i++)
{
RawGroupStats rawStats;

rawGroupsStats.push_back(rawStats);

}


NS_LOG_INFO("Total RAW Groups = " << totalRawGroups);
NS_LOG_INFO("Total RAW Slots = " << config.totalRawSlots);

transmissionsPerTIMGroupAndSlotFromAPSinceLastInterval = vector<long>(
config.totalRawSlots, 0);
transmissionsPerTIMGroupAndSlotFromSTASinceLastInterval = vector<long>(
config.totalRawSlots, 0);


// for (uint16_t i = 0; i < topology.GetNumZones(); i++)
// {
//  uint16_t cNodes = wifiStaNode.GetN();
//  NodeContainer zoneContainer;

//  uint16_t zoneSize = topology.m_zones[i]->GetNumSta();
//  zoneContainer.Create(zoneSize);

//  vector<uint16_t> staIds;
//  for (uint16_t idx = cNodes; idx < cNodes + zoneSize; idx++)
//  {
//  staIds.push_back(idx);
//  }
//  topology.m_zones[i]->SetNodes(staIds);
//  wifiStaNode.Add(zoneContainer);
// }
// // wifiStaNode.Create(config.Nsta);
// wifiApNode.Create(1);

NS_LOG_INFO("Created " << config.Nsta << " station nodes and the AP node.");

YansWifiChannelHelper channelBuilder = YansWifiChannelHelper();
channelBuilder.AddPropagationLoss("ns3::LogDistancePropagationLossModel",
"Exponent", DoubleValue(3.76), "ReferenceLoss", DoubleValue(8.0),
"ReferenceDistance", DoubleValue(1.0));
channelBuilder.SetPropagationDelay(
"ns3::ConstantSpeedPropagationDelayModel");
    channelBuider.AddPropagationLoss("ns3::NakagamiPropagationLossModel","m0", DoubleValue(1),
    "m1", DoubleValue(1),"m2", DoubleValue(1));
Ptr<YansWifiChannel> channel = channelBuilder.Create();

YansWifiPhyHelper phy = YansWifiPhyHelper::Default();
phy.SetErrorRateModel("ns3::YansErrorRateModel");
phy.SetChannel(channel);
phy.Set("ShortGuardEnabled", BooleanValue(false));
phy.Set("ChannelWidth", UintegerValue(getBandwidth(config.DataMode))); // changed
phy.Set("EnergyDetectionThreshold", DoubleValue(-110.0));
phy.Set("CcaMode1Threshold", DoubleValue(-113.0));
phy.Set("TxGain", DoubleValue(0.0));
phy.Set("RxGain", DoubleValue(0.0));
phy.Set("TxPowerLevels", UintegerValue(1));
phy.Set("TxPowerEnd", DoubleValue(0.0));
phy.Set("TxPowerStart", DoubleValue(0.0));
phy.Set("RxNoiseFigure", DoubleValue(6.8));
phy.Set("LdpcEnabled", BooleanValue(true));
phy.Set("S1g1MfieldEnabled", BooleanValue(config.S1g1MfieldEnabled));

WifiHelper wifi = WifiHelper::Default();
wifi.SetStandard(WIFI_PHY_STANDARD_80211ah);

S1gWifiMacHelper mac = S1gWifiMacHelper::Default();
Ssid ssid = Ssid("ns380211ah");
mac.SetType("ns3::StaWifiMac", "Ssid", SsidValue(ssid),
"ActiveProbing",BooleanValue(false),
"DynamicAidSupported", BooleanValue(true),
"UnDynamicAidSupported", BooleanValue(true));



NetDeviceContainer staDevice;

// vector<string> dataModes = {"MCS2_3", "MCS2_2", "MCS2_1", "MCS2_0"};

vector<double> datarates;

for (auto zone: topology.m_zones)
{
uint16_t cNodes = wifiStaNode.GetN();
uint16_t zoneSize = zone->GetNumSta();

StringValue DataRate;
DataRate = StringValue(getWifiMode(zone->GetDataMode()));

NodeContainer zoneContainer;
zoneContainer.Create(zoneSize);

wifiStaNode.Add(zoneContainer);

vector<uint16_t> staIds;
for (uint16_t idx = cNodes; idx < cNodes + zoneSize; idx++)
{
staIds.push_back(idx);

Ptr<Node> staNode = wifiStaNode.Get (idx);
NetDeviceContainer staDev;

string dataMode = zone->GetDataMode();
if (dataMode == "random")
dataMode = GetRandomDataMode(topology.GetNumZones());
else if (dataMode == "round")
dataMode = GetRoundRoubindDataMode(idx, topology.GetNumZones());

double datarate = getDataRate(dataMode);
stats.get(idx).SetDataRate(datarate);

StringValue DataRate;
DataRate = StringValue(getWifiMode(dataMode));
wifi.SetRemoteStationManager
   ("ns3::ConstantRateWifiManager",DataMode", DataRate,
   "ControlMode", DataRate);
staDev = wifi.Install(phy, mac, staNode);
staDevice.Add(staDev);
}
cout << endl;
zone->SetNodes(staIds);
}


// uint32_t nNodes = wifiStaNode.GetN ();
// for (uint32_t i = 0; i < nNodes; ++i)
// {
//  Ptr<Node> staNode = wifiStaNode.Get (i);

//  StringValue DataRate;
//  DataRate = StringValue(getWifiMode(dataModes[i%totalRawGroups]));
//  double datarate = getDataRate(dataModes[i%totalRawGroups]);
//  wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager", "DataMode",
 DataRate, "ControlMode", DataRate);
//  NetDeviceContainer staDev;
//  staDev = wifi.Install(phy, mac, staNode);
//  staDevice.Add(staDev);
// NS_LOG_INFO("Station " << i + 1 << " - DataMode: " << DataRate.Get()
 << " Datarate: " << datarate);
// }


// wifiStaNode.Create(config.Nsta);
wifiApNode.Create(1);
mac.SetType ("ns3::ApWifiMac",
"Ssid", SsidValue (ssid),
"BeaconInterval", TimeValue (MicroSeconds(config.BeaconInterval)),
"NRawStations", UintegerValue (config.NRawSta),
"RPSsetup", RPSVectorValue (config.rps),
"PageSliceSet", pageSliceValue (config.pageS),
"TIMSet", TIMValue (config.tim),
"DynamicAidSupported", BooleanValue(true),
"UnDynamicAidSupported", BooleanValue(true)
);
mac.SetType("ns3::ApWifiMac",
"RAWGroupping", StringValue(config.RawGroupping));

phy.Set("TxGain", DoubleValue(3.0));
phy.Set("RxGain", DoubleValue(3.0));
phy.Set("TxPowerLevels", UintegerValue(1));
phy.Set("TxPowerEnd", DoubleValue(30.0));
phy.Set("TxPowerStart", DoubleValue(30.0));
phy.Set("RxNoiseFigure", DoubleValue(6.8));

apDevice = wifi.Install(phy, mac, wifiApNode);

phy.EnablePcap ("ap-pcap", apDevice.Get(0));
Config::Set(
"/NodeList/*/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac/BE_EdcaTxopN/Queue
    /MaxPacketNumber",
UintegerValue(10));
```

```
Config::Set(
"/NodeList/*/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac/BE_EdcaTxopN/Queue
    /MaxDelay",
TimeValue(NanoSeconds(6000000000000)));

std::ostringstream oss;
oss << "/NodeList/" << wifiApNode.Get(0)->GetId()
<< "/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac/$ns3::ApWifiMac/";

Config::ConnectWithoutContext(oss.str() + "RpsIndex", MakeCallback(&RpsIndexTrace));
Config::ConnectWithoutContext(oss.str() + "RawGroup", MakeCallback(&RawGroupTrace));
Config::ConnectWithoutContext(oss.str() + "RawSlot", MakeCallback(&RawSlotTrace));
Config::ConnectWithoutContext(oss.str() + "S1gBeaconBroadcasted", MakeCallback(&S1gBeaconTrace));

oss.str("");
oss.clear();
oss << "/NodeList/" << wifiApNode.Get(0)->GetId()
<< "/DeviceList/0/$ns3::WifiNetDevice/Phy/MonitorSnifferRx";

// mobility.
MobilityHelper mobility;
double xpos = std::stoi(config.rho, nullptr, 0);
double ypos = xpos;

Ptr<ListPositionAllocator> positionAllocSta = CreateObject<
ListPositionAllocator>();

std::vector<double> coords;
// for (int i = 0; i < config.Nsta; i++){
//  double minR = (xpos / totalRawGroups) * (i % totalRawGroups);
//  double maxR = (xpos / totalRawGroups) * (i % totalRawGroups + 1);
//  coords = GetPositionOnCircle(xpos, ypos, minR, maxR);
//  positionAllocSta->Add(Vector(coords[0], coords[1], 0.0));
// }
for (uint16_t m = 0; m < topology.GetNumZones(); m++)
{
Zone * zone = topology.m_zones[m];
uint16_t minR = zone->GetMinRadius();
uint16_t maxR = zone->GetMaxRadius();
for (auto k: zone->m_nodes)
{
coords = GetPositionOnCircle(xpos, ypos, (double)minR, (double)maxR);
positionAllocSta->Add(Vector(coords[0], coords[1], 0.0));
}
}


mobility.SetPositionAllocator(positionAllocSta);
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");

for (uint16_t i = 0; i < wifiStaNode.GetN(); i++)
{
if (i != 0)
mobility.Install(wifiStaNode.Get(i));
}

mobility.SetMobilityModel("ns3::WaypointMobilityModel");
mobility.Install(wifiStaNode.Get(0));
```

```
Ptr <WaypointMobilityModel> wpMobility =
 DynamicCast <WaypointMobilityModel> (wifiStaNode.Get(0)->GetObject<MobilityModel>());

wpMobility->AddWaypoint(Waypoint (Seconds(0.0), Vector(xpos+5, ypos, 0.0)));
wpMobility->AddWaypoint(Waypoint (Seconds(49.9), Vector(xpos+5, ypos, 0.0)));
wpMobility->AddWaypoint(Waypoint (Seconds(50), Vector(xpos+190.0, ypos, 0.0)));

wpMobility->AddWaypoint(Waypoint (Seconds(69.9), Vector(xpos+190.0, ypos, 0.0)));
wpMobility->AddWaypoint(Waypoint (Seconds(70), Vector(xpos+100, ypos, 0.0)));

MobilityHelper mobilityAp;
Ptr<ListPositionAllocator> positionAlloc = CreateObject<
ListPositionAllocator>();
positionAlloc->Add(Vector(xpos, ypos, 0.0));
mobilityAp.SetPositionAllocator(positionAlloc);
mobilityAp.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobilityAp.Install(wifiApNode);

/* Internet stack*/
InternetStackHelper stack;
stack.Install(wifiApNode);
stack.Install(wifiStaNode);

Ipv4AddressHelper address;

address.SetBase("192.168.0.0", "255.255.0.0");

staNodeInterface = address.Assign(staDevice);
apNodeInterface = address.Assign(apDevice);

//trace association
std::cout << "Configuring trace sources..." << std::endl;
for (uint16_t kk = 0; kk < config.Nsta; kk++) {
std::ostringstream STA;
STA << kk;
std::string strSTA = STA.str();

assoc_record *m_assocrecord = new assoc_record;
m_assocrecord->setstaid(kk);
Config::Connect(
"/NodeList/" + strSTA
+ "/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac/$ns3::StaWifiMac/Assoc",
MakeCallback(&assoc_record::SetAssoc, m_assocrecord));
Config::Connect(
"/NodeList/" + strSTA
+ "/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::RegularWifiMac/$ns3::StaWifiMac/DeAssoc",
MakeCallback(&assoc_record::UnsetAssoc, m_assocrecord));
assoc_vector.push_back(m_assocrecord);
}

std::cout << "Populating routing tables..." << std::endl;
Ipv4GlobalRoutingHelper::PopulateRoutingTables();
std::cout << "Populating ARP cache..." << std::endl;
PopulateArpCache();

// configure tracing for associations & other metrics
std::cout << "Configuring trace sinks for nodes..." << std::endl;
configureNodes(wifiStaNode, staDevice);
```

```
Config::Connect(
"/NodeList/" + std::to_string(config.Nsta)
+ "/DeviceList/0/$ns3::WifiNetDevice/Phy/PhyRxDropWithReason",
MakeCallback(&OnAPPhyRxDrop));
Config::Connect(
"/NodeList/" + std::to_string(config.Nsta)
+ "/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::ApWifiMac/PacketToTransmitReceivedFromUpperLayer",
MakeCallback(&OnAPPacketToTransmitReceived));
Config::Connect(
"/NodeList/" + std::to_string(config.Nsta)
+ "/DeviceList/0/$ns3::WifiNetDevice/Mac/$ns3::ApWifiMac/ReceivedOnRawGroup",
MakeCallback(&OnAPReceivedOnRawGroup));

Ptr<MobilityModel> mobility1 =
wifiApNode.Get(0)->GetObject<MobilityModel>();
Vector apposition = mobility1->GetPosition();
if (OutputPosition) {
uint32_t i = 0;
while (i < config.Nsta) {
Ptr<MobilityModel> mobility = wifiStaNode.Get(i)->GetObject<
MobilityModel>();
Vector position = mobility->GetPosition();
nodes[i]->x = position.x;
nodes[i]->y = position.y;
std::cout << "Sta node#" << i << ", " << "position = " << position
<< std::endl;
dist[i] = mobility->GetDistanceFrom(
wifiApNode.Get(0)->GetObject<MobilityModel>());
i++;
}
std::cout << "AP node, position = " << apposition << std::endl;
}

/*Print of the state of the stations*/
for (uint32_t i = 0; i < config.Nsta; i++) {
std::ostringstream STA;
STA << i;
std::string strSTA = STA.str();

Config::Connect(
"/NodeList/" + strSTA
+ "/DeviceList/*/Phy/$ns3::YansWifiPhy/State/State",
MakeCallback(&PhyStateTrace));
}

eventManager.onStartHeader();
eventManager.onStart(config);
if (config.rps.rpsset.size() > 0)
for (uint32_t i = 0; i < config.rps.rpsset.size(); i++)
for (uint32_t j = 0;
j < config.rps.rpsset[i]->GetNumberOfRawGroups(); j++)
eventManager.onRawConfig(i, j,
config.rps.rpsset[i]->GetRawAssigmentObj(j));

for (uint32_t i = 0; i < config.Nsta; i++)
eventManager.onSTANodeCreated(*nodes[i]);

eventManager.onAPNodeCreated(apposition.x, apposition.y);
eventManager.onStatisticsHeader();
```

```
sendStatistics(true);

Simulator::Stop(Seconds(config.simulationTime + config.CoolDownPeriod));
 // allow up to a minute after the client & server apps are finished to process the queue

pAnim = new AnimationInterface (config.name+"-netanim.xml");
pAnim->SetMaxPktsPerTraceFile(50000000);

// Instala FlowMonitor em todos os nós
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll ();

Simulator::Schedule(Seconds(1), &StaAIDMonitor);

Simulator::Run();

monitor->CheckForLostPackets ();

ofstream delayFile(config.name+"-delay.txt", ios::app);

//   // Mostra estatísticas por fluxo
//   Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
(flowmon.GetClassifier ());
//   std::map<FlowId, FlowMonitor::FlowStats> flowStats = monitor->GetFlowStats ();

//   double totalDelay = 0;
//   double totalRxPackets = 0;

//   for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = flowStats.begin ();
i != flowStats.end (); ++i)
//   {
//  Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
//  std::cout << "Flow " << i->first << " (" << t.sourceAddress << ":" << t.sourcePort
<< " -> "<< t.destinationAddress << ":" << t.destinationPort << ")\n";
//  std::cout << "  Tx Pacotes: " << i->second.txPackets << "\n";
//  // std::cout << "  Tx Bytes:   " << i->second.txBytes << "\n";
//  std::cout << "  Tx Taxa de bits média:  " << ((i->second.txBytes * 8.0)/
((i->second.timeLastTxPacket.GetSeconds()-i->second.timeFirstTxPacket.GetSeconds()))  << " bps\n";
//  std::cout << "  Rx Pacotes: " << i->second.rxPackets << "\n";
//  // std::cout << "  Rx Bytes:   " << i->second.rxBytes << "\n";
//  // std::cout << "  Rx Taxa de bits média: " << ((i->second.rxBytes * 8.0)/
((i->second.timeLastRxPacket.GetSeconds()-i->second.timeFirstRxPacket.GetSeconds()))  << " bps\n";
//  // std::cout << "  Throughput médio (simulação): " << (i->second.rxBytes * 8.0)
/config.simulationTime << " bps\n";
//  // std::cout << "  Atraso médio: " << i->second.delaySum.GetSeconds()/i->second.rxPackets
<< " s\n";
//  totalDelay += i->second.delaySum.GetSeconds();
//  totalRxPackets += i->second.rxPackets;
//  // std::cout << "  Jitter médio: " << 1000 * (i->second.jitterSum.GetSeconds()/
(i->second.rxPackets -1)) << " ms\n";
//  // std::cout << "  Número de pacotes perdidos: " << i->second.lostPackets << "\n";
//   }

//  cout << "Total Delay: " << totalDelay << endl;
//  cout << "N RX packets: " << totalRxPackets << endl;
//  cout << "Average Delay: " << totalDelay / totalRxPackets << endl;

//  delayFile << totalDelay / totalRxPackets << endl;
// Visualizer throughput
int pay = 0, totalSuccessfulPackets = 0, totalSentPackets = 0, totalPacketsEchoed = 0;
uint32_t totalBytesReceived  = 0;

int k = 0;
for (auto zone: topology.m_zones)
{
for (auto i: zone->m_nodes)
{
totalSuccessfulPackets += stats.get(i).NumberOfSuccessfulPackets;
totalBytesReceived += (stats.get(i).NumberOfSuccessfulPackets * stats.get(i).PayLoadSize);
cout << i << " sent: " << stats.get(i).NumberOfSentPackets
<< " ; delivered: " << stats.get(i).NumberOfSuccessfulPackets
<< " ; echoed: " << stats.get(i).NumberOfSuccessfulRoundtripPackets
<< "; packetloss: "
<< stats.get(i).GetPacketLoss(config.trafficType) << endl;
}
k++;
}

// Result files
ofstream throughputFile(config.name+"-throughput.txt", ios::app);
if (config.trafficType == "udp")
{
double throughput = 0;
uint32_t totalPacketsThrough =
DynamicCast<UdpServer>(serverApp.Get(0))->GetReceived();
throughput = totalBytesReceived * 8
/ (config.simulationTime * 1000000.0);
cout << "totalPacketsThrough " << totalPacketsThrough << " ++my "
<< totalSuccessfulPackets << endl;
cout << "throughput " << throughput << " ++my "
<< totalBytesReceived * 8. / (config.simulationTime * 1000000.0) << endl;
std::cout << "throughput" << std::endl;
std::cout << throughput << " Mbit/s"
<< std::endl;

throughputFile << throughput * 1000 <<",";

}
cout << "total packet loss % "
<< 100 - 100. * totalPacketsEchoed / totalSentPackets << endl;

ofstream lossFile(config.name+"-loss.txt", ios::app);
lossFile << 100 - 100. * totalPacketsEchoed / totalSentPackets << endl;
lossFile.close();

Simulator::Destroy();

ofstream risultati;
string addressresults = config.OutputPath + "moreinfo.txt";
risultati.open(addressresults.c_str(), ios::out | ios::trunc);

risultati << "Sta node#,distance,timerx(notassociated),timeidle(notassociated),
 timetx(notassociated),timesleep(notassociated),timecollision(notassociated)" << std::endl;
int i = 0;
string spazio = ",";
```

```
uint16_t slotDuraitonCount = config.rps.rpsset[0]->GetRawAssigmentObj(0).GetSlotDurationCount();
uint16_t numRAWSlots = config.rps.rpsset[0]->GetRawAssigmentObj(0).GetSlotNum();
uint16_t slotDuration = 500 + slotDuraitonCount * 120;
double totalRAWSlotTime = (slotDuration * beaconCount) / 1000000;


ofstream rawThrougputFile(config.name+"-raw-th.txt", ios::app);
for (uint16_t g = 0; g < totalRawGroups; g++)
{
cout << "Byte receive = " << rawGroupsStats[g].totalByteReceived << endl;
cout << "Packets receive = " << rawGroupsStats[g].numberOfReceivedPackets << endl;
double rawGroupThroughput = (double) rawGroupsStats[g].totalByteReceived * 8 /
  (totalRAWSlotTime * numRAWSlots * 1000000);
cout << "RAW " << g + 1 << " THROUGHPUT: " << rawGroupThroughput << " Mbps" << endl;
rawThrougputFile << rawGroupThroughput * 1000 << ",";
}
rawThrougputFile << endl;
rawThrougputFile.close();

std::vector<double> zonesThroughput = {0.0, 0.0, 0.0, 0.0};
double sum1 = 0, sum2 = 0;
double squareSum1 = 0, squareSum2 = 0;

long double avgDelay = 0;

uint16_t zone_idx = 0;
for (auto zone : topology.m_zones)
{
for (auto i: zone->m_nodes)
{
risultati << i << spazio << dist[i] << spazio <<timeRxArray[i].GetSeconds() << ",
 ("<< timeRxNotAssociated[i].GetSeconds() << "),
 " << timeIdleArray[i].GetSeconds() << ",
 (" << timeIdleNotAssociated[i].GetSeconds() << ")," << timeTxArray[i].GetSeconds() << ",(" << timeT
long nOfSuccessfulPackets = stats.get(i).NumberOfSuccessfulPackets;
long payLoadSize = stats.get(i).PayLoadSize;
long double delay = stats.get(i).getAveragePacketSentReceiveTime();
Time txTime = stats.get(i).TotalTransmitTime;
double staThroughput = (nOfSuccessfulPackets * payLoadSize * 8) / (totalRAWSlotTime * 1000000.0);
sum1 += staThroughput;
squareSum1 += (staThroughput * staThroughput);

zonesThroughput[zone_idx] += staThroughput / numRAWSlots;

long nOfTransmissions = stats.get(i).NumberOfTransmissions;
double datarate = stats.get(i).DataRate;

double normThroughput = staThroughput / datarate;
sum2 += normThroughput;
squareSum2 += (normThroughput*normThroughput);

avgDelay = (delay + (avgDelay * i)) / (i + 1);

cout << "Station "<< i + 1 <<":"<<endl;
cout << " Throughput = "<< staThroughput * 1000 << " Kbit/s" <<endl;
cout << "Tx Time = " << txTime.GetSeconds() << std::endl;
cout << "Datarate = " << datarate * 1000 << " Kbit/s" << std::endl;
cout << "Payload Size = " << stats.get(i).PayLoadSize << " bytes" << std::endl;
cout << "Throughput / RX rate = " << normThroughput << std::endl;
cout << "Delay = " << delay << std::endl;
cout << std::endl;

throughputFile << staThroughput * 1000 << ",";
}
zone_idx += 1;
}

throughputFile << endl;
throughputFile.close();

delayFile << avgDelay << endl;
delayFile.close();

ofstream zonesthFile(config.name+"-zone-th.txt", ios::app);
ofstream fairnessFile(config.name+"-fairness.txt", ios::app);
ofstream normFairnessFile(config.name+"-norm-fairness.txt", ios::app);

for (uint32_t k = 0; k < zonesThroughput.size(); k ++)
{
zonesthFile << zonesThroughput[k] * 1000 << ",";
}
zonesthFile << endl;
zonesthFile.close();

double fairness1 = (sum1*sum1) / (config.Nsta * squareSum1);
double fairness2 = (sum2*sum2) / (config.Nsta * squareSum2);
std::cout << "Jain’s Fairness Index = " << fairness1 << endl;
std::cout << "Jain’s Fairness Index (normalized )= " << fairness2 << endl;

fairnessFile << fairness1 << endl;
normFairnessFile << fairness2 << endl;

fairnessFile.close();
normFairnessFile.close();
risultati.close();



uint64_t rngRun = RngSeedManager::GetRun();
ofstream kmeansFile("kmeans/"+config.name+"-"+to_string(rngRun)+".txt");

for (auto staParams : macToStaParams)
{
kmeansFile << staParams.second.packetSize << ",";
kmeansFile << staParams.second.rate << ",";
kmeansFile << staParams.second.signalDbm << ",";
kmeansFile << staParams.second.rawGroup << endl;
}
kmeansFile.close();

return 0;
}
```

Configuration.h

```
#pragma once
```

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/internet-module.h"
#include "ns3/extension-headers.h"
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <ctime>
#include <fstream>
#include <sys/stat.h>
#include <string>

using namespace ns3;
using namespace std;

struct Configuration {
/*
 * New configuration parameters
 *
 * */

RPSVector rps;
uint32_t nRps; // Ordinal number of current RPS element; RPS Index
uint64_t totalRawSlots = 0;
 // Total number of RAW slots in all RAW groups in all RPS elements
std::string RawConfigString;
 // RPS=2;{RAW=2;[0,1,1,204,2,0,1,16][0,1,1,412,1,0,17,32]}{RAW=1;[0,1,1,180,3,0,33,35]}

UintegerValue maxNumberOfPackets = 4294967295u; // 4294967295u
string trafficType = "udp";
 // important - udp is considered to be only uplink in NodeStatistics::GetPacketLoss tcpipcamera
 tcpfirmware

// Page slicing
pageSlice pageS;
TIM tim;

/*pageSliceCount = 0 means:
 *
 *  - if pageSliceLength > 1, 32nd TIM in this DTIM can contain DL information for STAs that do not
  support
 *    page slicing and for STAs who's AID is within the 32nd block of this page and do support page
 *
 *  - if pageSliceLength = 1, all STAs for which the AP has DL BU are included in the *only TIM*
  that is scheduled
 *    within the DTIM
 *
 * */

uint32_t pagePeriod = 1;
 // Number of Beacon Intervals between DTIM beacons that carry Page Slice element
 for the associated page
uint8_t pageIndex = 0;
```

```
 // The Page Index subfield indicates the page whose slices are served during beacon intervals
 within a page period
uint32_t pageSliceLength = 4;
 // Number of blocks in each TIM for the associated page except for the last TIM (1-31)
 (value 0 is reserved);
// The number of blocks in each page slice is equal to the value of the Page Slice Length subfield
uint32_t pageSliceCount = 1;
 // Number of TIMs in a single page period (1-31)
uint8_t blockOffset = 0;
 // The 1st page slice starts with the block with blockOffset number
uint8_t timOffset = 0;
 // Offset in number of Beacon Intervals from the DTIM that carries the first page slice of the
 page
/*
 * Common configuration parameters
 * */
double simulationTime = 120; // in seconds
uint32_t seed = 1;
int NRawSta;
uint32_t Nsta;
uint32_t BeaconInterval = 199840; // In microseconds

double bandWidth = 2; // in MHz
string rho="200";    // Maximum distance of the AP

string visualizerIP = "localhost"; // empty string if no visualization TODO
int visualizerPort = 7707;
double visualizerSamplingInterval = 1;

string RawGroupping = "kmeans"; // 'kmeans', 'random', 'static'
string name = "kmeans"; //
string APPcapFile = "appcap"; // empty string if no visualization TODO
string NSSFile = "test.nss";

/*
 * Le's config params
 * */
// uint32_t payloadSize = 1400;
// string folder="./scratch/";
// string file="./scratch/mac-sta.txt";
string TrafficPath="./OptimalRawGroup/traffic/data-32-0.82.txt";
bool S1g1MfieldEnabled=false;
string RAWConfigFile = "./OptimalRawGroup/RawConfig-test.txt";
string DataMode = "MCS2_0";
string OutputPath = "./OptimalRawGroup/";
string topologyFile = "./topology/zones4-sta20-uniform.txt";
/*
 * Amina's configuration parameters
 * */
bool useV6 = false; //false
uint32_t nControlLoops = 0;//  = 100;
uint32_t coapPayloadSize = 0;//  = 15;

uint32_t trafficInterval = 2; //ms 55,110,210,310,410,515,615,720,820,950,1024 beacon interval *4
uint32_t trafficIntervalDeviation = 1000; //1000 discuss with Jeroen

int SlotFormat=0; //0;
int NRawSlotCount=0; //162;
uint32_t NRawSlotNum=0;
```

```
uint32_t NGroup=0;

/*
 * tcpipcamera configuration parameters
 * */
double ipcameraMotionPercentage = 1; //0.1
uint16_t ipcameraMotionDuration = 10; //60
uint16_t ipcameraDataRate = 128; //20
uint32_t MinRTO = 81920000; //819200
uint32_t TCPConnectionTimeout = 6000000;
uint32_t TCPSegmentSize  = 3216; //536
uint32_t TCPInitialSlowStartThreshold = 0xffff;
uint32_t TCPInitialCwnd = 1;

int ContentionPerRAWSlot=0; //-1
bool ContentionPerRAWSlotOnlyInFirstGroup=false; //false

double propagationLossExponent = 3.67; //3.76
double propagationLossReferenceLoss = 8;

bool APAlwaysSchedulesForNextSlot = false;
uint32_t APScheduleTransmissionForNextSlotIfLessThan = 0;// = 5000;


uint32_t firmwareSize = 0;// = 1024 * 500;
uint16_t firmwareBlockSize = 0;// = 1024;
double firmwareNewUpdateProbability;// = 0.01;
double firmwareCorruptionProbability;// = 0.01;
uint32_t firmwareVersionCheckInterval;// = 1000;

uint16_t sensorMeasurementSize;// = 54; //1024

uint16_t MaxTimeOfPacketsInQueue = 100; //100

uint16_t CoolDownPeriod = 4; //60

Configuration();
Configuration(int argc, char *argv[]);

};
```

APÊNDICE B

MATLAB

```
% model with channel error 2 slots per RAW group
% The stations are arranged in rings of 50/100/150/200 meters
%Uniform case
clear all;
close all;
% Definio de variveis
basic_rate = 1000000; %bits/s

data_rate_0 = 650000; % bits/s
%data_rate_1 = 1300000; % bits/s
%data_rate_2 = 1950000; % bits/s
%data_rate_3 = 2600000; % bits/s

phy_header = 0.000192; %in seconds, ah parameter
%h = 416/basic_rate; % (416 = 224 bits cabealho MAC + 192 bits cabealho
%PHY)em segundos, cabealho antigo
%ack = 304/basic_rate; % (304= 112 bits + 192 bits phy) seconds
mac_header = (34*8)/basic_rate;
h = phy_header + mac_header;
ack = (14*8)/basic_rate + phy_header;
e = 1024*8; % payload size/bitrate
%e2=768*8;
%e3 = 512*8;
%e4 = 256*8;

e_duracao = e/data_rate_3;
%e_duracao_2 = e2/data_rate_0;
%e_duracao_3 = e3/data_rate_0;
%e_duracao_4 = e4/data_rate_0;

%sifs = 0.000160; % seconds
%difs = 0.000303; % seconds
%delta = 0.000006; % seconds
%sigma = 0.000052; % seconds
%ack_timeout = 0.000600;
sifs = 0.000160; % seconds
difs = 0.000304; % seconds
delta = 0.0000033; % seconds
sigma = 0.000052; % seconds
ack_timeout = 2*delta + sifs + ack;
beacon_interval = 0.19984; % seconds

x0 = [0.01;0.01]; % valor inicial para fsolve
% time occupied by a successful transmission
Ts1 = h + e_duracao + sifs + (2.*delta) + ack + difs;
Ts2 = h + e_duracao + sifs + (2.*delta) + ack + difs;
Ts3 = h + e_duracao + sifs + (2.*delta) + ack + difs;
Ts4 = h + e_duracao + sifs + (2.*delta) + ack + difs;
% time occupied by a collision
Tc1 = h + e_duracao + difs + delta + ack_timeout;
Tc2 = h + e_duracao + difs + delta + ack_timeout;
```

```
Tc3 = h + e_duracao + difs + delta + ack_timeout;
Tc4 = h + e_duracao + difs + delta + ack_timeout;
% guard period and holding period
Tg = 0.000008;
Th1 = Ts1 + 2*Tg;
Th2 = Ts2 + 2*Tg;
Th3 = Ts3 + 2*Tg;
Th4 = Ts4 + 2*Tg;

n = 10;
nslots = 2;

Tidle_2_R1 = (beacon_interval/8)-Th1;
Tidle_2_R2 = (beacon_interval/8)-Th2;
Tidle_2_R3 = (beacon_interval/8)-Th3;
Tidle_2_R4 = (beacon_interval/8)-Th4

prob_erro_50m;
prob_erro_100m;
prob_erro_150m;
prob_erro_200m;

% 10 stations within a RAW slot, each group have 2 RAW slots
% for 50m R1
fun1 = @solucao_10sta_50m;
result1 = fsolve(fun1,x0);
taud1 = result1(1);
b0_01 = result1(2);


% probabilities
ptr1 = 1 - ((1-taud1)^n);
pds1 = ((n*taud1*((1-taud1)^(n-1)))/ptr1)*(1-perr50);

% success probabilities
t_slot1 = ((1 - ptr1).*sigma + ptr1.*pds1.*Ts1 + ptr1.*(1 - pds1).*Tc1);
s1 = (pds1.*ptr1.*e./t_slot1);

% for 100m R2
fun2 = @solucao_10sta_100m;
result2 = fsolve(fun2,x0)
taud2 = result2(1);
b0_02 =result2(2);


ptr2 = 1 - ((1-taud2)^n);
pds2 = ((n*taud2*((1-taud2)^(n-1)))/ptr2)*(1-perr100);
t_slot2 = ((1 - ptr2).*sigma + ptr2.*pds2.*Ts2 + ptr2.*(1 - pds2).*Tc2);
s2 = (pds2.*ptr2.*e./t_slot2);

% for 150m R3
fun3 = @solucao_10sta_150m;
result3 = fsolve(fun3,x0);
taud3 = result3(1)
b0_03 = result3(2)


ptr3 = 1 - ((1-taud3)^n);
pds3 = ((n*taud3*((1-taud3)^(n-1)))/ptr3)*(1-perr150);
```

```
t_slot3 = ((1 - ptr3).*sigma + ptr3.*pds3.*Ts3 + ptr3.*(1 - pds3).*Tc3);
s3 = pds3.*ptr3.*e./t_slot3;

% for 200m R4
fun4 = @solucao_10sta_200m;
result4 = fsolve(fun4,x0);
taud4 = result4(1);
b0_04 = result4(2);

ptr4 = 1 - ((1-taud4)^n);
pds4 = ((n*taud4*((1-taud4)^(n-1)))/ptr4)*(1-perr200);
t_slot4 = ((1 - ptr4).*sigma + ptr4.*pds4.*Ts4 + ptr4.*(1 - pds4).*Tc4);
s4 = pds4.*ptr4.*e./t_slot4;



vazao_R1 =(((Tidle_2_R1)/beacon_interval)/100000)*s1*nslots
vazao_R2 =(((Tidle_2_R2)/beacon_interval)/100000)*s2*nslots
vazao_R3 =(((Tidle_2_R3)/beacon_interval)/100000)*s3*nslots
vazao_R4 =(((Tidle_2_R4)/beacon_interval)/100000)*s4*nslots
```

prob_erro_50.m

```
% Error probability 50 m
clc;
d = 50; %meters
f = 900; % MHz
Pt = 0;%10*log10(0.001/0.001); %  dB
Gt = 0; % dB
Gr = 3; %dB
N = 6.8; %dB
R = 0.5;%650000; %bits/s
B = 2000000; %Hz
L = 256*8; %bits
dfree = 10;
adfree = 11;
%path loss
pl = 8 + 36.7*log10(d)% + 21*log10(f/900) %dB
%Received Power
Pr = Pt + Gt + Gr - pl %dB
pr_lin = 10.^(Pr/10)
N_lin = 10.^(N/10)
% cálculo de SNR
snr = pr_lin/(N_lin*B) %dB
% cálculo de Eb/N0
EbNodB= snr*(B/R) %dB
EbNolin=10.^(EbNodB/10)
% BER BPSK
Berb = 0.5*erfc(sqrt(EbNolin))
%BER M-QAM
M = 16
k = log2(M)
if(M==4)
    a=1;
else
    a=4/log2(M);
end
b=3*log2(M)/(M-1);
```

```
Ber= [0.5*a*(1-sqrt(0.5*b*EbNolin/(1+0.5*b*EbNolin)))]

%k=log2(m)

% rayleigh channel
Ber1 = (4/k)*erfc(sqrt(3*EbNolin/(M-1)))

%viterbi decoder
dstart = floor((dfree+1)/2)
dend = dfree
pd = 0
if (mod(d,2) == 1), %odd

  for i = dstart:1:dend
    pd = pd + nchoosek(dfree,i)*(Berb^i)*((1-Berb)^(dfree-i))
  endfor;

else
for i = dstart:1:dend
 pd = pd + 0.5*nchoosek(dfree,i)*(Berb^i)*((1-Berb)^(dfree-i))
 endfor
endif;


pu = adfree*pd
pmu = min(pu,1)
perr50 = 1 - (1 - pmu)^L

solucao_10sta_50m.m

% Function with 10 stations in a Raw slot
% error probabilty for 50 m

function F = solucao_10sta_50m(x)
taud = x(1);
b0_0 = x(2);

n = 10;%number of stations in each group
m = 6;%maximum retransmissions

%contention window in each backoff stage
w0 = 16;
w1 = 2*w0;
w2 = 4*w0;
w3 = 8*w0;
w4 = 16*w0;
w5 = 32*w0;
w6 = 64*w0;

prob_q_20sta;
prob_erro_50m;

pcol = 1 - ((1-taud)^(n-1));
g = pcol;
p = pcol - pcol*perr50 + perr50;

%vectors bi,j
b0_j = zeros(1,w0);
```

```
b1_j = zeros(1,w1);
b2_j = zeros(1,w2);
b3_j = zeros(1,w3);
b4_j = zeros(1,w4);
b5_j = zeros(1,w5);
b6_j = zeros(1,w6);


% states bi,j
% N
n_linha0 = (1 - g*(1-q0));
n_linha1 = (1 - g*(1-q1));
n_linha2 = (1 - g*(1-q2));
n_linha3 = (1 - g*(1-q3));
n_linha4 = (1 - g*(1-q4));
n_linha5 = (1 - g*(1-q5));
n_linha6 = (1 - g*(1-q6));

 b1_j(w1)= (p*(1-q0))/(w1*n_linha1).*b0_0;
b1_j_rev = flip(b1_j);
for j = 1:1:w1-1

    b1_j_rev(j+1) = (p*(1-q0))/(w1*n_linha1).*b0_0 + (((1-q1)*(1-g))/n_linha1).*b1_j_rev(j);

end
b1_j_rev(w1) =  (p*(1-q0)/w1).*b0_0 + (((1-q1)*(1-g))/n_linha1).*b1_j_rev(w1-1);
b1_0 = b1_j_rev(w1);

b2_j(w2)= (p*(1-q1))/(w2*n_linha2).*b1_0;
b2_j_rev = flip(b2_j);
for j = 1:1:w2-1

    b2_j_rev(j+1) = (p*(1-q1)/(w2*n_linha2)).*b1_0 + (((1-q2)*(1-g))/n_linha2).*b2_j_rev(j);

end
b2_j_rev(w2) = ((p*(1-q1))/w2).*b1_0 + (((1-q2)*(1-g))/n_linha2).*b2_j_rev(w2-1);
b2_0 = b2_j_rev(w2);

b3_j(w3)= (p*(1-q2))/(w3*n_linha3).*b2_0;
b3_j_rev = flip(b3_j);
for j = 1:1:w3-1

    b3_j_rev(j+1) = (p*(1-q2))/(w3*n_linha3).*b2_0 + (((1-q3)*(1-g))/n_linha3).*b3_j_rev(j);

end
b3_j_rev(w3) = ((p*(1-q2))/w3).*b2_0 + (((1-q3)*(1-g))/n_linha3).*b3_j_rev(w3-1);
b3_0 = b3_j_rev(w3);

b4_j(w4)= (p*(1-q3))/(w4*n_linha4).*b3_0;
b4_j_rev = flip(b4_j);
for j = 1:1:w4-1

    b4_j_rev(j+1) = (p*(1-q3))/(w4*n_linha4).*b3_0 + (((1-q4)*(1-g))/n_linha4).*b4_j_rev(j);

end
b4_j_rev(w4) = (p*(1-q3)/w4).*b3_0 + (((1-q4)*(1-g))/n_linha4).*b4_j_rev(w4-1);
b4_0 = b4_j_rev(w4);

b5_j(w5)= (p*(1-q4))/(w5*n_linha5).*b4_0;
```

```
b5_j_rev = flip(b5_j);
for j = 1:1:w5-1

    b5_j_rev(j+1) = (p*(1-q4))/(w5*n_linha5).*b4_0 + (((1-q5)*(1-g))/n_linha5).*b5_j_rev(j);

end
b5_j_rev(w5) = (p*(1-q4)/w5).*b4_0 + (((1-q5)*(1-g))/n_linha5).*b5_j_rev(w5-1);
b5_0 = b5_j_rev(w5);

b6_j(w6)= (p*(1-q5))/(w6*n_linha6).*b5_0;
b6_j_rev = flip(b6_j);
for j = 1:1:w6-1

    b6_j_rev(j+1) = (p*(1-q5))/(w6*n_linha6).*b5_0 + (((1-q6)*(1-g))/n_linha6).*b6_j_rev(j);

end
b6_j_rev(w6) = (p*(1-q5)/w6).*b5_0 + (((1-q6)*(1-g))/n_linha6).*b6_j_rev(w6-1);
b6_0 = b6_j_rev(w6);


%para o calculo de b0,0
soma_bi_0 = b0_0 + b1_0 + b2_0 + b3_0 + b4_0 + b5_0 + b6_0;
soma_bi_01 = (1-q0).*b0_0 + (1-q1).*b1_0 + (1-q2).*b2_0 + (1-q3).*b3_0 + (1-q4).*b4_0 + (1-q5).*b5_0
soma_dados1 = q1.*sum(b1_j_rev)+ q2.*sum(b2_j_rev)+ q3.*sum(b3_j_rev)+ q4.*sum(b4_j_rev)+ q5.*sum(b5
b_idle = soma_dados1;

m_linha = (1-p)*soma_bi_01 + b_idle;

b0_j_rev = flip(b0_j);

b0_j_rev(1)= m_linha./(w0*n_linha0);% termo bo,w0-1,0
for j = 1:1:w0-1

    b0_j_rev(j+1)= m_linha/(w0*n_linha0) + (((1-q0)*(1-g))/w0*n_linha0).*b0_j_rev(j);

end
 b0_j_rev(w0)= m_linha/w0 + (((1-q0)*(1-g))/w0*n_linha0).*b0_j_rev(w0-1);
 b0_0 = b0_j_rev(w0);

soma_dados = sum(b0_j_rev)+ sum(b1_j_rev)+ sum(b2_j_rev)+ sum(b3_j_rev)+ sum(b4_j_rev)+ sum(b5_j_rev

 F(1)= soma_bi_0 - taud;
 F(2)= soma_dados-1;

end


prob_q_10sta.m

m = 6;
constante_c;
n = 10;
fator_n10 = (n-1)/n;

q0 = fator_n10*c*0;
q1 = fator_n10*c*1/(m+1);
q2 = fator_n10*c*2/(m+1);
q3 = fator_n10*c*3/(m+1);
q4 = fator_n10*c*4/(m+1);
```

```
q5 = fator_n10*c*5/(m+1);
q6 = fator_n10*c*6/(m+1);
```

constante_c.m

```
beacon_interval = 0.19984;
nslots = 2;
Tidle = (beacon_interval/nslots);

razao_nslots = (beacon_interval-Tidle)/beacon_interval;
c = razao_nslots;
```