1-1-2020

Simulation Analysis Of Bluetooth Scanning Systems

George Humphrey

SIMULATION ANALYSIS OF BLUETOOTH SCANNING SYSTEMS

A Thesis
presented in partial fulfillment of requirements
for the degree of Master of Science in Engineering Science
in the Department of Electrical Engineering
The University of Mississippi

by

George Humphrey II

May 2020

ABSTRACT

We develop and present a simulation program to investigate the failure probability of a scanning job conducted by a Bluetooth Low-Energy scanner on a group of items. Each item is outfitted with a BLE device and are sorted into large, batch sizes to be moved into the scanning area. The BLE device broadcasts packets, known as advertisements, to the scanning system. The rate at which the advertisements broadcast is determined by its interadvertisement time, which is the sum between a fixed advertising interval and a pseudo-random advertising delay. The objective is to determine the optimal interadvertisement time that minimizes the scanning time while achieving a prescribed minimal probability of failure to successfully scan every advertiser. Using our analytical model to set the parameters for our simulations, we notice that the simulation results had an order of magnitude difference from the analytical results. Additionally, the simulation results failed to meet our objective. Thus, we retraced our steps and hypothesized that the events of collisions were correlated. We adjusted both the analytical and simulation model by implementing a discrete-Markov chain. This allowed us to explore successive collision events between the target advertiser and the collection of other advertisers. We found that the probability of successive collisions increases as the number of successive collisions increases. Thus, correlated collisions have a profound impact on the failure probability and the probability of collision between the target advertiser and any other advertiser on successive advertisements is independent of the interadvertisement time. Additionally, we show that longer scanning times are needed but failure probability of a scanning job is not excessively sensitive to interadvertisement time.

ACKNOWLEDGEMENTS

First, I would like to thank the faculty and staff of the University of Mississippi's Department of Electrical Engineering for their guidance and teaching throughout my student career. Next, I would like to express my humble gratitude to my advisor, Dr. John N. Daigle. Along with my colleagues, family and friends, he has provided me with tremendous encouragement throughout this process and the tools needed to continue my life-long journey of learning. I gratefully acknowledge all of my friends and colleagues who has been there to support me throughout my student career. Last but not the least, I am grateful to my family for their undying love, support, and encouragement.

University, Mississippi George A. Humphrey II

May 2020

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

We develop and present a simulation program to investigate the failure probability of a scanning job conducted by a Bluetooth Low-Energy (BLE) scanner on a group of items. These items are sorted into large batch sizes and moved to a scanning area. Each member of a group is outfitted with a BLE device that broadcasts packets, known as advertisements, to the system. The rate at which the advertisements broadcast is determined by its interadvertisement time, which is the the sum between a fixed advertising interval and a pseudo-random advertising delay as described in [1] and discussed here. If the system fails to successfully scan any item in a group, that operation is marked as a failure. For this thesis, we ask how to set the interadvertisement time that minimizes the scanning time while achieving a prescribed minimal probability of failure to successfully scan every advertiser.

The rest of the thesis is organized as follows. The remainder of this chapter describes work related to this thesis. In Chapter 2, we present a general introduction to BLE as well as insight into the advertising and scanning operations. Chapter 3 gives a description of the simulation program along with the methodology for finding the interadvertisement time and scanning time given the assumption of independent collisions from our analytical model. Additionally, the simulation results are discussed and analyzed in this chapter. In chapter 4, we explore correlation in the collision process given the results from the previous chapters. Finally, chapter 5 gives the conclusions based on our results. We summarize the content of the thesis, and discuss the accomplishments and contributions of our work. Recommendations for further research in this topic are also included in this chapter.

1.1 Related Work

Though most state-of-the-art BLE research focuses on the neighbor discovery process (NDP) and the discovery latency, the systems under study all involve numerous BLE devices in a crowded environment. Shan et al. [2] developed a Java simulation program to analyze the relationship between the number of advertisers and the total scan time to discover all advertisers. To compare their simulation results for accuracy, they build an experimental testbed of 40 iBeacons as advertisers and one Raspberry pi as a scanner. Aside from their main objective, they also tested for average waiting time for a successful scan with different number of advertisers and the relationship between a number of advertisers and the number of received complete signal until all advertisers discovered. With the simulation and experimental results only showed minor differences, they concluded the simulation program was accurate enough to simulate in advertisement collision in a crowded environment.

In [3], Shan et al. extended their previous work with an analytical model that minimizes the discovery time for all BLE advertisers in a given size by optimizing the advertising interval. To find the optimal advertising interval, they utilized a differential coefficient process. In their results, they found that for a given advertising interval as the number of advertisers increase so does the expected discovery time to scan for all surrounding advertisers. However, there is an optimal advertising interval that minimizes the discovery time for each number of advertises that varies. They also show how energy consumption and the scan interval effect the discovery time as well. In both works, their system consisted of one scanner and numerous advertisers, similar to our system. However, we provide a specific formula to compute the optimal interadvertisement time, which is shown in a later chapter of this work. Unlike [2] and [3], we take an in-depth approach to understanding when a scanner fails to scan all advertisers and analytically expressed it as the probability of a scanning job failure ($P\{\mathrm{SJF}\}$). Additionally, we investigate the reason the assumption of independent collisions fails to produce a reasonable approximation of the probability of scanning job failure. Furthermore, we explain the simulation program in detail, including how we obtain and set our parameters.

A performance model was derived in [4] to analyze the average discovery time by adjusting scanning and advertisement intervals. The model assumed scanning in discontinuous mode, which will be discussed in the next chapter. In their analysis, they revealed a coupling procedure that could lower the efficiency of the device discovery if the advertising interval is close to or equal the initiating interval. The advertising delay can play a decoupling role. However, they found that it is not sufficiently large enough to decouple advertisers with colliding advertisements. Thus, their solution was to modify the advertisement interval to mitigate long discovery latency from the scanners. They created three strategies Fast Shrink, Threshold-based Shrink, and Periodic-based Shrink. In their results, the Fast Shrink strategy attained the lowest latency. The other two strategies were able to obtain slightly higher latencies than the Fast shrink. However, unlike Fast shrink, users are able to adjust the other strategies to accommodate variances in attributes such as node density.

In [5], the same authors from [4] developed an analytical BLE model derived from a pure ALOHA system model. This model investigates device discovery and connection setup latency in wireless body area networks with the scanner in both continuous and discontinuous mode. They showed that as the number of advertisers increases so does the average discovery latency for an advertiser in the network. As a solution, they created a connection setup report that piggybacks on the connection response packet from the advertiser to the nearest scanner. From there, the scanner can use the report to adaptively adjust its parameters mitigating the interference and reducing latency. By validating their model with extensive simulations, they were able to see reductions in latency as low as 68% in the discontinuous scanning scenario.

In [6], Ghamari et al. developed an analytical model that investigates the probability of packet collisions amongst BLE advertisers. Assuming each advertiser's advertisements arrive according to a Poisson process, they demonstrate that that the probability of collisions is dependent on the number of advertisers and the advertising interval. However, advertisements in a real system do not occur according to a Poisson process as stated above. Kim et al. develops a backoff scheme in [7] to reduce collisions among scanners responding to an advertisement. Unlike the previous proposals, this paper focuses on one advertiser and numerous scanners. In their system under study, they have an advertiser broadcast an advertisement to multiple scanners within the vicinity. As multiple scanners send their individual responses to the advertiser, collisions between the response packets can ensue. Their solution is to develop a backoff window for the scanners in responding to an advertisement. They discover that a longer backoff window does reduce the collision probability. However, there is no significant improvement in its reduction when the window exceeds a certain value. Instead of focusing on the 3 advertising channels in BLE, Kalaa et al. chose to focus on the remaining 37 data channels in [8]. They derived the probability of selecting a data channel once a BLE pair of devices connects. Then they used a channel selection algorithm to find the probability of collision among multiple BLE connections and the maximum achievable aggregate throughput.

For a more practical solution, Julien et al. [9] developed a protocol that takes into account the real effects of packet collisions and adjust BLE parameters to minimize discovery latency. Their protocol is able to do uni-directional and bi-directional neighbor device discovery. They also implemented a way for the advertisers to determine which advertiser has been discovered through stored information in the advertisements. In their real-world results, they find their discovery latency curves match closely with their simulations. Furthermore, they demonstrate the protocol's adaptiveness in adjusting the amount of advertisements sent as the density of advertisers increase.

On the topic of BLE, these are the very few studies that relates to our own but none of them directly addressed the specific question in this thesis. Though every article presented computes the probability of collisions, they do not address the probability of failing to detecting all advertisers. Additionally, their simulation results lack in-depth analysis and confidence intervals to show the strength of their findings.

1.2 Thesis Summary

Here we investigate a BLE scanning system where a large number of items are sorted into groups to be scanned. The objective is to determine the optimal interadvertisement time that minimizes the scanning time while achieving a prescribed minimal probability of failure to successfully scan every advertiser. Using the BLE Core Specification 5.0, we developed a Python simulation program that replicates the functionality of the BLE advertising and scanning process. Using our analytical model to set the parameters for our simulations, we notice inconsistencies between the their results. Thus, we retraced our steps to see if the events of collisions were correlated. More methods were added to the simulation to investigate the nature of dependence. We discovered that the correlated collisions impact the failure probability and that the probability of collision between the target advertiser and any other advertiser on successive advertisements is independent of the interadvertisement time. Additionally, we show that longer scanning times are needed but failure probability of a scanning job is not excessively sensitive to interadvertisement time.

CHAPTER 2

BLUETOOTH LOW-ENERGY

In this chapter, we will delve into the BLE protocol, specifically from the Bluetooth 5.0 core specification. We will not fully explore the protocol, as most of it does not pertain to this body of work. The primary focus is the introduction of the scanning and advertising process as well as a bit about the packet format. The chapter will explain these two processes as it applies to this particular study.

2.1 Overview of BLE

Designed by the Bluetooth Special Interest Group, the BLE protocol debuted in June 2010 with version 4.0 of the Bluetooth core specification. The objective of BLE was to provide the communication range of classic Bluetooth to power-constrained devices. Like classic Bluetooth, BLE utilizes the 2.4 GHz ISM band for wireless communication. Yet, modifications were made in the PHY and Link layers of the Bluetooth core to differentiate between the two. At the time of this thesis, the current adoption is Bluetooth 5.1. However, we will be using Bluetooth 5.0 as a reference.

For the PHY layer, the BLE protocol uses Gaussian frequency-shift keying modulation, which shifts the carrier frequency to carry the modulation and filters it through a filter with a Gaussian response curve, similar to classic Bluetooth. The modulation scheme has options between coded and uncoded data. The coded data is only available at the default symbol rate of 1 Msym/s. However, the uncoded data has an additional option of 2 Msym/s. The type of packet from the link layer will determine which symbol rate is used. Starting from 2402 MHz to 2480 MHz, the spectrum is allocated into 40 channels with 2 MHz spacing, instead of 79 channels with 1 MHz spacing. Using these channels, the PHY layer is separated into 3 distinct channel operations: advertising, data, and periodic.

In advertising mode, all channels are used for broadcasting data, device discovery, and connection initiation [1]. Out of the 40 channels, channels 2402, 2426, and 2480 MHz, indexed as 37, 38, 39 respectively, are known as primary advertising channels [10]. These channels are use to initiate broadcasting of data. The other 37 channels are secondary advertising channels, which are used in maintaining communications after initial advertising. Furthermore, these same channels are used in periodic and data modes of the PHY layer.

The PHY layer in data mode establishes and maintains connections between BLE devices. The periodic mode is similar to the advertising mode in the PHY layer. However unlike the advertising mode, where the packet data are fixed, the periodic mode has packet data vary from time to time [1]. To avoid interference in the crowded 2.4 spectrum, BLE uses a Time division duplex scheme to enable a frequency hopping mechanism akin to classic Bluetooth. However, the timing between hops is more relaxed [4], allowing for lower power consumption. For the purposes of our research, we only focused on the advertising channel operation. Furthermore, our main objective concerns the primary advertising channels, which were mentioned above.

For the link layer (MAC sub-layer), the system operates in terms of a state machine with the following states shown in Fig 2.1. While there can be multiple instances of the state machine, only one state can be active at any time. BLE devices always start in the standby state, where packets are neither received or transmitted. If a device goes to the advertising state, the BLE device is denoted as an advertiser and broadcasts packets on advertising channels. The scanning and initiating states are similar, except the device in the initiating state tries to establish a connection with the advertising device. As seen from Fig 2.1, the connection state is when two devices are connected, notably one device in the advertising and another in the initiating state, and cannot be entered from the standby state.

There are a variety of packets for the different states, excluding the standby state. Given which symbol rate is used on the PHY layer, packets can be transmitted between 44

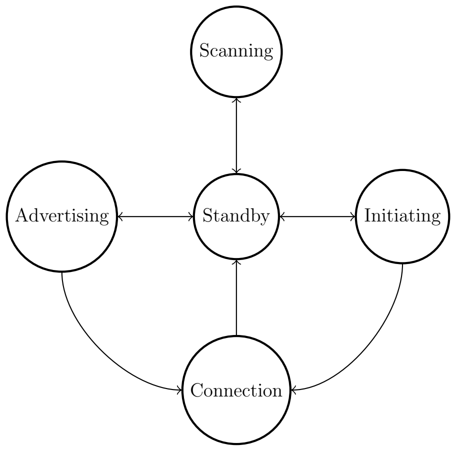


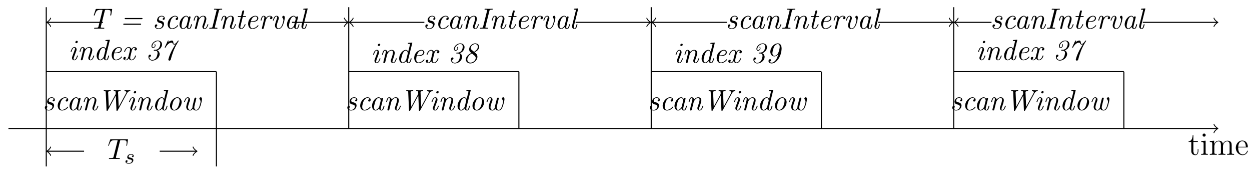
Figure 2.1: BLE Link layer state machine

to 2140 μs. Given our problem description, we are not concerned with connecting devices. Thus, our state machines will not be relying on the connection and initiating state. Moreover, we are using a particular packet format for our advertisements, which will be defined later in this chapter [1].

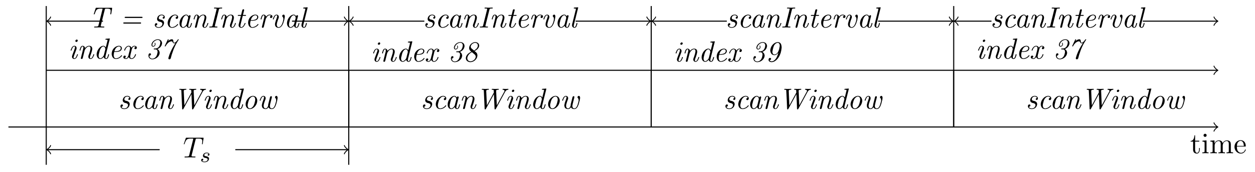
2.2 Scanning Process

In the scanning state, the BLE device, known as a scanner, listens on each advertising channel for BLE packets or advertisements in a round-robin fashion. The scanner will perform this operation for the duration of its scanning window, T_s. The period between two consecutive scanning windows is called the scan interval, T. Since we are referencing Bluetooth 5.0 for this thesis, both the scanning window and scanning interval can be set by a user in the range of 0 s to 40.96 s. However, the scanning window has to be less than or equal to the scanning interval.

Figure 2.2 illustrates this process. Part (a) demonstrates the case of $T_\mathrm{s} \leq T$ or discontinuous scanning. Generally, this case is found in most real-world applications, as it allows a sleep/wake-up cycle for low-energy consumption. For this thesis, we are trying to find the minimal scan time. Therefore, a sleep/wake-up cycle is not useful to our study. We need the scanner to listen continuously, which is when $T_\mathrm{s} = T$, as shown in Part (b).



(a) Discontinuous scanning mode



(b) Continuous scanning mode

Figure 2.2: Illustrations of scanning windows and scanning intervals

Given that there will be no response from the scanner, we will be passively scanning for advertisements.

2.3 Advertising Process

In the advertising state, the BLE device, known as an advertiser, broadcasts advertisements on each advertising channel during its advertising event, which occurs every interadvertisement time, $\tilde{t}_{\mathrm{IA}}$. The interadvertisement time is the sum between an advertising interval ($\tilde{t}_{\mathrm{AI}}$) and the advertising delay ($\tilde{t}_{\mathrm{AD}}$). Figure 2.3 shows the advertising process, but not to scale. The advertising event starts at the beginning of the advertising interval, which is a large fixed period set by the user. The time used in $\tilde{t}_{\mathrm{AI}}$ can be between 20 ms and 10,485.759375 s, but has to be a multiple of 0.625 ms [1]. During that time, one advertisement is transmitted on each of the advertising channels. The first advertisement is sent on channel index 37, while the last one sent on channel index 39. The transmission time of an advertisement is denoted as T_{A}. The time between two consecutive advertisement transmissions is the dead time, which can be between 0 to 10 ms.

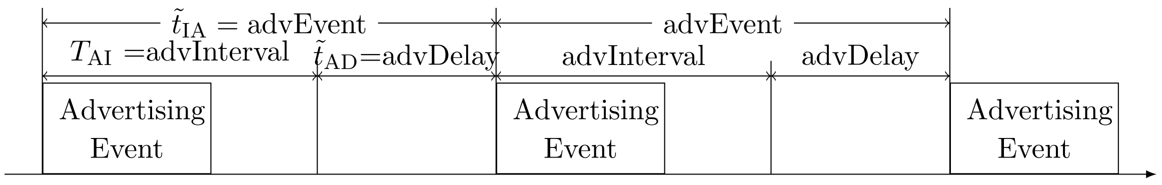

$\tilde{t}_{\text{IA}}$ = advEvent
advEvent
T_{AI} =advInterval
$\tilde{t}_{\text{AD}}$=advDelay
advInterval
advDelay
Advertising Event
Advertising Event
Advertising Event

Figure 2.3: Illustration of the advertising process

After the last transmission, there is a long radio silence of the time left in $\tilde{t}_{\text{AI}}$, followed by the advertising delay. The advertising delay has a uniform distribution over 0 to 10 ms. While there are many types of advertising events, we are only focusing on advertising events where the scanner's response is unnecessary.

The packet format can be seen in Fig. 2.4. Using the default symbol rate of the PHY layer, the preamble is one octet. Along with the PDU of 37 octets, the overall length of the advertisement is 47 octets or 376 bits. One symbol is translated as one bit. Therefore, the transmission rate of the PHY layer is 1 Mb/s, which results in a packet transmission time of $T_{\text{A}} = 376$ μs. While the present interest is focused on reception of these particular advertisements, any and all analysis can be modified to address other cases.

LSB MSB

Preamble (1 or 2 octets)	Access Address (4 octets)	PDU (2 to 257 octets)	CRC (3 octets)

Figure 2.4: Link layer packet format for the LE Uncoded PHYs[1, adapted from pg. 2562]

CHAPTER 3

INDEPENDENCE OF SUCCESSIVE COLLISIONS

In this chapter, we will explore a particular system under study. In this system, a number of items need to be scanned by a BLE scanner. These items are sorted into large, fixed-sized clusters and move to a scanning area. Each member of the group is outfitted with a BLE device that broadcasts packets, known as advertisements, to the system. The rate at which the advertisements broadcast is determined by its interadvertisement time, which is the time between two consecutive advertising events within the same data set as described in [1] and discussed in Chapter 2.

We denote a scanning job (SJ) to be the operation in which a batch of advertisers is scanned. If the system fails to successfully scan any item in a group, that operation is marked as a failure (SJF). For this thesis, we ask how to set the interadvertisement time that minimizes the scanning time while achieving a prescribed minimal probability of failure to successfully scan every advertiser. This question was investigated in a recent working paper. However, for this body of work, we examine the simulation program and analysis in more detail within this chapter. First, we define how to minimize the probability of failure. Next, we look at the key parts of the simulation program, as well as the results from multiple runs.

3.1 Minimizing Probability of Scanning Job Failure

Recalling from Chapter 2, advertisers transmit advertisements sequentially on the primary advertising channels (indexed at 37, 38, and 39). During an arbitrary advertiser's transmission, the scanner will only be listening to one of those channels at that time. Thus, the load placed on a channel is the advertisement transmitted at the current scanning frequency at an arbitrary point in time. We denote the load on a channel due to a single advertiser as

$$\rho_{\mathrm{A}} = T_{\mathrm{A}} / \mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right],$$

where T_{A} is the packet transmission time and $\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$ is the expectation of the interadvertisement time.

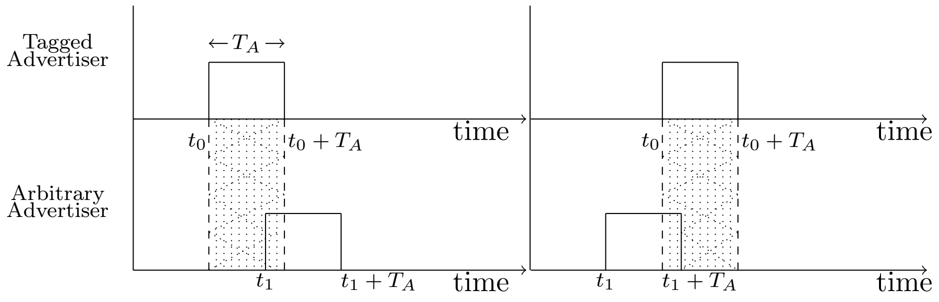


Figure 3.1: Illustrations of packet collisions

Fig 3.1 illustrates an advertisement collision between a tagged advertiser and an arbitrary advertiser. Shown are two cases of collision with the tagged advertiser's transmission. The left side shows a late arrival of an arbitrary advertisement, while the right side shows the early arrival case. In chapter 2, we saw that T_{A} represents the packet transmission time, as noted in the figure. Suppose the tagged advertiser sends an advertisement at t_0. If the arbitrary advertiser's advertisement overlaps with the tagged advertiser's advertisement, we say that the packets collided. In both instances, the arbitrary advertiser's advertisement is within the shaded area of the tagged advertiser's advertisement, indicating a collision. Thus, the vulnerable period is $t_1 \in (t_0 - T_{\mathrm{A}}, t_0 + T_{\mathrm{A}})$ so that the length of the vulnerable period is $2T_{\mathrm{A}}$.

Given the vulnerable period results in a failed scanned advertisement for the tagged and arbitrary advertisers, the probability of failure due to overlap is $2\rho_{\mathrm{A}}$. Hence, the probability that an arbitrary advertiser does not interfere with a tagged advertiser's advertisement is $(1 - 2\rho_A)$.

Define N_A to be the batch size. Given a batch size of size N_A, all transmitting independently, the probability that a given arbitrary advertisement is successfully scanned is

$$\mathcal{S}_\mathrm{AA} = P\left\{\mathcal{S}_\mathrm{AA}\right\} = (1 - 2\rho_\mathrm{A})^{N_\mathrm{A}-1} \approx e^{-2\rho_\mathrm{A}(N_\mathrm{A}-1)},$$

where the approximation is based on the definition of the mathematical constant e and is very good for large N_A and a small ρ_A [10].

We have defined a scanning job (SJ) to be the scanning of a batch of advertisers for a given period of time, T_S. If all advertisers are successfully scanned at least once during the scanning period, that SJ is considered a success. Otherwise, it is considered a SJ failure, which is denoted by SJF. Assuming the event of independent, successful scans by all advertisers with each advertiser having $T_\mathrm{S}/\mathrm{E}\left[\tilde{t}_\mathrm{IA}\right]$ advertising events per scan period, the failure probability of a scanning job is given by

$$P\left\{\mathrm{SJF}\right\} = \left(1 - e^{-2\frac{T_\mathrm{A}(N_\mathrm{A}-1)}{\mathrm{E}\left[\tilde{t}_\mathrm{IA}\right]}}\right)^{\frac{T_\mathrm{S}}{\mathrm{E}\left[\tilde{t}_\mathrm{IA}\right]}}. \tag{3.1}$$

From the previous equation, we can rewrite the equation in the following form

$$P\left\{\mathrm{SJF}\right\} = f(x) = \left(1 - e^{-ax}\right)^{xT_\mathrm{S}} = \left[\left(1 - e^{-ax}\right)^{x}\right]^{T_\mathrm{S}}, \tag{3.2}$$

where $x = 1/\mathrm{E}\left[\tilde{t}_\mathrm{IA}\right]$ and $a = 2T_\mathrm{A}\left(N_\mathrm{A} - 1\right)$. From (3.2), we see that $f(x)$ is by minimized for a particular value of x that is independent of T_S [10]. Thus, we can remove T_S from the minimization process. In further analysis, we see that minimizing $P\left\{\mathrm{SJF}\right\}$ with the value of x is equivalent to the same value that minimizes $\ln P\left\{\mathrm{SJF}\right\}$. Therefore, we can minimize

$\ln P\{\text{SJF}\}$ to find x. Thus,

$$x = \operatorname{argmin} \ln P\{\text{SJF}\} = \operatorname{argmin} \ln f(x) = \operatorname{argmin}[x \ln\left(1 - e^{-ax}\right)]. \tag{3.3}$$

In order to find the argmin$[x \ln(1 - e^{-ax})]$, we define that

$$y = 1 - e^{-ax} \quad \Rightarrow x = -\frac{\ln(1-y)}{a}, \tag{3.4}$$

Substituting 3.4 into 3.3, we want to find the following:

$$\operatorname{argmin} \ln f(x) = \operatorname{argmin}\ g(y) = \operatorname{argmin}\left[-\frac{1}{a}\ln(1-y)\ln y\right]. \tag{3.5}$$

From [10], proof of convexity of $g(y)$ was obtained. Therefore we can obtain argmin $g(y)$ by solving $g'(y) = 0$ for y. Upon taking the derivative, we obtain the following:

$$\frac{d}{dy}g(y) = -\frac{1}{a}\left[\frac{\ln(1-y)}{y} - \frac{\ln y}{1-y}\right]. \tag{3.6}$$

By setting $g'(y) = 0$, we removed $-\frac{1}{a}$ and obtained the following result:

$$\frac{\ln(1-y^*)}{y^*} - \frac{\ln y^*}{1-y^*} = 0. \Rightarrow (1-y^*)\ln(1-y^*) = y^* \ln y^*$$

From here, it can be clearly seen that $y^* = 1 - y^*$, which results in $y^* = 0.5$. With $a = 2T_{\text{A}}(N_{\text{A}} - 1)$ and $x = 1/\text{E}\left[\tilde{t}_{\text{IA}}\right]$, we solved for the expected value of the interadvertisement time, which results in the following equation

$$\text{E}\left[\tilde{t}_{\text{IA}}\right] = -\frac{2T_{\text{A}}(N_{\text{A}}-1)}{\ln 0.5} = \frac{2T_{\text{A}}(N_{\text{A}}-1)}{\ln 2}. \tag{3.7}$$

By solving (3.1) for T_{S}, we found

$$T_{\mathrm{S}} \geq \mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right] \frac{\ln(P\left\{\mathrm{SJF}\right\})}{\ln 0.5}, \tag{3.8}$$

3.2 Simulation Analysis

In this section, we will review the structure of the BLE simulation, which was written in using SimPy, an open-source, object-oriented framework for discrete-event simulations in Python. The simulations are performed in discrete steps in the form of an event queue, where each event has a simulation time at which the event occurs.

In order to understand the system under study, we carefully modeled the BLE advertisers and their advertisements along with their interactions with a BLE scan routine and each other. The following section will go over three classes necessary to simulate these advertisers and their advertisements as well as track the scanner's frequency. Afterwards, we will observe how the simulator behaves in the key methods section. Analysis of the simulation results is next, followed by conclusion of the results.

3.2.1 Classes

Based on gathered insight from [1] and [10], the Advertiser class carefully models identical, independent BLE advertisers. When creating an advertiser, the very first method called is Initialization. Along with an ID number, the initialization method sets several attributes, allowing us to keep track of the individual advertiser, their advertisements, and their transmission status. At the end of the method, initialization calls the Arrive method. In this method, arrival times for each advertisers' advertising event, as described earlier, are generated and placed in the event queue.

In the Arrival method, the Transmit method is called to process the transmission of the advertisements within the advertising event. The Transmit method provides several boolean and integer variables that aid in tracking and counting advertisements and their interactions. Within the Transmit method, an object of type Frame is created using the Create Frame method and marked with an advertising frequency to broadcast. In this method, three advertisements are created and placed in a transmit queue. Once ready to transmit, the first three advertisements in the queue are sent out serially with the appropriate advertising frequency. Additionally, the Transmit method calls the Check Collision method.

The Check Collision method takes the individual advertisement in transmission and checks for overlap with other advertisements sent. Lastly, there are the Add Frame and Remove Frame methods. With these methods, advertisements, given their ID number are added or removed from the transmit queue at the beginning and end of their transmissions.

The Frame class is used to create advertisements for the individual advertisers. Fig 3.2 displays the executable code. As seen from this figure, the class only has an initialization method, where it takes in parameters from the Advertiser class's Create Frame method. The *frame_time* variable is packet transmission time, which is 376 μs. The *start* and *end* variables set a timestamp for the advertisement in the event queue. boolean variables are created in the case of advertisement collisions and unmatched frequencies with the scanner. Additionally, there is a counter for cases where the scanner switches frequency in the midst of a transmission.

```
class Frame:
    def _init_(self, start, end, frame_time, frequency):
        self.start = start
        self.end = end
        self.frame_time = frame_time
        self.frequency = frequency
        self.collision = False
        self.wrong_freq = False
        self.change_frequency= 0
```

Figure 3.2: Frame class

The Global variables class serves to keep variables that are used and updated in multiple parts of the program. We can see from Fig. 3.3 that there's only one variable under the class. For the *current frequency* variable, it is first initialized to zero, which represents the primary advertising channel index 37. From here, the variable is called during the scan routine, which cycles through the three primary advertising channels in a round-robin fashion. As the global variable consistently updates through the scan routine, it can be called in the Transmit method of the Advertiser class without issues.

```
class G:
    current_frequency = 0
```

Figure 3.3: Global variables class

3.2.2 Key Methods

3.2.2.1 Generation of Advertisements

For our model, we assumed that all advertisers advertise independently of each other and will start broadcasting at a random time relative to each other. As such, their first advertisements will occur after a random period of time within their advertising interval to the end of their advertising interval. This period of time is known as the forward recurrence time of the interadvertisement time, which is illustrated in Fig. 3.4. If the density of the interadvertisement time is given by $f_{\tilde{x}}(x)$, then the density of the forward recurrence time of the interadvertisement time is given by $\frac{1-F_{\tilde{x}}(x)}{\mathrm{E}[\tilde{x}]}$ [10]. In this particular case, $\tilde{x}$ consists of two parts, a deterministic part and a random part, with the result

$$\tilde{x} = T_{\mathrm{AI}} + \tilde{t}_{\mathrm{AD}},$$

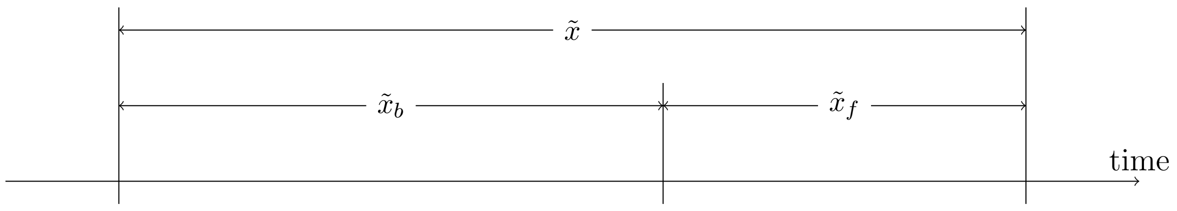


Figure 3.4: Illustration of forward recurrence time

where $\tilde{t}_{\text{AD}}$ is uniformly distributed over $(0, 10)$. Thus, $\tilde{x}$ is uniformly distributed over $(T_{\text{AI}}, T_{\text{AI}} + 10)$.

Through a series of mathematical manipulations, it can be shown that for the general case of a uniformly distributed random period, the density function for the forward recurrence time would be

$$f_{\tilde{x}_f}(x) = \begin{cases} \frac{1}{\left(T_{\text{AI}}+\text{E}\left[\tilde{t}_{\text{AD}}\right]\right)}, & x \in (0, T_{\text{AI}}) \\ \frac{2\text{E}\left[\tilde{t}_{\text{AD}}\right]+T_{\text{AI}}-x}{2\text{E}\left[\tilde{t}_{\text{AD}}\right]\left(T_{\text{AI}}+\text{E}\left[\tilde{t}_{\text{AD}}\right]\right)}, & x \in \left(T_{\text{AI}}, T_{\text{AI}} + 2\text{E}\left[\tilde{t}_{\text{AD}}\right]\right) \cdot \\ 0, & \text{else.} \end{cases} \tag{3.9}$$

and after integration, the cumulative distribution is found to be

$$F_{\tilde{x}_f}(x) = \begin{cases} \frac{x}{\left(T_{\text{AI}}+\text{E}\left[\tilde{t}_{\text{AD}}\right]\right)}, & x \in (0, T_{\text{AI}}) \\ \frac{1}{\left(T_{\text{AI}}+\text{E}\left[\tilde{t}_{\text{AD}}\right]\right)} \left[\frac{-x^2+2\left(T_{\text{AI}}+2\text{E}\left[\tilde{t}_{\text{AD}}\right]\right)x-T_{\text{AI}}^2}{4\text{E}\left[\tilde{t}_{\text{AD}}\right]}\right], & x \in \left(T_{\text{AI}}, T_{\text{AI}} + 2\text{E}\left[\tilde{t}_{\text{AD}}\right]\right) \cdot \\ 1, & x > T_{\text{AI}} + 2\text{E}\left[\tilde{t}_{\text{AD}}\right]. \end{cases} \tag{3.10}$$

Given the distribution of the forward recurrence times as in (3.10), the inversion process for choosing variates from the distribution is to solve (3.10) for x in terms of $F_{\tilde{x}_f}(x)$, which yields the following:

$$x = \begin{cases} \left(T_{\text{AI}} + \text{E}\left[\tilde{t}_{\text{AD}}\right]\right) F_{\tilde{x}_f}(x), & F_{\tilde{x}_f}(x) \leq \frac{T_{\text{AI}}}{T_{\text{AI}}+\text{E}\left[\tilde{t}_{\text{AD}}\right]} \\ T_{\text{AI}} + 2\text{E}\left[\tilde{t}_{\text{AD}}\right] - 2\sqrt{\left[1 - F_{\tilde{x}_f}(x)\right] \text{E}\left[\tilde{t}_{\text{AD}}\right] \left(T_{\text{AI}} + \text{E}\left[\tilde{t}_{\text{AD}}\right]\right)}, & F_{\tilde{x}_f}(x) \in \left(\frac{T_{\text{AI}}}{T_{\text{AI}}+\text{E}\left[\tilde{t}_{\text{AD}}\right]}, 1\right), \end{cases} \tag{3.11}$$

which is used to generate arrival times of the initial advertisements. With the aforementioned equations, we created a method within our simulator, as shown in Fig 3.5. As clearly seen, we use an if-else statement that determines the value of x. The method is called during the Arrive method of the Advertise class which is shown in Fig 3.6. The *T_AI* and the *Et_AD* variables

```
def exceptional_first_arrival():
    Fx = random.uniform(0,1)
    if $Fx \leq (T\_AI/(T\_AI + Et\_AD))$:
        x = $(T\_AI + Et\_AD) * Fx$
    else:
        x = $T\_AI + (2 * Et\_AD) - m.sqrt((1 - Fx) * (4 * Et\_AD) * (T\_AI + Et\_AD))$
    return x
```

Figure 3.5: Initial advertisement arrival generator

represent the advertising interval and the expectation of the advertising delay, respectively. As mentioned earlier in Chapter 2, the advertising delay uses a uniform distribution from 0 to 10 ms. Thus, the expectation, or average, of the delay is 5ms. Obtaining the $\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$ value from (3.7), we subtract $\mathrm{E}\left[\tilde{t}_{\mathrm{AD}}\right]$ from it to get t_{AI}.

As noted earlier, Arrive is defined in the Advertiser class. From the code provided in Fig 3.6, there is a counter that initializes to zero, indicating the very first advertising event. The following two lines calls the first arrival method and uses the return value as a delay in the event queue. Afterwards, the initial set of advertisements is sent to the Transmit method, along with the counter as a frame ID. After the first advertising event, all subsequent advertising events use the arrival formula indicated in the while loop. Though the while loop is infinite, control is passed back to the simulation environment at the simulation termination time.

```
def arrive(self):
    i = 0
    inter_t = exceptional_first_arrival()
    yield env.timeout(inter_t)
    env.process(self.transmit('Frame %d'   % i))
    i += 1
    while True:
        inter_t = T_AI + random.uniform(0, RAN_INTER_LENGTH)
        yield env.timeout(inter_t)
        env.process(self.transmit('Frame %d'  % i))
        i += 1
```

Figure 3.6: Advertisement arrival generator

In continuing our journey of generating advertisements, we turn to the Transmit method, shown in Fig 3.7. With the advertising events created from the Arrive method, the Transmit method calls the Frame class to create the individual advertisements and broadcasts them sequentially through the primary advertising channels. To begin the process, we initialize the variable, *freq*, to zero. The *freq* counter is used to denote which frequency is used to broadcast an advertisement. This allows us to consistently stay with the *G.current_frequency* variable, which is utilized the same way.

The three advertisements are created in a while loop for the advertising event. The code in the while loop is executed 3 times, representing the transmission of an advertisement on each advertising channel. For each loop through the while statement, the Create Frame method is called, generating an advertisement with a name and ID for tracking purposes. Then the Add Frame method is called to add the advertisements to a transmit list for that advertiser. We compare the frequency on the advertisement to the current frequency of the scanner. If they match, then the Check Collision method is called. Otherwise, we marked the boolean variable, *frame.wrong_freq*, as *True*. This is to denote the unsuccessful scan of the advertisement by the scanner.

From chapter 2, we saw that the advertisement's transmission duration is 376 μs. Additionally, there is a period of time between the start times of two consecutive advertisement. In the Transmit method, these two variables are referred to as the *FRAME_TIME* and *DEAD_TIME* variable in the Transmit method. These two variables are used to simulate the transmission of advertisements once they have been removed from the transmit list, using the Remove Frame method. To accommodate for the other reasons advertisements fail to be successfully scanned besides a collision, a couple of if statements are used. The first if statement accounts for unmatched frequencies between the scanner's current frequency and advertisement. The second if statement accounts for the scanner changing frequencies during the transmission of the advertisement. After those statements, an if/else statement is used to update advertiser attributes for tracking and statistical purposes. At the end of the loop,

```
def transmit(self, name):
    freq = 0
    while freq < 3:
        frame_name = 3 * self.num_initial_transmits + freq
        frame = self.create_frame(FRAME_TIME, freq)
        frame_id = (self.name, frame_name)
        self.add_frame_in_transmit(frame, frame_id)
        if freq != G.current_frequency:
            frame.wrong_freq = True
        else:
            self.num_transmit += 1
            self.check_collision(frame, frame_id, freq)
        yield env.timeout(FRAME_TIME)
        self.remove_frame_in_transmit(frame_id)
        if freq != G.current_frequency and frame.wrong_freq != True:
            frame.change_freq = True
            self.num_change_freq += 1
        yield env.timeout(DEAD_TIME)
        self.sum_transmit = self.sum_transmit + FRAME_TIME
        self.busy_time = self.busy_time + FRAME_TIME
        if frame.wrong_freq = True:
            self.num_wrong_freq += 1
        elif frame.collision == True:
            self.num_collision += 1
        else:
            self.num_success += 1
        freq += 1
```

Figure 3.7: Transmit method

the *freq* is updated and returns back to the beginning of the while loop, until $freq = 3$.

3.2.2.2 Scan Routine

Recalling Fig. 2.2, the scanner listens for advertisements on an advertising channel for the duration of its scan window. For our system, the scanner does not sleep. Therefore, we will be utilizing part (b) of Fig. 2.2. In the scan routine, we model the scan frequency as a variable that holds a value of 0, 1, or 2. These values represent the primary advertising channels. From Fig 3.8, the scanner starts at 0, representing advertising channel 37. This was initialized using the *current_frequency* variable from the Global variables class. Once an event occurs that calls the routine, the scanner scans for a certain duration. Then it updates the *current_frequency* variable to 1, which represents advertising channel 38. Similar to the Arrive method, it will continue moving through these 3 states in a while loop until the simulation terminates.

```
def scan():
    while True:
        yield env.timeout(SCAN_DURATION)
        G.current_frequency = (G.current_frequency + 1) % 3
```

Figure 3.8: Scan routine

3.2.2.3 Reset Statistical Counters

In our simulation, each advertiser goes through a warm-up period. We denote this as the variable, *TRANSIENT_TIME*. Due to the various counter and boolean variables used to gain precise statistical data , the Reset method ensures that the artifacts of the warm-up period are eliminated from the steady state simulation results. Fig 3.9 displays the lines of code for the Reset method. The method is called after each advertiser is made. During that time, the simulation environment holds that state for the duration of the warm-up period. Then it resets the appropriate values for each advertiser in the current scanning group.

```
def reset_statistical_counters(advertisers):
    print "transient time is %s" %str(TRANSIENT_TIME)
    yield env.timeout(TRANSIENT_TIME)
    print"the statistics are reset at %s" %str(env.now)
    for advertiser in advertisers:
        advertiser.num_transmit = 0
        advertiser.sum_transmit = 0
        advertiser.num_initial_transmits = 0
        advertiser.num_transmit = 0
        advertiser.busy_time = 0
        advertiser.num_collision = 0
        advertiser.num_wrong_freq = 0
        advertiser.num_change_freq = 0
        advertiser.num_success = 0
        advertiser.steady_state_time = 0
        advertiser.Util = 0
```

Figure 3.9: Reset method

3.2.2.4 Detecting Collisions

There are three ways in which a specific BLE advertisement fails to be scanned successfully. First, if the channel on which the advertisement is transmitted fails to match the scanning frequency at any time during the transmission of the advertisement, the scanning fails. Second, if the channel on which the advertisement is transmitted matches the scanning frequency and the scanning frequency changes during the transmission of the advertisement, the scanning fails. These two types of failures are checked in the Transmit method as shown in Fig 3.7. The final method of a specific BLE advertisement failing to be scanned successfully is from colliding with an advertisement from an arbitrary advertiser. Figure 3.10 displays the code for process a collision between advertisements.

```
def check_collision (self, frame, frame_id, frequency):
    if frame.frequency != frequency:
        print"something is wrong in check_collision"
    for key in Advertiser.frames_in_transmit.keys():
        if key != frame_id:
            if Advertiser.frames_in_transmit[key].frequency == frame.frequency:
                if (Advertiser.frames_in_transmit[key].end > frame.start) and
                 (Advertiser.frames_in_transmit[key].start < frame.end):
                    frame.collision = True
                    Advertiser.frames_in_transmit[key].collision = True
```

Figure 3.10: Check collisions method

The Check Collision method is called from the Transmit method, which occurs at the beginning of each transmission of a BLE advertisement. The method takes the advertisement, its unique ID, and the current frequency of its transmission as parameters for the process. First it checks if the frequency from the parameter accurately matches what the advertisement is supposed to have. Next, we cycle through every advertisement in the transmit queue. We only want to check for collisions between the current (tagged) advertiser's advertisement and an arbitrary advertiser's advertisement. Therefore, we do not want the advertisement that matches the tagged advertisement, which is why we give them each unique frame IDs. If an arbitrary advertisement does not have the same ID as the tagged advertisement, we check to see if the frequency matches. Assuming at this point the channel on which the tagged advertisement is transmitted matches the scanning frequency, an arbitrary advertisement, transmitted on the same channel, could hinder the successful scanning of the tagged advertisement. Finally, recalling Fig. 3.1, the advertisements collide within the vulnerable period of the start time of the tagged advertisement. Hence, we only need to check the tagged advertisement at the beginning of its transmission. If there is an overlap with the arbitrary advertisement, boolean variables for both is marked True for collisions. This cycle continues until every advertisement in the transmit queue is checked with the tagged advertisement.

3.2.2.5 Transmission List

In the Advertiser class, a list is instantiated to keep track of all the advertisements that will be transmitted, shown in Fig. 3.11. We denoted this as the *frames_in_transmit* attribute in the program. In order to add or remove advertisements within the list to simulated advertisements arriving and transmitting, the Add and Remove Frame methods were created as shown in the figure. These methods are called in the Transmit method and can be seen from Fig. 3.7. They both require the specific advertisement's ID as a pointer to its location within the transmit list.

```
class Advertiser(object):
    frames_in_transmit = {}
    def add_frame_in_transmit(self, frame, frame_id):
        Advertiser.frames_in_transmit[frame_id] = frame
        return
    def remove_frame_in_transmit(self, frame_id):
        del Advertiser.frames_in_transmit[frame_id]
        return
```

Figure 3.11: Add and remove frame from transmit queue methods

3.2.3 Script Automation

We use a bash shell script to automate the process of making runs in the simulation per batch size . After specifying model parameters, i.e number of advertisers, interadvertisement time, scan time, number of repetitions per run, and the simulation time, the script is ready to run. After each simulation run, statistics are gathered and logged into a file that is labeled by date, time, and system parameters. Afterwards, we use the results to plot graphs to view the system's performance in a more elegant perspective.

3.2.4 Simulation Results

For our simulation model, we prescribed the target failure probability of a scanning job ($P\{\text{SJF}\}$) to be 10^{-5} and simulated the scanning process for 1 million advertisements given some prescribed number of advertisers. In order to obtain a sufficient sample of failures, each simulation run is replicated for a number of times such that the total number of advertisements sums to 1 million, depending on the batch size. With the target failure probability at 10^{-5}, each batch size should yield about 10 failures per run. Table 3.1 displays the batch sizes and the number of repetitions to sum to 1 million advertisers per run, as well as the values for $\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$ and T_{S} . For example, a simulated run of 100 advertisers is replicated 10000 times. We ran 30 simulation runs for each batch size to obtain a sufficient number of samples of the mean number of failures. To achieve the true value of the mean number of failures, we would have to run each batch size for an infinite amount of time, which is not feasible.

N_{A}	$\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$	T_{S}	Replica	Runs
100	0.107	1.91	10000	30
200	0.216	3.80	5000	30
400	0.433	7.50	2500	30
800	0. 867	14.90	1250	30
1600	1.74	29.60	625	30

Table 3.1: Simulation parameters given number of advertisers

With the target $P\{\mathrm{SJF}\} = 10^{-5}$, we select the values of $\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$ and T_{S} to be placed in the automation script. Using (3.7) and (3.8) with $T_{\mathrm{A}} = 376\ \mu$s, we are able to derive $\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$ and T_{S} for each batch size of advertisers. In the case of $N_{\mathrm{A}} = 100$ as an example, $\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right] = 0.107$ and $T_{\mathrm{S}} = 1.777$. This equates to an average of $T_{\mathrm{S}}/\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right] = 16.53$ advertising cycles within a scanning period. Analyzing more closely, we note that in this case only 53% of the advertisers would generate 17 advertisements, with the remaining 47% only generating 16 advertisements. Ensuring a higher probability of success, we increased the scanning period to 1.91 s, to ensure that all advertisers have at least 17 advertisements. Now, a little over 76.7% of the advertisers would have 17 advertising cycles and the other 24.3% would have 18. We complete this procedure for every batch size.

The following table displays a comparison between the simulation and analytical results of the $P\{\mathrm{SJF}\}$ for $N_{\mathrm{A}} \in \{100, 200, 400, 800, 1600\}$ with $\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$s set optimally and required scan periods based on independent collisions with the target $P\{\text{SJF}\}$ set at 10^{-5} A 95% confidence interval, which is given by ± 1.96 standard deviations, for each batch size of the simulated results is also given in the table below. In comparing the simulated result of $P\{\text{SJF}\}$ with the analytical result, we see that there is a order of magnitude difference between them, even with the confidence interval. Moreover, the simulated results fail to meet the target $P\{\text{SJF}\}$ of 10^{-5}. This could suggest that the collisions may not be as independent as we hypothesize.

N_A	T_S (s)	Simulation		Analytical
		$P\{\text{SJF}\}$	CI [95%]	$P\{\text{SJF}\}$
100	1.91	4.04e-05	± 1.377e-06	4.43e-06
200	3.80	5.55e-05	± 9.119e-07	5.03e-06
400	7.50	6.52e-05	± 9.485e-07	6.09e-06
800	14.90	7.45e-05	± 6.555e-07	6.69e-06
1600	29.60	7.74e-05	± 4.542e-07	7.30e-06

Table 3.2: Comparison of failure probability of scanning job between the simulation and analytical results

Let us define $\tilde{n}_\text{AA}$ as the number of successful scans of an arbitrary advertiser over a scanning period. Fig. 3.12 depicts the PMF of $\tilde{n}_\text{AA}$ for $N_\text{A} = 100$, $\text{E}\left[\tilde{t}_\text{IA}\right] = 0.107$, $T_\text{S} = 1.91$. Observing when $\tilde{n}_\text{AA} < 5$ in the figure, it can be clearly seen that the simulation rests higher than the analytical. For example, when $\tilde{n}_\text{AA} = 1$ signifying one successful scan for an arbitrary advertiser, the simulation yields 4.45×10^{-4} while the analytical yields 1.21×10^{-4}, which is about 4 times less than the simulation. This signifies that the simulation presents a higher probability of fewer successes than the analytical results, which attributes to the higher probability of SJF in Table 3.2. Figures 3.13-3.16 show the same observation as the previous figure, which shows consistency regardless of the size of N_A. It can be reasoned from the table and associated figures that the simulation presents a larger $P\{\text{SJF}\}$ than would be expected if collisions were independent. Therefore, it would be of interest to examine if there is a dependence among collisions.

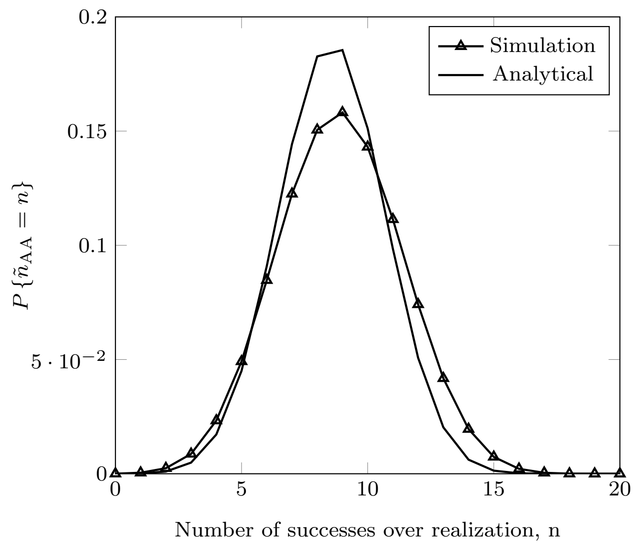


Figure 3.12: PMF of a number of successes from simulation run for batch size of 100 advertisers with a scanning period of 1.91 s and an average advertisement cycle of 0.107 s

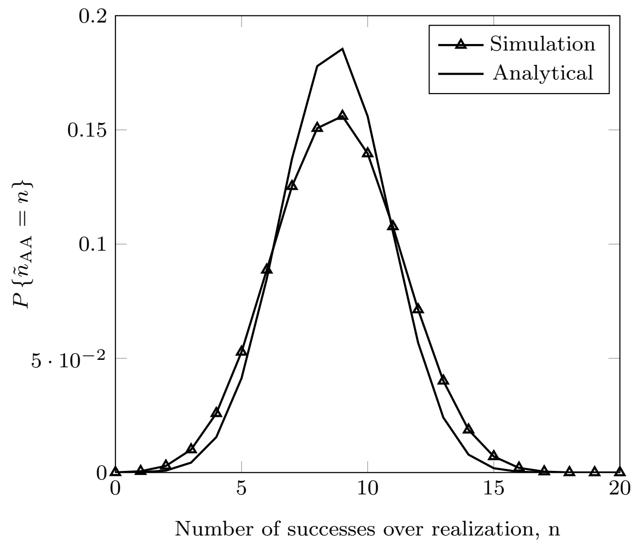


Figure 3.13: PMF of a number of successes from simulation run for batch size of 200 advertisers with a scanning period of 3.80 s and an average advertisement cycle of 0.216 s

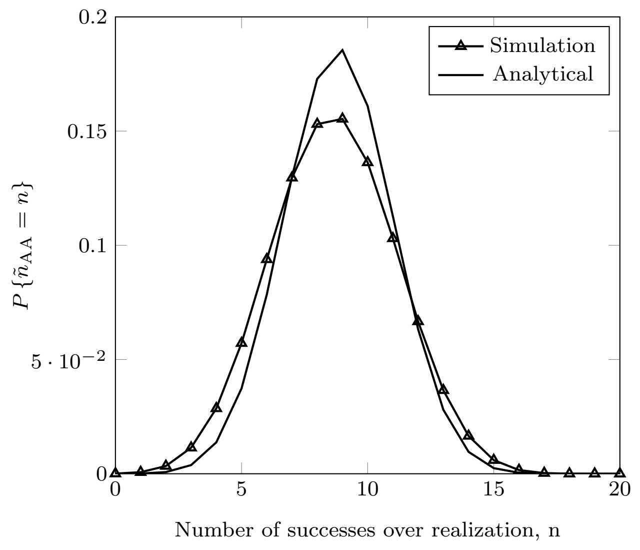


Figure 3.14: PMF of a number of successes from simulation run for batch size of 400 advertisers with a scanning period of 7.5 s and an average advertisement cycle of 0.433 s

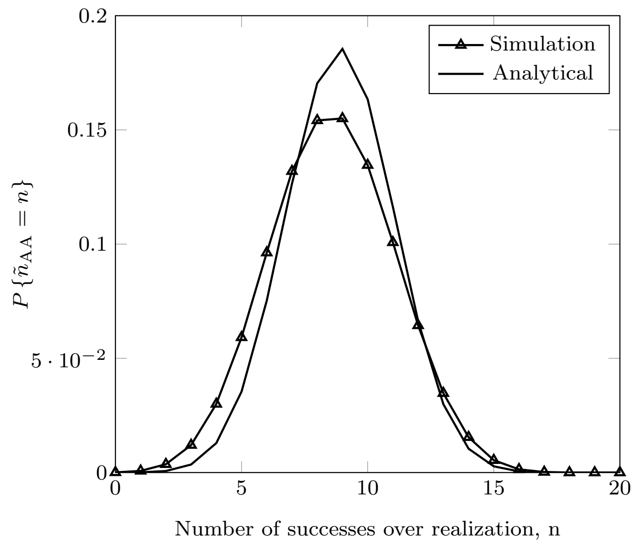


Figure 3.15: PMF of a number of successes from simulation run for batch size of 800 advertisers with a scanning period of 14.9 s and an average advertisement cycle of 0.867 s

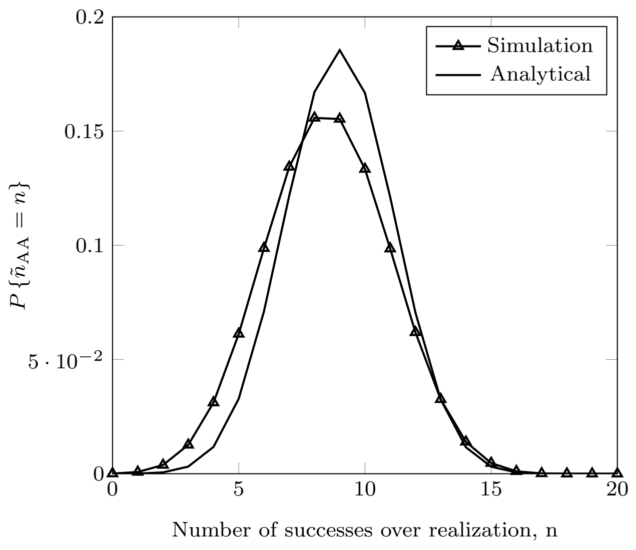


Figure 3.16: PMF of a number of successes from simulation run for batch size of 1600 advertisers with a scanning period of 29.6 s and an average advertisement cycle of 1.74 s

3.2.4.1 Chapter Summary

In this chapter, the simulation program is presented with the relevant classes, methods, and key function to model the BLE scanning and advertising process for our particular system. Parameters for each run were discussed as well as the reasoning behind them. Results were shown from the simulations and compared with our analytical findings. Analysis of the simulated results present the presence of not only a higher probability of SJF, but the failure to meet the prescribed $P\{\text{SJF}\}$ target of 10^{-5}. Thus, we will examine the nature of dependence of collisions in the following chapter.

CHAPTER 4

DEPENDENCE OF SUCCESSIVE COLLISIONS

In this chapter, we will examine the nature of dependence among advertisement collisions. We will derive the probability of successive collisions, $P\{\text{SC}\}$, using mathematical tools. Afterwards, analysis of successive collisions via simulations will be conducted. Following analysis, we will revisit finding the interadvertisement and scan times per batch size and analyze the simulation results. Lastly, we will test the modified interadvertisement time to determine its optimality compared to the interadvertisement time found under the assumption of independent collisions.

4.1 Mathematical Analysis of Probability of Successive Collisions

The following analysis has been presented in [10]. However for continuity purposes, it shall be discussed here as well. Though noted that while initial advertisement collision between two arbitrary advertisers are independent, we are unsure whether the events of the subsequent collisions are correlated to the first collision between those advertisers [10]. Therefore, using mathematical tools, we re-evaluate collision probabilities with respect to that correlation. For the rest of this section, we denote the tagged advertiser as advertiser A, while an arbitrary advertiser is denoted as advertiser B.

As seen in Fig. 3.1, suppose advertiser A starts transmitting an advertisement at time t_0 and overlaps with an advertisement from advertisement B. We know that advertiser B started at some arbitrary time, $\tilde{t}_{B,0}$, that is $\pm T_A$ from the start time of advertiser A's advertisement transmission, which is noted as the interval $(t_0 - T_\text{A}, t_0 + T_\text{A})$. From [10], we discovered analytically that the conditional probability of a repeated collision $(\mathcal{R}_{\mathcal{B},1})$ from advertiser B given the initial collision ($\mathcal{C}_0$) with said advertiser is

$$P\left\{\mathcal{R}_{\mathcal{B},1}|\mathcal{C}_0\right\} = \frac{2T_\mathrm{A}}{T_\mathrm{D}} - \frac{4}{3}\frac{T_\mathrm{A}}{T_\mathrm{D}} = 0.0733,$$

which is entirely independent of $\mathrm{E}\left[\tilde{t}_\mathrm{IA}\right]$. To put this value into perspective, suppose a group of 100 items are going to be scanned, $N_\mathrm{A} = 100$. The $\mathrm{E}\left[\tilde{t}_\mathrm{IA}\right]$ for that batch size is 0.107 s. Observing advertiser A, the probability that it collides with advertiser B is $\frac{2T_A}{\mathrm{E}\left[\tilde{t}_\mathrm{IA}\right]} = 0.007$. Thus, the event of a repeated collision with the same advertiser is over 10 times more likely to occur than colliding with a completely different advertiser. Furthermore, it was discovered that the conditional probability of a second repeated collision ($\mathcal{R}_{\mathcal{B},2}$) given the first repeated collision ($\mathcal{R}_{\mathcal{B},1}$) is $P\left\{\mathcal{R}_{\mathcal{B},2}|\mathcal{R}_{\mathcal{B},1}\right\} = 0.0736$ [10], which has slightly increased compared to the $P\left\{\mathcal{R}_{\mathcal{B},1}|\mathcal{C}_0\right\}$. Though more analysis is needed, it can be seen that repeated collisions have a more profound effect on the $P\left\{\mathrm{SJF}\right\}$ compared to independent collision events. The following section will discuss analysis of successive collisions via simulations.

4.2 Simulation Analysis of Successive Collisions

Due to the findings on repeated collisions, we used a discrete Markov chain in our simulations in order to observe the transition matrix of collisions. The objective is to see the conditional probability of a future collision given a string of repeating collisions, starting with the first advertisement's transmission. In order to complete this task, we needed two things: 1). additional methods in the simulation program and 2). a large sample size per batch size.

To begin, we go back to the Advertiser class. From Fig. 4.1, we see the initialization method for when the object of type Advertiser is created, as mentioned in Chapter 3. However, for observing correlated collisions, we added the *ad_record* attribute. This variable is utilized in the later part of the while loop in the Transmit method, as seen in Fig. 4.2. After transmitting an advertisement in one instance in the loop, we check certain boolean variables and update the appropriate counters. It is there that we see if the *frame.collision* variable was set to True. Should that be the case, then we append the *ad_record* variable with a boolean statement False, signifying that an advertisement collision for that advertiser's current transmission. Otherwise, a boolean statement True is appended to the list, classifying that a collision did not occur for that advertiser's current transmission. This is done for every advertiser's transmissions per batch size per repetition. In order to prevent artifacts of the warm-up period from compromising the list, the *ad_record* variable was also in the Reset method.

```
def _init_(self, name):
    self.name = name
    self.num_initial_transmits = 0
    self.x = 0
    self.num_transmit = 0
    self.num_collision = 0
    self.num_wrong_freq = 0
    self.num_change_freq = 0
    self.num_success = 0
    self.ad_record =[]
    self.viableFrames=[]
    self.initial_reset_completed = False
    self.sum_transmit = 0
    self.mean_transmit = 0
    self.busy_time = 0
    self.steady_state_time = 0
    self.Util = 0
    env.process(self.arrive())
```

Figure 4.1: Initialization method

```
def transmit(self, name):
    freq = 0
    while freq < 3:
        frame_name = 3 * self.num_initial_transmits + freq
        frame = self.create_frame(FRAME_TIME, freq)
        frame_id = (self.name, frame_name)
        self.add_frame_in_transmit(frame, frame_id)
        if freq != G.current_frequency:
            frame.wrong_freq = True
        else:
            self.num_transmit += 1
            self.check_collision(frame, frame_id, freq)
        yield env.timeout(FRAME_TIME)
        self.remove_frame_in_transmit(frame_id)
        if freq != G.current_frequency and frame.wrong_freq != True:
            frame.change_freq = True
            self.num_change_freq += 1
        yield env.timeout(DEAD_TIME)
        self.sum_transmit = self.sum_transmit + FRAME_TIME
        self.busy_time = self.busy_time + FRAME_TIME
        if frame.wrong_freq = True:
            self.num_wrong_freq += 1
        elif frame.collision == True:
            self.num_collision += 1
            self.ad_record.append(False)
        else:
            self.ad_record.append(True)
            self.num_success += 1
        freq += 1
```

Figure 4.2: Modified Transmit method

To take advantage of the *ad_record* list in the simulation program, we created the Chain class. This class takes in the list as a parameter and instantiates attributes for a discrete Markov chain. In this class, it creates an object of type Chain, as shown in Fig. 4.3. Since each *ad_record* list contains a series of True and False boolean statements, we need a method to convert them into numerical values suitable for statistical analysis. Shown in Fig. 4.4 is the Convert_2 method. In computer programming languages, True and False are equivalent to a binary 1 and a binary 0. Thus, the Convert method transforms binary values from the *ad_record* list into decimal values. For example, an advertiser had three transmissions, where the first one succeeds and the other two failed. This results in a binary 001, which equates to a 1 when using the Convert_2 method.

```
class Chain:
    def _init_(self, run_record):
        if False in run_record:
            self.first_collision = run_record.index(False)
        else:
            self.first_collision = len(run_record)
        self.first_collision_set = True
        self.transitions = []
```

Figure 4.3: Chain class

```
def convert_2(lst):
    n=len(lst)
    j=lst[n-1]
    for i in range(1,n):
        $j+ = lst[i-1] * 2^{**}(n-i)$
        return j
```

Figure 4.4: Convert_2 method

Now that we have a way to convert binary numbers and instantiate a Markov chain object, we need a way to determine a state space by memory length, which represents the length of successive failures for an advertiser we wish to observe. For example, a memory length of 3 means the state space is $0, 1, 2, 3, 4, 5, 6, 7$. This creates the number of transitions going into and from each state from which transition probabilities are obtained. Thus, we created the Correlate Failures method, which is displayed in Fig. 4.5. In this method, the *ad_record* list for each replica of an advertiser in a batch size per simulation run is used as a parameter along with the variable *memory.* Raised to the power of 2, memory is used to determine the number of states in a state space. For example, if memory equal 3, the number of states in a state space is 8. Next, the method creates a object of type Chain, using the *ad_record* as a parameter. Afterwards, the method creates columns and vectors in the Chain object equal to the number of states in a state space. Using the earlier example, this will create a 8x8 matrix. Finally, the method goes through the *ad_record* list and completes the transition matrix, given the memory length. Going back to the example, we already have a 8x8 matrix. Let *ad_record* have a length of 11, which means there is 2^{11} combinations of True and False in a list. Taking one instance of the list, it goes through the last for loop 8 times, which is the difference between the length of the list and the memory. Inside the for loop, the transition values from one state to another are calculated and added to the transition matrix, using the Convert_2 method. The end result is an estimate of the conditional probability for correlated failures given the memory length in the form of transition matrix for a particular *ad_record* list for a replica of an advertiser in a batch size per simulation run.

```
def correlate_failures_nStep(memory,run_record):
    $num\_states = 2^{**}memory$
    x = Chain(run_record)
    $x.transitions = [0] * num\_states$
    for $i$ in range(num_states):
        $x.transitions[i] = [0] * num\_states$
    for $i$ in range(x.length − memory):
        $j = convert\_2(run\_record[i : i + memory])$
        $k = convert\_2(run\_record[i + 1 : i + 1 + memory])$
        $x.transitions[j][k] += 1$
    return x
```

Figure 4.5: Correlate failure method

Now that there is a transition matrix per replica of an advertisement in a batch size per simulation run, we can collect the mean value for an entire batch size. This is done through the Compute Transition method, shown in Figure 4.6. First, we compute the number of states using the memory parameter, similar to the Correlate Failures method. Then columns and vectors are created and initialized in the *cum_transition* variable. The *cum_transition* variable acts as a summation of each transition matrix per replica of an advertisement in a batch size per simulation run, which were gathered in the parameter *all_chains*. The summation takes places in a for loop after the initialization of the *cum_transition* variable. Afterwards, the sum of each row is calculated. Finally, the average transition probability from one state to another is computed using the *cum_transition* variable and the row sum.

```
def compute_transition_probabilities(memory,all_chains):
    $num\_states = 2^{**}memory$
    cum_transition = $num\_states * [0]$
    for $i$ in range(num_states):
        $cum\_transition[i] = num\_states * [0]$
    for run_chains in all_chains:
        for ec in run_chains:
            if $len(ec.transitions) > 0$:
                for $i$ in range(num_states):
                    for $j$ in range(num_states):
                        $cum\_transition[i][j] += ec.transitions[i][j]$
    $trans\_probs = num\_states * [0]$
    for $i$ in range(num_states):
        $trans\_probs[i] = num\_states * [0]$
        row_sum = sum(cum_transition[i])
    if row_sum != 0:
        for $j$ in range(num_states):
            $trans\_probs[i][j] = float(cum\_transition[i][j])/row\_sum$
    $s = num\_states - 1$
    return
```

Figure 4.6: Compute transition method

Though we have created the necessary tools in the simulation, a large sample size must be gathered in order to get a decent sizable string of repeated collisions. We decided that a memory length of 8 would by acceptable, as $2^8 = 256$, which gives us 256 transition states. However, the parameters in Fig. 3.1 will not suffice. For example, with the $N_{\text{A}} = 800$, $\text{E}\left[\tilde{t}_{\text{IA}}\right] = 0.86684$, and $T_{\text{S}} = 14.9$, each advertiser would advertise $\frac{T_{\text{S}}}{\text{E}\left[\tilde{t}_{\text{IA}}\right]} = 17$ times. Since only 17 advertisements are broadcast per cycle, only a small fraction of the states will be visited. Thus to ensure more visits to each state, either longer scan times or more replicas are required. For the purposes of this study, we chose to increase the scan times for each batch size to 12,000 sec. To maintain the same number of advertisements per cycle across all batch sizes, the number of replicas were adjusted to reflect the consistency. Figure 4.1 displays the new simulation parameters for transition probability analysis.

N_A	$\text{E}\left[\tilde{t}_\text{IA}\right]$	T_S	Replicas	Runs
100	0.107	12000	96	30
200	0.216	12000	48	30
400	0.433	12000	24	30
800	0. 867	12000	12	30
1600	1.74	12000	6	30

Table 4.1: Simulation parameters given number of advertisers for transition probability analysis

Table 4.2 displays the results from the simulation analysis of the transition probabilities per batch size given the memory length. From the table, we can see that the probability of repeated collisions, $P\left\{\mathcal{C}_n|\mathcal{C}_{n-1},\mathcal{C}_{n-2},\ldots,\mathcal{C}_0\right\}$ [10], does increase as the memory length increases as will. Additionally, it is bigger than the 0.5 we found in our independence results and seems to be converging to a value in the neighborhood of 0.6.

Memory	Mean Collision Return Probability				
Length	Number of Advertisers				
	100	200	400	800	1600
1	0.521	0.525	0.525	0.526	0.527
2	0.536	0.540	0.541	0.542	0.543
3	0.547	0.552	0.554	0.555	0.556
4	0.556	0.563	0.565	0.566	0.567
5	0.564	0.572	0.574	0.575	0.577
6	0.571	0.580	0.582	0.584	0.586
7	0.578	0.587	0.590	0.592	0.593
8	0.583	0.593	0.597	0.599	0.600

Table 4.2: Conditional probabilities of collision for a range of group sizes and memory lengths

4.3 Revisiting Optimal Interadvertisement and Minimal Scan Times

With the increase in the probability of repeated collisions, we went back to (3.7) and (3.8). We know that the $P\left\{\text{SJF}\right\}$'s are dependent on the probability of repeated collisions, due to the fact that the repeated collisions are the reasons for the SJF. Based on Table 4.2, we arbitrarily chose $P\left\{\mathcal{C}_n|\mathcal{C}_{n-1},\mathcal{C}_{n-2},\ldots,\mathcal{C}_0\right\} = 0.6$; however, a more thorough analysis will be needed to determine if this was the correct value. For now, we assume that the transition matrix will converge to this value. As such, we used the difference between 0.6 and 0.5 to establish a bias value B so that (3.7) and (3.8) becomes

$$\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right] = \frac{2T_{\mathrm{A}}(N_{\mathrm{A}}-1)}{\ln[2/(1+B)]}, \tag{4.1}$$

and

$$T_{\mathrm{S}} \geq \mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right] \frac{\ln(\text{target } P\left\{\mathrm{SJF}\right\})}{\ln[0.5/(1+B)]}. \tag{4.2}$$

Similar to chapter 3, we set the target $P\left\{\mathrm{SJF}\right\}$ to be 10^{-5} and simulated the scanning process for 1 million advertisers. With $T_{\mathrm{A}} = 376$ μs, we derived new values $\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$ and T_{S} for each batch size of advertisers using the modified equations to be placed in the automation script.. Utilizing $N_{\mathrm{A}} = 100$ as an example like in the previous chapter, $\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right] = 0.1245$ and $T_{\mathrm{S}} = 2.398$ with the modified equations. This equates to an average of $T_{\mathrm{S}}/\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right] = 19.26$ advertising cycles within a scanning period. Analyzing more closely, we note that in this case only 74% of the advertisers would generate 20 advertisements, with the remaining 26% only generating 19 advertisements. To maximize the number of advertisements per cycle, we increased the scanning period to 2.59 s, raising the amount of advertisements per cycle for all advertisers get at least 20 advertisements and ensuring a higher probability of success. We complete this procedure for every batch size. Table 4.3 displays the batch sizes and the number of repetitions to sum to 1 million advertisers per run, as well as the modified values for $\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$ and T_{S} .

N_{A}	$\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$	T_{S}	Replicas	Runs
100	0.1245	2.59	10000	30
200	0.2503	5.11	5000	30
400	0.5019	10.14	2500	30
800	1.005	20.2	1250	30
1600	2.011	40.32	625	30

Table 4.3: Simulation parameters given the number of advertisers, with modified interadvertisement and scan times

4.4 Simulation Analysis with Modified Interadvertisement Times

A comparison between the independent and dependent results of the $P\{\text{SJF}\}$ for $N_{\text{A}} \in \{100, 200, 400, 800, 1600\}$ via simulations is presented in Table 4.4. The table displays the required scan periods based on optimal interadvertisement times derived from (3.7) and (4.1), with the target $P\{\text{SJF}\}$ set at 10^{-5}. A 95% confidence interval for both independent and dependent results is also given in the table below. From the table, we can see that the dependent results successfully meet the target $P\{\text{SJF}\}$. For instance, at $N_{\text{A}} = 1600$, the $P\{\text{SJF}\}$ equals 4.87×10^{-6}. Even with the upper end of the confidence interval, it comes out to be 8.351×10^{-6}, which is still less than the prescribed target of 10^{-5}. However, the confidence intervals of all dependent results are looser compared to the confidence intervals of the independent results. This is caused by a corresponding reduction in the number of failure events [10].

N_{A}	Independent⌢			Dependent		
	T_{S} (s)	$P\{\text{SJF}\}$	CI [95%]	T_{S} (s)	$P\{\text{SJF}\}$	CI [95%]
100	1.91	4.04e-05	± 1.377e-06	2.59	1.83e-06	± 2.064e-06
200	3.80	5.55e-05	± 9.119e-07	5.11	3.77e-06	± 4.358e-06
400	7.50	6.52e-05	± 9.485e-07	10.14	3.50e-06	± 4.045e-06
800	14.90	7.45e-05	± 6.555e-07	20.20	4.73e-06	± 3.951e-06
1600	29.60	7.74e-05	± 4.542e-07	40.32	4.87e-06	± 3.481e-06

Table 4.4: Comparison of failure probability of scanning job via simulations between the Independent and Dependent collisions

With $\tilde{n}_{\text{AA}}$ as the number of successful scans of an arbitrary advertiser over a scanning period like in the previous chapter, Fig. 4.7 depicts the PMF of $\tilde{n}_{\text{AA}}$ for $N_{\text{A}} = 100$, $\text{E}\left[\tilde{t}_{\text{IA}}\right] = 0.1245$, $T_{\text{S}} = 2.59$. Concerning only with $\tilde{n}_{\text{AA}} < 5$ in the figure, the simulation plot seems to rests on top of the analytical plot, symbolizing that the simulation results are fairly congruent with the analytical results. Figures 4.8-4.11 display similar observation as the previous figure, again illustrating consistency regardless of the size of N_{A}.

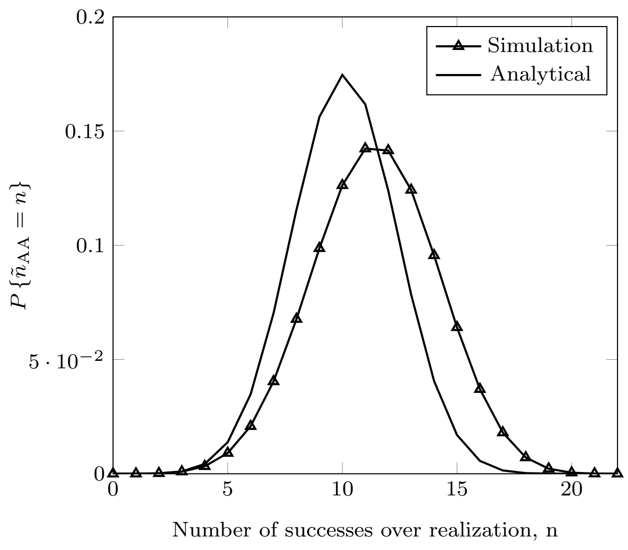


Figure 4.7: PMF of a number of successes from simulation run for batch size of 100 advertisers with a scanning period of 2.59 s and an average advertisement cycle of 0.1245 s

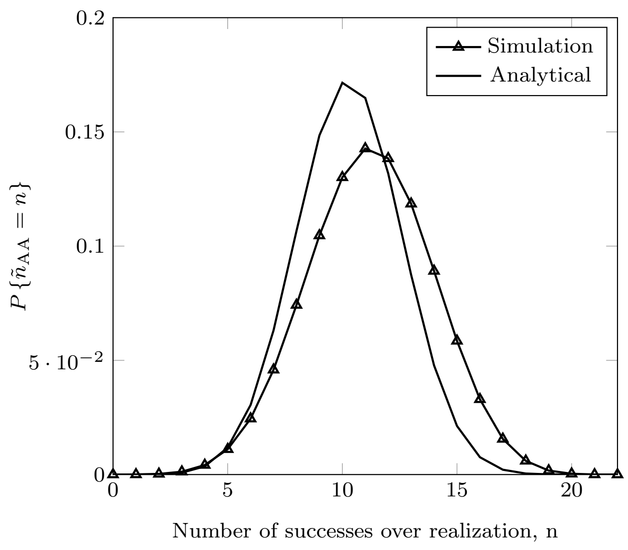


Figure 4.8: PMF of a number of successes from simulation run for batch size of 200 advertisers with a scanning period of 5.11 s and an average advertisement cycle of 0.2503 s

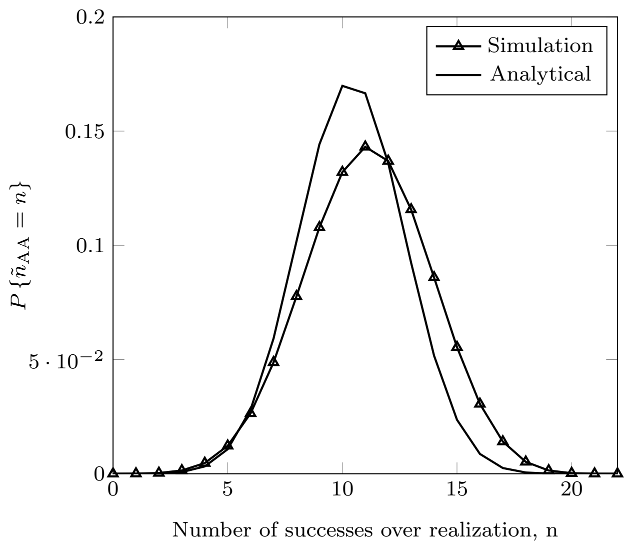


Figure 4.9: PMF of a number of successes from simulation run for batch size of 400 advertisers with a scanning period of 10.14 s and an average advertisement cycle of 0.5019 s

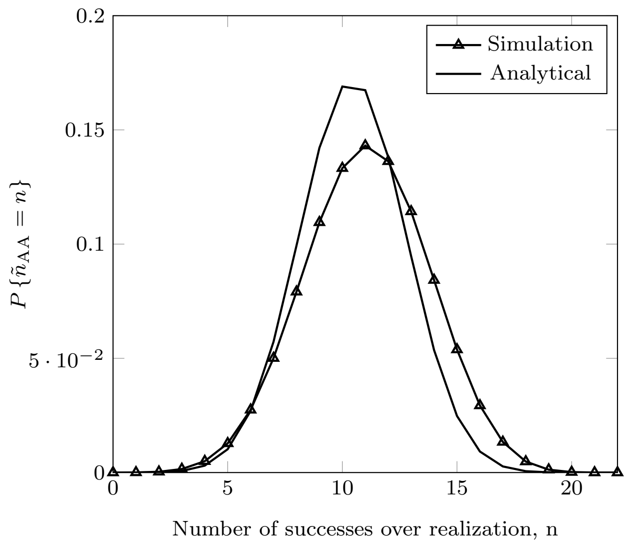


Figure 4.10: PMF of a number of successes from simulation run for batch size of 800 advertisers with a scanning period of 20.2 s and an average advertisement cycle of 1.005 s

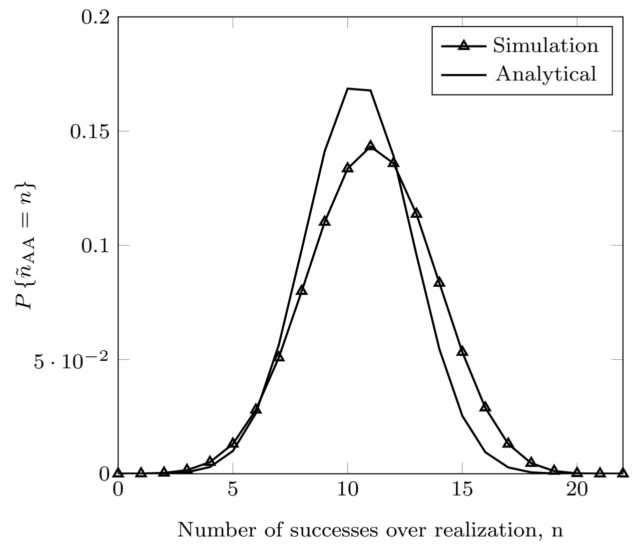


Figure 4.11: PMF of a number of successes from simulation run for batch size of 1600 advertisers with a scanning period of 40.32 s and an average advertisement cycle of 2.011 s

Though it can be clearly seen that correlated collisions effect the $P\{\text{SJF}\}$, there is still uncertainty due to the the difference in scanning times from Table 4.4. Therefore, it is imperative that we clarify if the $\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$ derive from the modified equation is truly optimal.

4.5 Testing Optimality of Interadvertisement Time

In this section, we test optimality of interadvertisement times derive from the modified equation for each batch size. In order to accomplish this, we ran the interadvertisement times, obtained under independent assumption for each batch size, with the scanning times determined under the correlated collision scenario. The purpose of this exercise is to observe if there is a lower probability of SJF when the interadvertisement time is based on correlated collisions. Table 4.5 shows the parameters used in the automation script.

N_A	$\text{E}\left[\tilde{t}_\text{IA}\right]$ (s)	T_S (s)	Replicas	Runs
100	0.107	2.59	10000	30
200	0.216	5.11	5000	30
400	0.433	10.14	2500	30
800	0.867	20.2	1250	30
1600	1.74	40.32	625	30

Table 4.5: Simulation parameters given the number of advertisers, with scanning time under correlated collision scenario and interadvertisement time under independent collision assumption

Table 4.6 displays a comparison between the independent and dependent $\text{E}\left[\tilde{t}_\text{IA}\right]$ times per batch size with 95% confidence interval. Though the dependent $P\{\text{SJF}\}$ exceed the prescribed target $P\{\text{SJF}\}$, the independent $\text{E}\left[\tilde{t}_\text{IA}\right]$ times give slightly better results on average. However, the confidence intervals are larger than the mean average for the results. The reasoning could be the improper amount of runs or replicas. Thus, we reran the simulation parameters again from Table 4.6, but increased the number of replicas for each batch size by 4000. This should reduce the standard deviation of the $P\{\text{SJF}\}$ for each batch size by a power of 2. Table 4.7 shows the new comparison results.

N_A	T_S (s)	Independent			Dependent		
		$\text{E}\left[\tilde{t}_\text{IA}\right]$	$P\{\text{SJF}\}$	CI [95%]	$\text{E}\left[\tilde{t}_\text{IA}\right]$	$P\{\text{SJF}\}$	CI [95%]
100	2.59	0.107	2.13e-06	± 2.568e-06	0.1245	1.83e-06	± 2.058e-06
200	5.11	0.216	2.83e-06	± 2.92e-06	0.2503	3.77e-06	± 4.351e-06
400	10.14	0.433	3.50e-06	± 3.802e-06	0.5019	3.50e-06	± 4.038e-06
800	20.2	0.867	3.40e-06	± 3.90e-06	1.005	4.73e-06	± 3.959e-06
1600	40.32	1.74	4.33e-06	± 4.547e-06	2.011	4.87e-06	± 3.489e-06

Table 4.6: Comparison of failure probability of scanning job via simulations between the Independent and Dependent interadvertisement times with the same scanning time

N_{A}	T_{S} (s)	Independent			Dependent		
		$\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$	$P\{\mathrm{SJF}\}$	CI [95%]	$\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$	$P\{\mathrm{SJF}\}$	CI [95%]
100	2.59	0.107	1.81e-06	± 1.066e-06	0.1245	1.87e-06	± 1.298e-06
200	5.11	0.216	2.85e-06	± 1.448e-06	0.2503	3.23e-06	± 1.380e-06
400	10.14	0.433	3.63e-06	± 1.837e-06	0.5019	3.80e-06	± 1.844e-06
800	20.2	0.867	4.10e-06	± 2.109e-06	1.005	4.32e-06	± 2.376e-06
1600	40.32	1.74	4.18e-06	± 2.106e-06	2.011	4.55e-06	± 2.296e-06

Table 4.7: Comparison of failure probability of scanning job via simulations between the Independent and Dependent interadvertisement times with the same scanning time and $4000x$ the number of original replicas

Shown from Table 4.7, the independent $\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$ times still fair better than the dependent $\mathrm{E}\left[\tilde{t}_{\mathrm{IA}}\right]$ times. However, the confidence intervals are reduced by a significant degree, making them tighter than previous table. Thus, large amounts of replicas per batch size are needed as hypothesized.

4.6 Chapter Summary

In this chapter, the simulation program was modified to investigate the nature of dependence amongst advertisements. Thus new methods and a class were added to help simulate a discrete-Markov chain, which were discussed in detail. Additionally, equations for finding the interadvertisement and scan times were modified and utilized in new parameter settings. New results were shown and compared to results found in the previous chapter. Analysis of the comparison display that the independent collision results are slightly better than correlated collision results when interadvertisement time is based on using the scan time under assumption of correlated collisions were used for both independent and dependent interadvertisement times. This shows that longer scanning times are needed but $P\{\mathrm{SJF}\}$ is not excessively sensitive to interadvertisement time.

““# CHAPTER 5

CONCLUSION

In this thesis, a simulation program was presented in order to examine the probability of scanning job failure conducted by a BLE scanner on a group of items, where each item was outfitted with a BLE advertiser. These items were sorted into large batch sizes and moved to the scanning area. The objective is to determine the optimal interadvertisement time that minimizes the scanning time while achieving a prescribed minimal probability of failure to successfully scan every advertiser. We utilized our analytical model, under the assumption that the advertisement collisions were independent, to set the parameters of our simulation program. Upon noticing inconsistencies between the analytical and simulation results, we decided to explore the nature of dependence in collisions between the target advertiser and the collection of other advertisers.

In order to develop the simulation program, we took great lengths to understand the inner working of the BLE advertising and scanning process. We understood how the BLE advertisers send their advertisements, though not according to a Poisson process. We learned how the scanner operates in either one of two modes: discontinuous and continuous scanning modes. Furthermore, we learned the importance of the adjustable parameters that can affect the performance in the simulation program. We have done extensive analytical modeling and modified and validated our analytical work through the simulation results presented herein.

Though we made some findings concerning the failure probability of a scanning job, there are additional investigations needed for this experiment. First, more analytical work is needed on the probability of successive collisions. In the previous chapter, we arbitrarily chose a number from the transition probabilities to calculate the bias in our modified

““

equations. We would need to do more extensive simulations into the transition probabilities in order to get a more precise value. The simulation results indicated that the PSJF is only slightly sensitive to the interadvertisement time setting over a fairly broad range. For example, for a system with 800 advertisers, optimal interadvertisement times were found to be 0.867 and 1.005 respectively when based on independent and correlated collisions. These interadvertisement times have a percent difference of 14.7%. When the scanning period is based on correlated collisions, the probabilities of scanning job failure are found to be 4.10×10^{-6} and 4.32×10^{-6} with standard deviations of 2.109×10^{-6} and 2.376×10^{-6}, respectively. Thus, while choosing a suitable interadvertisement time is important, the more important issue is minimizing scanning period to achieve the target probability of failure. The minimizing scanning period must be determined on the basis of correlated failures.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Bluetooth Special Interest Group, *Bluetooth Specification Version 5.0 — Vol 6, Part A*, Dec 2016.

[2] G. Shan, B. Lee, S. Shin, and B. Roh, “Design and implementation of simulator for analysis of BLE broadcast signal collision,” in *2017 International Conference on Information Networking (ICOIN)*, pp. 448–452, Jan 2017.

[3] G. Shan and B. Roh, “Advertisement interval to minimize discovery time of whole BLE advertisers,” *IEEE Access*, vol. 6, pp. 17817–17825, 2018.

[4] Jia Liu, Canfeng Chen, and Yan Ma, “Modeling and performance analysis of device discovery in Bluetooth Low Energy networks,” in *2012 IEEE Global Communications Conference (GLOBECOM)*, pp. 1538–1543, Dec 2012.

[5] J. Liu, C. Chen, Y. Ma, and Y. Xu, “Adaptive device discovery in Bluetooth Low Energy networks,” in *2013 IEEE 77th Vehicular Technology Conference (VTC Spring)*, pp. 1–5, June 2013.

[6] M. Ghamari, E. Villeneuve, C. Soltanpur, J. Khangosstar, B. Janko, R. S. Sherratt, and W. Harwin, “Detailed examination of a packet collision model for Bluetooth Low Energy advertising mode,” *IEEE Access*, vol. 6, pp. 46066–46073, 2018.

[7] J. Kim and K. Han, “Backoff scheme for crowded Bluetooth Low Energy networks,” *IET Communications*, vol. 11, no. 4, pp. 548–557, 2017.

[8] M. O. Al Kalaa and H. H. Refai, “Selection probability of data channels in Bluetooth Low Energy,” in *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 148–152, Aug 2015.

[9] C. Julien, C. Liu, A. L. Murphy, and G. P. Picco, “Blend: Practical continuous neighbor discovery for Bluetooth Low Energy,” in *2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 105–116, 2017.

[10] J. N. Daigle, “Report on modeling BLE scanning,” internal report, University of Mississippi, Department of Electrical Engineering, Sept. 2019.

VITA

Born on 1991 in Jackson, MS, George Anthony Humphrey II received a Bachelor's degree in Electrical Engineering at the University of Mississippi on May 2016. During his undergraduate studies, he served in the Mississippi National Guard and deployed to Afghanistan for nine months. In August 2016, he was accepted to graduate school at the University of Mississippi. He is working towards finishing his Master's degree in Electrical Engineering under Dr. John N. Daigle.