



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**USE OF PACKET CAPTURE (PCAP) SOFTWARE FOR
VIRTUAL ACCESS POINT CORRELATION**

by

Corey E. Lutton

March 2019

Thesis Advisor:

John D. Roth

Co-Advisor:

James B. Michael

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE March 2019	3. REPORT TYPE AND DATES COVERED Master's thesis	
4. TITLE AND SUBTITLE USE OF PACKET CAPTURE (PCAP) SOFTWARE FOR VIRTUAL ACCESS POINT CORRELATION			5. FUNDING NUMBERS	
6. AUTHOR(S) Corey E. Lutton				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) Virtual access points (VAP) are a commonly utilized method to broadcast service set identifiers (SSID) with different privilege levels from the same access point (AP). While research has been focused on securing information transmitted using the IEEE 802.11 standard and the authentication of users on wireless local area networks (WLAN), little attention has been given to the security of VAPs utilized on APs to determine whether the presence of a less-privileged, less-secured SSID is a security vulnerability for the AP that hosts it. In this thesis, we collected beacon frames from VAPs hosted on WLAN APs and attempted to correlate VAPs using graph theory and beacon frame characteristics. We discovered that it is possible to correlate beacon frames using the beacon frame timestamp and, to a lesser extent, the received signal strength indicator.				
14. SUBJECT TERMS virtual access point correlation, VAP correlation, IEEE 802.11, wireless, WIFI, graph theory, packet capture, Wireshark, Tshark, Gephi			15. NUMBER OF PAGES 103	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**USE OF PACKET CAPTURE (PCAP) SOFTWARE FOR VIRTUAL ACCESS
POINT CORRELATION**

Corey E. Lutton
Lieutenant, United States Navy
BS, University of Illinois at Urbana-Champaign, 2009

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
March 2019**

Approved by: John D. Roth
Advisor

James B. Michael
Co-Advisor

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Virtual access points (VAP) are a commonly utilized method to broadcast service set identifiers (SSID) with different privilege levels from the same access point (AP). While research has been focused on securing information transmitted using the IEEE 802.11 standard and the authentication of users on wireless local area networks (WLAN), little attention has been given to the security of VAPs utilized on APs to determine whether the presence of a less-privileged, less-secured SSID is a security vulnerability for the AP that hosts it. In this thesis, we collected beacon frames from VAPs hosted on WLAN APs and attempted to correlate VAPs using graph theory and beacon frame characteristics. We discovered that it is possible to correlate beacon frames using the beacon frame timestamp and, to a lesser extent, the received signal strength indicator.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	PROBLEM STATEMENT	1
B.	MOTIVATION AND BACKGROUND	1
C.	BENEFITS OF STUDY.....	2
D.	RESEARCH GOALS AND SCOPE	4
E.	KEY FINDINGS AND CONCLUSIONS	4
F.	THESIS ORGANIZATION.....	5
II.	BACKGROUND	7
A.	IEEE 802.11 WIRELESS STANDARD.....	7
B.	WIRELESS NETWORK COMPONENTS.....	7
C.	IEEE 802.11 STANDARD FOR NETWORK MANAGEMENT FRAMES.....	8
D.	GRAPH THEORY AND ITS APPLICABILITY TO VAP CORRELATION	11
E.	PREVIOUS WORK IN VAP CORRELATION.....	15
III.	EXPERIMENTAL SETUP	21
A.	WIRESHARK/TSHARK	21
B.	GEPHI.....	22
C.	PYTHON	22
D.	EXPERIMENT PREPARATION	23
E.	BEACON FRAME ATTRIBUTES IDENTIFIED AS POSSIBLE CORRELATION FEATURES	23
	1. Beacon Frame Timestamp	23
	2. SSID.....	24
	3. BSSID.....	24
	4. RSSI.....	24
	5. Beacon Frame Reception Time.....	24
F.	TOOLS AND TESTING ENVIRONMENT SET-UP	25
G.	EXPERIMENT PROCEDURES.....	26
H.	EDGE WEIGHTING PROCEDURES.....	28
	1. Timestamp	29
	2. Reception Time.....	29
	3. RSSI.....	30
I.	ANALYSIS OF RESULTING GRAPHS.....	31
J.	ETHICAL CONSIDERATIONS.....	32

IV.	EXPERIMENT RESULTS	33
A.	PREPARATION OF THE GRAPHS IN GEPHI	33
B.	TIMESTAMP METHOD.....	33
C.	RECEPTION TIME METHOD	37
D.	RSSI METHOD	39
E.	CONCLUSIONS	42
V.	CONCLUSIONS AND FUTURE WORK.....	45
A.	FINDINGS AND CONCLUSIONS.....	45
B.	CONTRIBUTIONS OF THIS STUDY	45
C.	STEPS TO REDUCE THE EFFECTIVENESS OF VAP CORRELATION EFFORTS.....	46
D.	RECOMMENDATIONS FOR FUTURE WORK.....	46
	APPENDIX A. TIMESTAMP PYTHON CODE.....	47
	APPENDIX B. RECEPTION TIME PYTHON CODE	61
	APPENDIX C. RSSI PYTHON CODE	69
	APPENDIX D. RSSI GRAPHING AND ANALYSIS PYTHON CODE	77
	LIST OF REFERENCES	83
	INITIAL DISTRIBUTION LIST	85

LIST OF FIGURES

Figure 1.	Beacon frame structure. Source: [8].	9
Figure 2.	Probe request frame. Source: [8].	10
Figure 3.	Probe response frame. Source: [8].	10
Figure 4.	An example of a graph constructed from a three vertex and edge set	11
Figure 5.	Example of communities of vertices with edges between them. Source: [12].	13
Figure 6.	Overview of workflow for the data collection and analysis in this thesis	21
Figure 7.	Wireshark capture interface GUI	25
Figure 8.	Example Wireshark capture of 802.11 information.	26
Figure 9.	Example Tshark command to transfer selected data from a PCAP file to a CSV	27
Figure 10.	Weighted, undirected graph of beacon frames colored by AP and correlated by timestamp	34
Figure 11.	Weighted, undirected graph of beacon frames colored by modularity class correlated by timestamp	34
Figure 12.	Size distribution for beacon frames correlated by timestamp	35
Figure 13.	AP one τ versus reception time	35
Figure 14.	AP two τ versus reception time	36
Figure 15.	AP three τ versus reception time	36
Figure 16.	AP four τ versus reception time	37
Figure 17.	Weighted and undirected graph of beacon frames correlated by reception time.	38
Figure 18.	Size Distribution for beacon frames correlated by reception time	39
Figure 19.	Weighted, undirected graph of beacon frames colored by AP and correlated by RSSI	40

Figure 20.	Weighted, undirected graph of beacon frames colored by modularity class and correlated by RSSI.....	40
Figure 21.	Size distribution of beacon frames correlated by RSSI	41
Figure 22.	Histogram of RSSI for all APs.....	42

LIST OF TABLES

Table 1.	Summary of experiment results	43
----------	-------------------------------------	----

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AITM	announcement indication traffic map
AP	Access Point
BSS	basic service set
BSSID	basic service set identification
CSV	comma separated value
dBm	decibel-milliwatt
DoD	Department of Defense
DoS	denial of service
FN	false negative
FP	false positive
GHz	gigahertz
GUI	graphical user interface
IEEE	Institute of Electrical and Electronics Engineers
IBSS	independent basic service set
IRB	institution review board
LAN	local area network
MAC	media access control
NIST	National Institute of Standards and Technology
OUI	organizationally unique identifier
PCAP	packet capture
QoS	quality of service
RF	radio frequency
RSSI	received signal strength indicator
SSID	service set identification
TN	true negative
TP	true positive
VAP	virtual access point
WIFI	wireless fidelity
WLAN	wireless local area network

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Dr. John Roth, and my co-advisor, Dr. Bret Michael, for their tireless dedication and assistance during this process. Your guidance and direction have inspired me to keep pushing through the problems to break through to the other side.

I thank CAPT Frank B. Ogden and CDR John F. Bradford for teaching me about leadership, attention to detail, overcoming adversity, dealing with setbacks, and most importantly, taking care of your colleagues and subordinates.

I thank RADM James W. Kilby, CAPT Vernon J. Parks Jr., CAPT Stephen D. Barnett, CAPT Thomas J. Dickinson, CAPT Margaret V. Wilson, CAPT Michael Riggins, and CDR Michael J. Herlands for opening the doors of opportunity for me. Without your decision, I would be unable to embark on my exciting future.

Last but not least, I thank my parents, Davy and Vanessa Lutton, for their unwavering and unconditional support while working towards my master's degree. Thank you for being my sounding board and providing me with advice and guidance when I needed it.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

The Department of Defense (DoD) makes extensive use of technology that implements the Institute of Electrical and Electronics Engineers (IEEE) 802.11 wireless networking (WIFI or Wi-Fi) standards. A large body of research exists in literature on the weaknesses and vulnerabilities of transmitted data in wireless local area networks (WLANs). Despite this, there has been little research reported in the open literature into the security of the access points (APs) that the WLAN depend upon to transmit and receive data. It is imperative to better understand the weaknesses inherent in WIFI router technology in order to develop and implement practices to enhance the security afforded by APs, in particular, making APs resistant to an adversary's attempts to penetrate or subvert them.

B. MOTIVATION AND BACKGROUND

WLANs allow many users wireless access over a dispersed area. Use of WLANs has become more commonplace due to the relative ease of set-up and ability to increase network coverage compared to traditionally wired local area networks (LANs). These networks have been a net positive for users and administrators of WLANs since they are able to easily authenticate and connect their mobile devices to the network. However, wireless communications introduce security concerns that need to be addressed.

Efforts to secure WLANs tend to center on authentication of clients wishing to join a network by requiring the client to enter a password and then protecting the confidentiality of the data that is transmitted between the AP and the client via encryption. One type of vulnerability that requires additional investigation arises when an AP uses multiple virtual access points (VAPs) to broadcast multiple service set identifications (SSIDs), and each SSID has a different level of access and protection. A SSID is a string of characters which uniquely identifies a WLAN. For example, in [1] it is explained that an AP can have two VAPs, one broadcasting "My_SSID" for trusted users and the other broadcasting "My_SSID_Guest" for less trusted users. The intention of VAPs is to reduce the number of physical APs required to provide wireless network access. This thesis explores whether the aforementioned practice

allows malicious actors to attack the least-protected SSID. The attacker could use this as a vector to gain access to the physical AP and thereby compromise it.

The exploration of this potential vulnerability is worth addressing in order to begin the discussion of what steps and techniques security professionals can take to mitigate the risks associated with the vulnerability.

C. BENEFITS OF STUDY

A review of DoD and federal government publications regarding the deployment and securing of IEEE 802.11 WLAN technology does not address the potential vulnerability presented by an AP utilizing VAPs to broadcast SSIDs at different authentication and privilege levels. The publication, *A Guide to Securing Networks for Wi-Fi (IEEE 802.11 Family)* published by the U.S. Department of Homeland Security [2] recognizes the following vulnerabilities to trusted networks:

- hidden or rogue APs
- misconfigured APs
- banned devices due to organizational policy
- authorized clients using devices that have accessed unsecured and unmonitored networks
- unauthorized clients using the trusted network
- devices that either share their Internet connection to untrusted device or allow simultaneous connection to trusted and untrusted networks
- unauthorized AP-to-AP associations
- unauthorized peer-to-peer connections
- malicious APs designed to appear as legitimate ones
- denial of service (DoS) attacks against the trusted network

A system administrator could carefully configure and guard a United States Government or DoD trusted network from each of these valid vulnerabilities, but the APs may still be vulnerable. This vulnerability list needs to include a solution to close the exploitation vulnerability for APs broadcasting multiple SSIDs by using VAPs.

The National Institute of Standards and Technology (NIST) publication, *Guide to Securing Legacy IEEE 802.11 Wireless Networks*, takes a more comprehensive approach to WLAN security. NIST's publication states that the common security objectives of a WLAN are: protecting the confidentiality of data on the WLAN, protecting the integrity of data on the WLAN, and ensuring the availability of WLAN resources [3]. The publication further acknowledges that "passive eavesdropping on legacy IEEE 802.11 WLAN communications may cause significant risk to an organization. An adversary can scan radio frequency (RF) signals and capture data traversing the wireless medium. Sensitive information, including proprietary information, network IDs [identifications] and passwords, and configuration data, are some examples of data that may be captured" [3]. These data are not limited to the data exchanged between the AP and a client when the client utilizes the WLAN. It also includes beacon frame data that are transmitted by the AP to permit clients to discover and join the WLAN.

In "Analysis of Security Issues and Their Solutions in Wireless LAN," the authors review passive and active attacks that can be conducted against an IEEE 802.11 WLAN. Under passive attacks, the authors state:

By their nature, wireless LANs intentionally radiates network traffic into space. This makes it impossible to control who can receive the signals in any wireless LAN installation. In the wireless network, eavesdropping by third parties is the most significant threat because the attacker can intercept the transmission over the air from a distance, away from the premise of the company. The attacker monitors wireless data transmissions between devices for message content, such as authentication credentials or passwords[4].

By the very nature of a WLAN's ability to cover a wide area, the same coverage area also provides the potential that a malicious actor can be located outside of the institution's physical boundaries and passively surveil, and capture data being broadcast by the WLAN. Passive surveillance with long standoff distances is an acknowledged problem, but what

has not been fully explored is the data being broadcast by the AP prior to a client joining the WLANs.

To date, no one has published the results of research that would answer the following questions: Do beacon frames present a security vulnerability to APs by allowing a malicious actor/adversary to use widely available tools to correlate a weakly protected SSID to a physical AP? Moreover, which attributes of beacon frames, if any, create an exploitable vulnerability?

D. RESEARCH GOALS AND SCOPE

The goal of the research documented in this thesis is to determine to what extent the use of VAPs present exploitable vulnerabilities. We limit the scope of our research to exploring whether it is possible to use graph-based correlation techniques applied to beacon frames as a reconnaissance tool to aid in the correlation of WLAN APs. Our research addresses the following questions:

- Can graph theory be effectively applied to detect the presence and number of VAPs present on an AP?
- Which beacon frame fields are the best to determine the presence and number of VAPs on an AP?
- What actions can be taken to mitigate a malicious actor's ability to identify and correlate multiple VAPs to an AP?

E. KEY FINDINGS AND CONCLUSIONS

In this thesis, three methods are examined and analyzed to determine their utility for correlating an AP's VAP via beacon frame attributes: the beacon frame timestamp, received signal strength indicator (RSSI), and the reception time of the beacon frames. Of those three methods, the method predicated on using beacon frame timestamps provided the best results. RSSI provided moderate to low correlation ability, and reception time provided negligible correlation ability.

The results of the study indicate that in order for a defender to guard against an adversary's ability to correlate VAPs, the defender should cycle power to the WLAN so that all APs share the same timestamp. Once the timestamp is eliminated as a means of correlation, further correlation of VAPs via RSSI is hampered by multipath interference of the APs in the surrounding area. If the WLAN is located inside a building, the indoor environment is sufficient to mitigate the RSSI method of correlating VAPs.

F. THESIS ORGANIZATION

Chapter II reviews the 802.11 standard, the role of beacon frames, probe requests, and probe responses in a WLAN, as well as basic wireless network components as they relate to this thesis. Chapter II reviews basic graph theory, measuring communities in a graph via modularity, and examines previous work conducted in VAP correlation and how that work is built upon and expanded in this thesis. Chapter III documents the procedure and tools for conducting our experiments with a real-world WLAN. Chapter IV presents the results of the experiments and a comparison of the correlation techniques explored in this study. Chapter V provides conclusions and suggestions for further research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

The purpose of this chapter is to examine previous research that makes VAP correlation to an AP possible. Topics covered here are graph theory and its applicability as well as the IEEE 802.11 standard for wireless beacon frames and the role it plays in AP discovery for a client device.

A. IEEE 802.11 WIRELESS STANDARD

The IEEE 802.11 wireless standard is more commonly referred to as “WIFI” or “WI-FI.” WIFI is a robust and dynamic way for clients to access the wider Internet to send and receive information via the WLAN on which they are operating. The wireless nature of WIFI allows connected clients to be highly mobile within the coverage area provided by the wireless APs and permits greater freedom of movement for connected clients. This makes WIFI an excellent choice in situations where connected clients are likely to move frequently, such as educational institutions and business settings.

IEEE 802.11 has several distinct variants which have been introduced to remedy shortcomings in previous standards, address evolutions in technology, or add improvements to existing standards. In [5], Gast explains that the IEEE 802.11 task groups working on a standard are assigned a lower-case letter, which signifies that the standard is dependent upon the parent standard. The first wireless standard, 802.11, was introduced by the IEEE in 1997 [5]. He further explains that since then, subsequent standards have been introduced to improve upon security and data transmission. According to [6], there are 17 active IEEE 802.11 standards as of this writing. The networks observed in this study are exclusively 802.11n, which [5] explains was the result of an IEEE 802.11 task group that was founded to create a standard that supported high data throughput.

B. WIRELESS NETWORK COMPONENTS

Gast details in [5] that in its most basic form, a WLAN consists of a client, an AP, and a distribution system. A client is any wirelessly enabled device, such as a laptop or cellular phone, that is seeking to connect to a WLAN. He further explains that an AP is the

device which advertises the presence of a network through beacon frames and probe responses as well as serving as a bridge between the distribution system and the client. Finally, the distribution system is used to forward frames to their destination and allow communication between APs to track the movements of mobile clients [5].

The network that is formed by an AP or a group of APs communicating with one another is called a basic service set (BSS) [5], [7]. Initially, APs had the capability to create a single BSS with all users on that BSS having the same privileges [5]. Later, when demand grew for segregating wireless users into trusted and untrusted user groups, APs were developed that could create multiple BSSs simultaneously with different privilege levels possible for each BSS [1].

C. IEEE 802.11 STANDARD FOR NETWORK MANAGEMENT FRAMES

Once a BSS is set up, the APs in the BSS have to utilize management frames to authenticate clients to the WLAN, associate clients to an AP, and provide network services to the clients that are utilizing the WLAN [5]. In [5], the author details 14 management frames used in WLANs: beacon, probe requests, probe responses, independent basic service set (IBSS) announcement indication traffic map (AITM), dissociation, de-authentication, association request, re-association request, association response, re-association response, authentication frames, and action frames. He further explains that these management frames are divided into three classes, with each class handling a different network state. Network state one is where a client is not authenticated or associated with an AP [5]. State two is where a client is authenticated with an AP, but not associated [5]. State three is where a client is both authenticated and associated with an AP and can send data to and receive data from the AP [5].

Gast explains in [5] that class one management frames consist of probe requests/responses, beacon, authentication/de-authentication, and AITM. He further details that these frames can be transmitted at any network state and provide the basic operations used by WLAN clients. In addition, they allow clients to find a WLAN and authenticate themselves to it. Class two management frames consist of association request/response, re-association request/response, and disassociation [5]. They are transmitted only after the

client has authenticated itself with the WLAN and can be used in network states two and three [5]. Class three management frames consist of de-authentication and are used when a client has both authenticated and associated with an AP [5].

Beacon frames are transmitted from an AP at set intervals, which are specified in the beacon interval field [5]. They serve as wireless network management frames that announce the presence of a wireless network and perform an important role in many different network maintenance tasks [5]. Their primary function is to allow client devices to discover a WLAN and transmit the parameters required for joining the network [5]. The structure of a beacon frame, including mandatory and optional fields, can be seen in Figure 1.

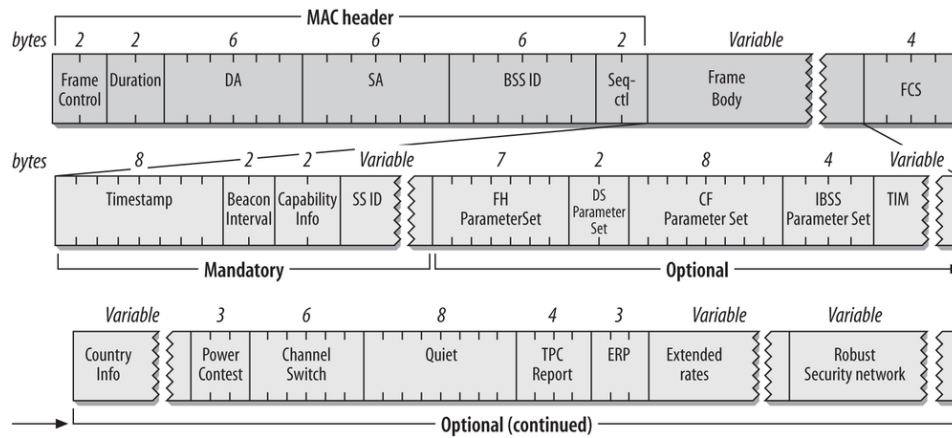


Figure 1. Beacon frame structure. Source: [8].

In [7], there are a combined 68 mandatory and optional fields are specified to be in the beacon frame body. It states that the mandatory fields consist of the beacon frame timestamp, the beacon interval, capability information, the SSID, the supported rates, and BSS membership selectors. Among the optional fields are the basic service set identification (BSSID), the time advertisement, and what the IEEE calls the “vendor specific” field. The BSSID is used to identify different LANs in the same area and is the media access control (MAC) address of the wireless interface in the AP [5]. Finally, the vendor specific field is at the end of the beacon frame and contains whatever data an AP vendor wants to include and does not have a standard format [1], [7].

Clients also utilize probe requests to query for APs in their surrounding area [5]. In order for the client to join the network, the client must support all of the data rates required by the WLAN [5]. The probe request frame structure can be seen in Figure 2.

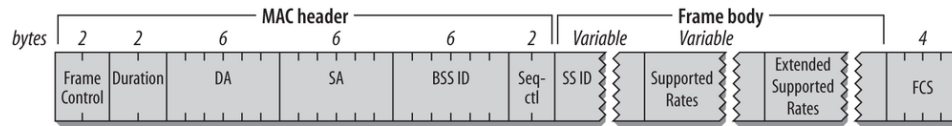


Figure 2. Probe request frame. Source: [8].

If a wireless network receives a probe request that contains compatible parameters, it will send a probe response in return [5]. The AP designated to respond to the client's probe request is the AP that sent the last beacon frame [5]. A probe response carries all of the parameters of the beacon frame which allows a receiving client to match the parameters and join the wireless network, such as beacon interval, timestamp, and sequence number [1], [5]. The probe response frame structure can be seen in Figure 3.

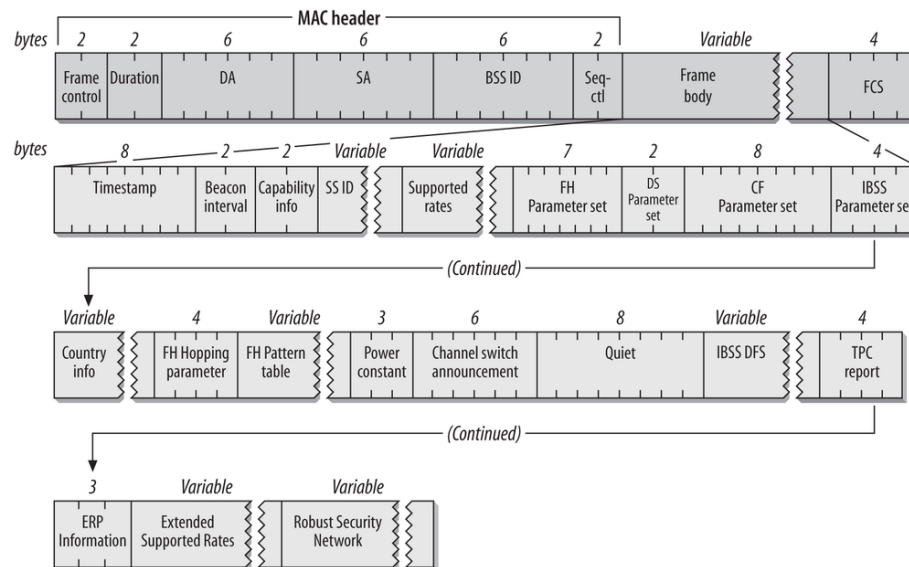


Figure 3. Probe response frame. Source: [8].

Similar to the beacon frame, the probe response also has a vendor specific field where device vendors can place data [5]. The authors in [1] refer to this field as the “vendor specific information elements.”

D. GRAPH THEORY AND ITS APPLICABILITY TO VAP CORRELATION

West defines a graph as “a triple consisting of a vertex set $V(G)$, an edge set $E(G)$, and a relation that associates with each edge two vertices (not necessarily distinct) called its endpoints” [9]. Under this definition, graphing two or more objects of interest as vertices and creating an edge set between them allows a more in-depth analysis by providing an ability to analyze their relationship with each other. This provides a visual model of the relationship between them.

Graphs can be described by an adjacency matrix. In [10], Biggs describes an adjacency matrix as: The adjacency matrix of Γ is the $n \times n$ matrix $A=A(\Gamma)$ whose entries a_{ij} are given by

$$a_{ij} = \begin{cases} 1, & \text{if } v_i \text{ and } v_j \text{ are adjacent} \\ 0, & \text{otherwise} \end{cases}.$$

Here, Γ is used to refer to the graph, which will be referred to as G in this study. Suppose that there is a graph G , with a vertex set $V(G)=\{1,2,3\}$ and an edge set $E(G)=\{12,23,31\}$. Such a graph would look like Figure 4.

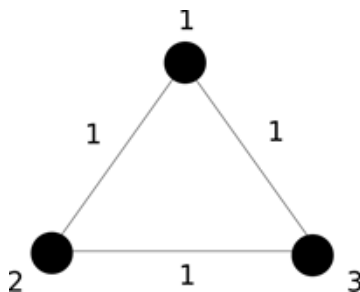


Figure 4. An example of a graph constructed from a three vertex and edge set

The edges depicted in Figure 4 are considered undirected because they do not start at one vertex and end at another. A directed edge would be signified by directed arc pointing to its destination vertex. Because the edge set does not have edges that originate and terminate at the same node (i.e., 11, 22, or 33), it is said to not have any self-loops. Edges in a graph can show a relationship between the connected vertices. A weight can be assigned to an edge with higher values corresponding to a stronger relationship between the vertices. In Figure 4, all of the edges on the edge set have been given an edge weight of 1.

Figure 4 can be represented as $A(G)$ or the adjacency matrix for graph G . An adjacency matrix is constructed by using the rows and columns of the matrix as indices for the graph with “1” representing a connection between those vertices and a “0” if there is no connection between them. For the graph depicted in Figure 4, the adjacency matrix would be constructed as follows:

$$\begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} & \left(\begin{array}{ccc} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{array} \right) \end{matrix}$$

with $A(G)$ taking the form

$$A(G) = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

This adjacency matrix can also be described as symmetric due to the fact that the values in the adjacency matrix are reflected along the diagonal [9].

There are many systems that function as networks and can be analyzed via graph theory. In [11] the authors offer the example of a social network which consists of a network of friendships and acquaintances between individuals. They assert, in such a network, it can be expected that there are communities that reside within the larger interconnected network. In each of these communities, the connections between vertices

may be quite dense. Outside of these communities, there will be fewer connections, or edges, to other communities of vertices. An example of how a community of vertices might look in an undirected graph can be seen in Figure 5.

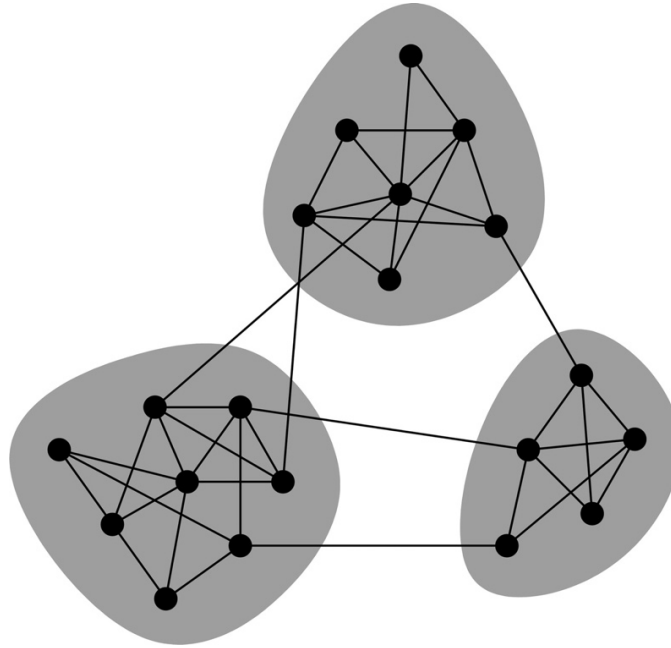


Figure 5. Example of communities of vertices with edges between them.
Source: [12].

This same model can also be used to determine if distinct communities exist within the larger interconnected network. One method of measuring the structure of a graph is called “modularity” [12]. Newman describes modularity as:

This idea, that true community structure in a network corresponds to a statistically surprising arrangement of edges, can be quantified by using the measure known as modularity. The modularity is, up to a multiplicative constant, the number of edges falling within groups minus the expected number in an equivalent network with edges placed at random ... The modularity can be either positive or negative, with positive values indicating the possible presence of community structure. Thus, one can search for community structure precisely by looking for the divisions of a network that have positive, and preferably large, values of the modularity.

The ability to detect individual communities of vertices inside the larger set of vertices allows for finer granularity in determining relationships that exist within those communities.

The authors in [13] provide an example of how modularity for a network is determined. The derivation that follows is from their method in [13]. Let A_{vw} be an element of an adjacency matrix where A_{vw} is 1 if vertices v and w are connected and 0 if not. Suppose that the vertices are divided into communities such that vertex v belongs to community c_v and vertex w belongs to community c_w , these communities are also known as modularity classes. The number of meaningful edges that fall within any community is:

$$\frac{\sum_{vw} A_{vw} \delta(c_v, c_w)}{\sum_{vw} A_{vw}} = \frac{1}{2m} \sum_{vw} A_{vw} \delta(c_v, c_w),$$

where the δ function, $\delta(i, j)$, is 1 if i and j fall into the same community and 0 if they do not [13]. The authors assert that the number of the edges, m , in the graph is determined by:

$$m = \frac{1}{2} \sum_{vw} A_{vw}.$$

The degree k_v of vertex v is defined as the number of edges that connect to vertex v and can be determined by the function:

$$k_v = \sum A_{vw} \text{ [13].}$$

The probability that an edge exists between the vertices v and w if connections are random while still respecting vertex degrees is

$$\frac{k_v k_w}{2m} \text{ [13].}$$

Modularity, Q , can then be defined as:

$$Q = \frac{1}{2m} \sum_{vw} \left[A_{vw} - \frac{k_v k_w}{2m} \right] \delta(c_v, c_w) \text{ [13].}$$

Communities are divided into two at a time, with the goal being to always increase Q [13]. Q can also be negative, which means that the division decreased the modularity measure [13].

The authors in [1] describe why they prefer Newman's modularity measure method. They state that Newman's method enjoys a relative success among other partitioning methods due to the reason that Q provides a measurement of whether the partition was desirable or not. Once Q ceases to be positive, it indicates that any further divisions will negatively impact modularity and a stopping point has been reached. They also indicate that while the modularity measure requires successive trials to determine whether a partition is the most optimum, the fact that there exists a natural stopping point in this method is valuable.

E. PREVIOUS WORK IN VAP CORRELATION

In [14], the authors presented three approaches to utilize WLANs to conduct indoor fingerprint-based localization of a client device. This study was designed to examine and present methods for a client device to accurately determine its position in an area. The authors approached this by correlating VAPs to APs based on their RSSI. The RSSI is a measurement of the received beacon frames or data frames by the client [8]. This was then used to create what the authors in [14] called the APs' fingerprint and they utilized the AP locations as reference points to determine the location of the client device. To do this, the authors divided their WIFI fingerprinting into two phases: online and offline. In the offline phase, they used a client to collect RSSI vectors from the APs serving as reference points and place that information into a database. In the online phase of the study, the client device was used to conduct measurements of the RSSI vector of where the client device is located. The authors then estimated the client device's location by matching the measured RSSI with the closest fingerprinted RSSI in their database.

As a part of the process in [14], the authors needed to correlate VAPs to APs in order to ensure that they were only using one RSSI per AP. They stated that using RSSI vectors from VAPs originating from the same AP increased computational redundancy while doing little to improve their method's location accuracy. The authors accomplished

this by computing the pairwise correlation between the RSSI values amidst the reference APs. If the APs had a high degree of correlation, the authors judged them to be coming from the same AP, and therefore they were categorized as VAPs being broadcast from the same AP.

In [14], the authors considered each AP recognized by the client as a vertex in a graph with an edge existing between vertices if they were calculated to be highly correlated. The resultant graph was then subjected to a clique-finding algorithm the authors devised to seek out the communities of vertices that shared edges, which were merged into a single AP. The RSSI vectors for each vertex for the merged group were then averaged in order to represent their broadcasting AP's RSSI.

The authors of [14] were primarily focused on correlating VAPs to eliminate redundant data when conducting their localization. As a result, the authors' pairwise correlation calculations only take into account the RSSI vectors that were collected during the offline phase of their experiment and does not seek to definitively determine whether the APs that are calculated to have a high degree of correlation are falsely grouped together in their algorithm or not. This study seeks to build upon the previous work of [14] by utilizing RSSI as a method to correlate VAPs. Unlike [14], this study will also determine the reliability of utilizing RSSI as a correlation feature.

In [1], the authors take the VAP correlation findings of [14] and develop it further by refining the application of graph theory to correlate VAPs to an AP and researched other methods of correlating VAPs besides the RSSI method detailed in [14], which can vary based off of wave interference and location. To that end, [1] makes seven propositions on similarity features found in probe responses between VAPs where v_i represents the i th BSSID and a vertex in the graph G . Furthermore, the set of all M observed BSSIDs originating from the same device are represented by

$$V \equiv \{\{v_1\}, \{v_2\}, \dots, \{v_M\}\}.$$

The seven propositions that are proposed in [1] are:

- Proposition one: Vertex v_i is not similar to v_j if $\beta_i \neq \beta_j$, where β_i is the beacon interval set for v_i .
- Proposition two: Vertex v_i is similar to v_j if

$$\left| \tau_i - \tau_j \right| = \Delta \tau_{ij} < \varepsilon_\tau,$$

where

$$\tau_i = \hat{\tau}_i - t_r,$$

$\hat{\tau}_i$ is the timestamp of v_i , t_r is the time of reception, and ε is some small number.

- Proposition three: Vertex v_i is similar to vertex v_j if

$$\left| s_i - s_j \right| = \Delta s_{ij} < \varepsilon_s,$$

where s_i and s_j are sequence numbers attached to probe responses from v_i and v_j respectively and ε is some small number.

- Proposition four: Vertex v_i is similar to vertex v_j if $v_i = v_j$, where v_i is the vendor specific information element of v_i .
- Proposition five: Vertex v_i is similar to vertex v_j if $s_i = s_j$, where s_i is the signature of v_i . The signature being determined by combining the set of information elements in the probe request.
- Proposition six: Vertex v_i is similar to vertex v_j if the middle four octets of the MAC address belonging to v_i and v_j are identical.
- Proposition seven: Vertex v_i is similar to vertex v_j if

$$\left| t_i - t_j \right| = \Delta t_{ij} < \varepsilon_t,$$

where t_i and t_j are the times of reception at client u_k of a probe response from v_i and v_j respectively.

The authors in [1] then propose a method to determine vertex adjacency, or whether two vertices are connected, on graph G . First, the client is placed in a set consisting of all observed clients. This set, which the authors call V , consists of the subset of BSSIDs. V is then used in the expression $V_i^{(K)}$ where each K is an index within the set V on the physical AP and i represents the i th subset of BSSIDs originating from the same physical AP. The authors assert that if $K > I$, then each element of set V is considered a VAP. They continue to assert that if $K = I$, then there is only one BSSID associated with the AP and therefore the singular BSSID is not a VAP.

The following is a summary of the derivation used in [1] to determine vertex adjacency. The authors of [1] state that upon the reception of a probe request from a client, the VAPs on the AP will send K probe responses

$$\forall v_i \in V.$$

This will create

$$\begin{pmatrix} K \\ 2 \end{pmatrix}$$

edges between the vertices and the graph G . The structure of G is represented by adjacency matrix A which [1] defines as:

$$A = \begin{bmatrix} 0 & \omega_{1,2} & \dots & \omega_{1,M} \\ \omega_{2,1} & 0 & \dots & \omega_{2,M} \\ \vdots & & \ddots & \vdots \\ \omega_{M,1} & \omega_{M,2} & \dots & 0 \end{bmatrix},$$

where

$$\omega_{i,j} = \omega_{j,i} \geq 0$$

if and only if v_i is adjacent, or connected, to v_j and

$$\omega_{i,j} = 0$$

if they are not [1].

In [1], they further define the multidimensional graph $\hat{G}$, where each dimension is a feature being evaluated, for each of the similarity metrics via the product function

$$\hat{G} = \prod_k A_k.$$

With each adjacency matrix A representing another dimensional space of the graph [1].

The authors choose to view A_k as multiple dimensions of the overall graph $\hat{G}$. They project the information from each of the dimensions in $\hat{G}$ onto a single dimension, represented by graph G , by using the mapping function

$$f : \hat{G} \rightarrow G.$$

This function f is defined by the authors as their voting function that defines the voting weight for each of the K features based on the amount of information that particular feature carries.

The authors of [1] posit the final G that is generated is an unweighted, undirected, symmetric graph. They seek to determine communities within the overall graph, VAPs being broadcast by the same physical AP, by utilizing the work in [13] to divide their vertices into separate communities. This results in communities consisting of the estimated BSSID relationships thereby creating an estimation of which VAPs are correlated to one another [1].

The study conducted in [1] advances the features to correlate VAPs, but the authors focus only on the examination of the information contained in probe responses, which are dependent upon a client device sending a probe request. The results produced a smaller data set than what would have been provided if they had examined beacon frames, which

are regularly broadcast by APs regardless of whether client devices are present or not. This thesis seeks to examine beacon frames using selected features from [1] to determine if it is possible to correlate VAPs using only the information contained in an AP's beacon frame.

III. EXPERIMENTAL SETUP

This chapter reviews the experimental tools and set-up used for this study as well as the procedures to parse and analyze the data, as seen in Figure 6. The goal of designing the experiment and selecting the tools used for the collection and analysis of data is to simulate the ability of a real-world malicious actor to collect and correlate VAPs.

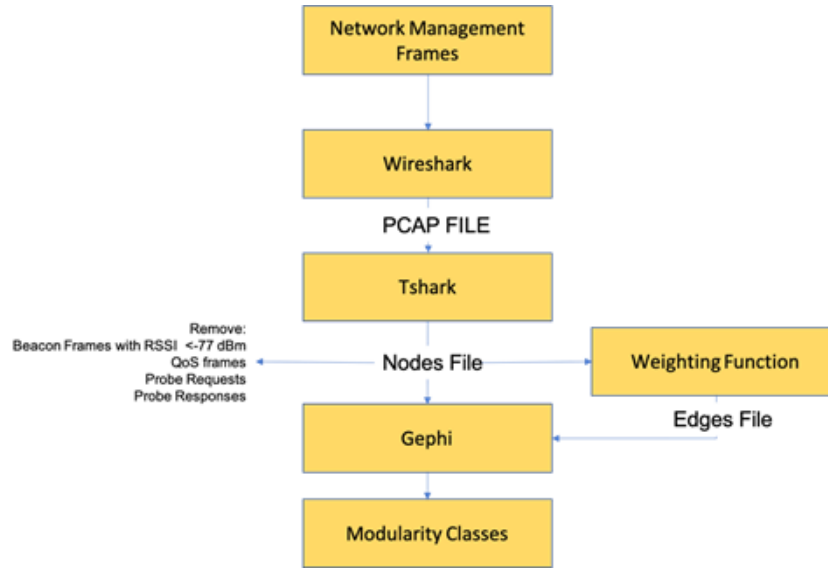


Figure 6. Overview of workflow for the data collection and analysis in this thesis

A. WIRESHARK/TSHARK

Wireshark version 2.6.3 is a widely used packet capture (PCAP) application that allows for fine-grain analysis of packets and protocols that are being transmitted on a network. It supports many different file capture formats and allows great flexibility in analysis of the captured packets [15]. Wireshark is used in this experiment due to its wide availability, ease of use, and its ability to export selected fields from the packet capture session into a comma separated value (CSV) document.

Tshark version 2.6.3 is network protocol analyzer accessible from the command line. It functions much like Wireshark in its ability to capture and display data from a network or read packets from a saved capture file. Tshark will use the PCAP library to capture network traffic from the first available network interface on the capture device and display a capture summary line via standard output for each packet that is captured [16]. Tshark was utilized in this study to extract selected data fields from the PCAP file generated by Wireshark to a CSV file for further analysis. Wireshark does not have the ability to export selected data fields or the ability to export data in a format other than pcap and pcapng.

B. GEPHI

Gephi version 0.9.2 is an open-source network visualization tool for generating three-dimensional rendering of large networks to visualize the relationships between nodes in the network. It allows users to import data and immediately visualize, manipulate, and filter it to better render relationships between the network nodes. Users can utilize the graphical user interface (GUI) to calculate graph statistics, including graph modularity [17]. The graph modularity measure function in Gephi utilizes the algorithm outlined in [18] to determine the graph modularity and the number of vertex communities present in the graph. Users can import graph nodes and edges in separate CSV files and export the final graph file containing Gephi-generated graph statistics.

C. PYTHON

Python is a high-level programming language with many different third-party plug-in modules designed for data manipulation and evaluation [19]. In this experiment, the following modules were utilized for visualizing, parsing, and manipulating the data exported from Wireshark: Matplotlib, Pandas, Random, Datetime, Decimal, Itertools, Functools, Seaborn, and Numpy. In addition, Python was utilized to calculate the recall, precision, and F-Scores of all of the correlation methods examined during this study. All of the software developed for use in this study is included in Appendix A, Appendix B, Appendix C, and Appendix D.

D. EXPERIMENT PREPARATION

The experiment was conducted at a location with an enterprise-level WLAN that supports thousands of mobile users on a daily basis. The location's information technology department provided permission to conduct the experiment. The department's assistance was necessary to understand the placement of APs in the data collection area. This information also provided the ground truth of how the observed VAPs correlated to physical APs, facilitating the creation of a test oracle for gauging the effectiveness of the correlation techniques explored in this thesis.

Data collection takes place while the collector is stationary inside of a room in a five-story building with windows. On the floor the data collection was conducted there are nine APs. Of those, beacon frames for four APs were collected at sufficient RSSI to analyze for this study.

E. BEACON FRAME ATTRIBUTES IDENTIFIED AS POSSIBLE CORRELATION FEATURES

While all of the mandatory and optional fields in a beacon frame contain data that are needed by a client attempting to connect to the network, only a relatively few contain information that is useful in correlating the broadcasted VAP to a physical AP. The fields identified in this study as possessing potential for VAP correlation and being evaluated by this study are: timestamp, SSID, BSSID, RSSI, and the beacon frame reception time. These attributes were selected based on the results presented in [1] and [14]. In addition, an analysis of the 68 mandatory and optional fields of a beacon frame did not reveal any further attributes which would provide useful correlation information.

1. Beacon Frame Timestamp

The timestamp field is a 64-bit counter in a beacon frame that allows for synchronization between the APs in a BSS, as explained in [5]. The timestamp contains the time in microseconds that an AP has been active [5].

2. SSID

In [1], the authors explain that the SSID being broadcast is valuable from a classification standpoint. In the event of an AP broadcasting several SSIDs using VAPs, the SSID may be used to convey the intended use or user of the VAP. For example, “My_SSID” and “My_SSID_Guest” could convey that one SSID is for trusted clients, while the other is intended for less trusted clients. This feature is less helpful in a scenario where the network administrator has named SSIDs in a less intuitive, more arbitrary way.

3. BSSID

Because the BSSID is the MAC address of wireless interface for the AP, then the normal conventions for MAC address composition apply [5]. For example, the organizationally unique identifier (OUI), which consists of the first three octets of a MAC address, is assigned to a manufacturer, company, or vendor of the equipment [20]. In [1], the authors found that the second and third octet can also be used as an indicator that two VAPs are hosted on the same AP. Furthermore, they found evidence that some AP manufacturers will keep the middle four octets of BSSIDs of correlated VAPs the same and increment the last octet.

4. RSSI

The RSSI, measured in decibel-milliwatt (dBm), provides a measurement of how strong or weak the received signals are for the client [6]. Because there is a roughly inverse relationship between RSSI and distance to the AP, beacon frames from VAPs that are originating from an AP further away should have a weaker RSSI than beacon frames from VAPs originating from an AP that is closer.

5. Beacon Frame Reception Time

The beacon frame reception time is the time at which Wireshark receives the beacon frame. In [1], the authors found promising evidence that probe responses received closely together by the client came from VAPs on the same AP. It should follow then that beacon frames from VAPs that arrive within a certain time period are correlated.

F. TOOLS AND TESTING ENVIRONMENT SET-UP

Prior to initiating the PCAP session, Wireshark was set up to capture IEEE 802.11 wireless AP beacon frames. This was done by selecting the correct interface through the Wireshark interface GUI as seen in Figure 7. Wireshark was then placed in ‘monitor mode’ to display IEEE 802.11 protocol management information.

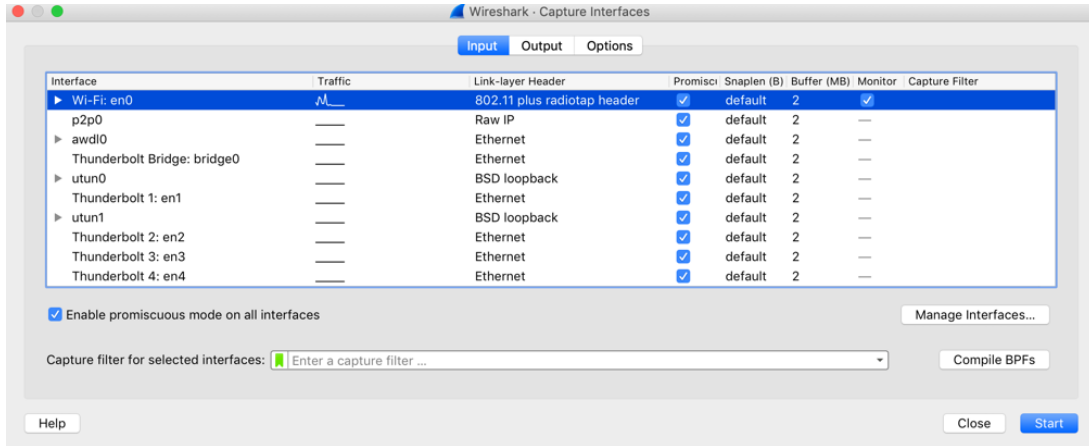


Figure 7. Wireshark capture interface GUI

‘Monitor mode’ ensured that the SSID filter was disabled and packets from all SSIDs in the area from the currently selected channel were displayed [21]. Once PCAP capture commences, Wireshark displays captured packet data in the GUI capture window, as seen in Figure 8.

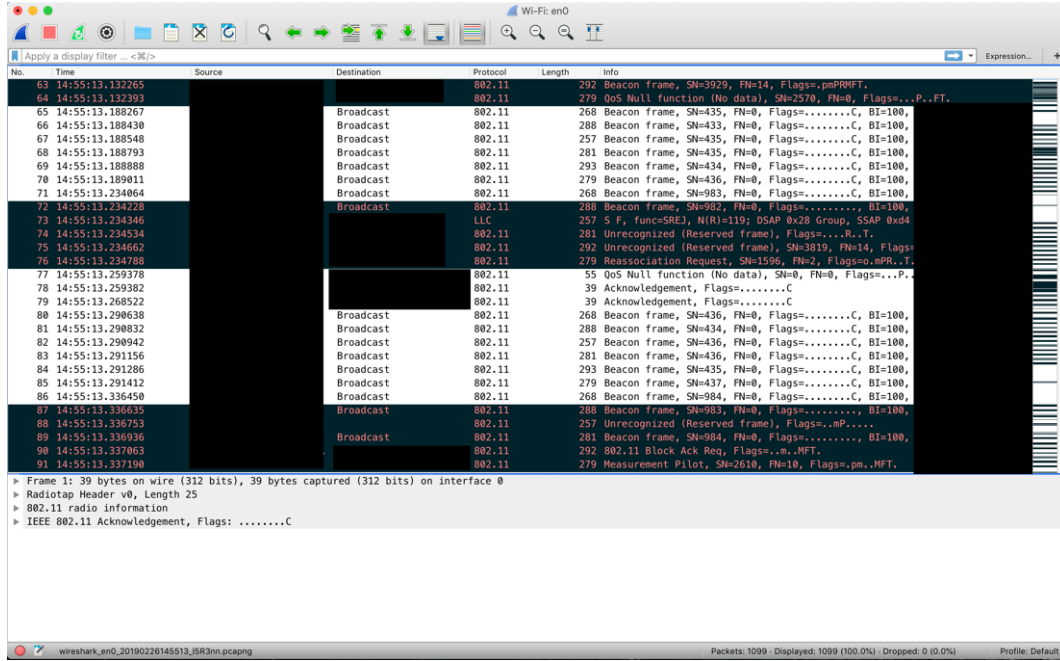
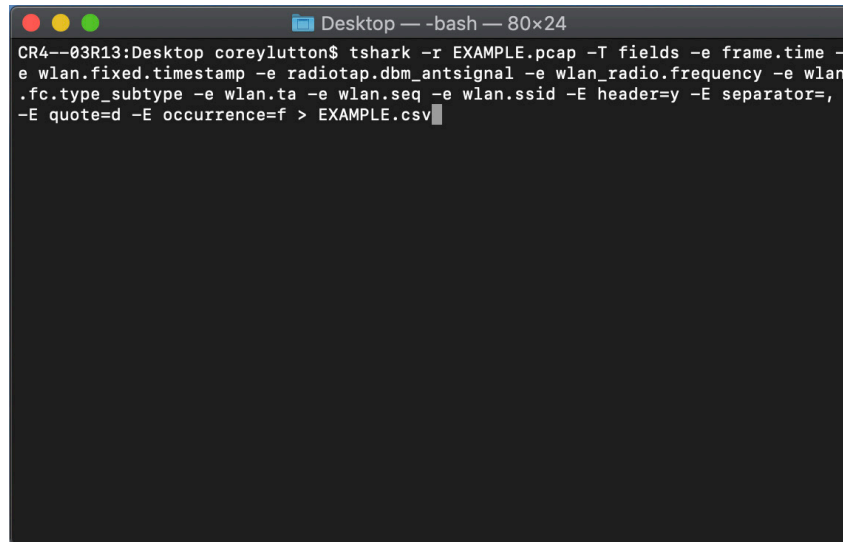


Figure 8. Example Wireshark capture of 802.11 information

G. EXPERIMENT PROCEDURES

Prior to the commencement of this experiment, we conducted a rudimentary parametric study to determine the amount of time needed to collect sufficient beacon frames to build the graph. From this study, we determined that 10 to 15 minutes provided an adequate number of beacon frames for the purposes of this experiment. For the data collection used in this study a single collection trial was conducted for a period of 611 seconds. In that period, 35,810 network management frames were captured, which included beacon frames as well as probe requests, probe responses, and quality of service (QoS) data packets. These data were saved in a PCAP file in order to commence data field extraction via Tshark. Data extraction via Tshark was executed from the command line interface, as seen in Figure 9.



```
CR4--03R13:Desktop coreylutton$ tshark -r EXAMPLE.pcap -T fields -e frame.time -  
e wlan.fixed.timestamp -e radiotap.dbm_antsignal -e wlan_radio.frequency -e wlan  
.fc.type_subtype -e wlan.ta -e wlan.seq -e wlan.ssid -E header=y -E separator=,  
-E quote=d -E occurrence=f > EXAMPLE.csv
```

Figure 9. Example Tshark command to transfer selected data from a PCAP file to a CSV

The options used in this command consisted of ‘-r’, ‘-T’, ‘-e’, ‘-E’, and ‘>’. The website, [16], details the purpose of all of these options. It states that the ‘-r’ option is used to have Tshark read from the PCAP file. The option ‘-T’ followed by ‘fields’ is used to indicate that user-specified fields that are being extracted from the PCAP file. The option ‘-e’ is used to preface each specific Wireshark data field that is desired. The ‘-E’ option allows for user-controlled printing of the selected fields and for the use of formatting options to control how the data will be displayed in the resulting file. ‘Header’ was set to ‘y’ to enable a field header to be displayed in the CSV file, ‘separator’ was set equal to ‘,’ to insert commas in between values and create the file as a CSV. The argument ‘quote’ is set to ‘d’ to cause double quotes to surround the selected fields and ‘occurrence’ is set to ‘f’ to select the first occurrence in the event that there are multiple occurrences for a field. Finally, the ‘>’ option was used to write the resulting data to a CSV file [16].

After extracting the fields to the CSV, the resulting data were further refined by removing all extraneous network management frames. Any beacon frames with RSSIs less than -70 dBm were removed in order to refine the data set to beacon frames that were received by the collecting client at sufficient signal strength for analysis. The headers for the columns in the CSV were also renamed to “Reception Time,” “Timestamp,” “dBm,”

“Frequency,” “Frame Type,” “MAC Address,” “Sequence Number,” and “SSID.” An additional column, called “ID” was inserted into the CSV to the left of the “Reception Time” column and was numbered in sequential order to provide each beacon frame with a unique identification number. The data set was reduced to 5001 beacon frames to lighten the computation load while maintaining a sufficient number of beacon frames to observe trends.

Initially a Dell XPS13 laptop was used for data collection and analysis. While the Dell laptop collected both network management frames on 2.4- and 5-gigahertz (GHz) frequencies, the number of beacon frames was relatively low for the period the collection time. Once the experimental procedure was finalized, experimentation took place on a MacBook Pro. The MacBook Pro was only able to display network management frames on the 5 GHz frequency. However, it was able to collect a greater number of beacon frames.

Despite reducing the number of beacon frames being analyzed to 5001, this thesis encountered data-analysis issues. Gephi was unable to import the edges files for the RSSI and reception time experiments due to the size of the files being too large and causing Gephi to terminate. Once the number of beacon frames was reduced to 4501, the edge files became sufficiently small enough for Gephi to import and display.

H. EDGE WEIGHTING PROCEDURES

Python scripts were written to examine possible correlations between VAPs using the following beacon frame attributes: timestamp, reception time, and RSSI. In each case, the panda’s library was used to import the beacon frame data from the CSV into a data frame for analysis. The identification numbers for the beacon frames were placed into two separate data frames, one serving as the edge’s origin and the other as the edge’s destination, to create unweighted edges between the beacon frames. Because itertools created edges between every beacon frame, care was taken to ensure that self-loops were not generated.

1. Timestamp

The timestamps for all the beacon frames collected were found to increase without differentiation between APs. A possible explanation for this is that the building where the observation took place may have had its power cycled. This would mean that power to all APs is restored simultaneously and their timestamps start at the same time. In an effort to test the validity of utilizing timestamp as a correlation feature, a Python program was written to subtract a random number between one and 10,000,000 based off of the BSSID for the VAPs, simulating a unique timestamp for each AP observed. The reception time of the beacon frame in microseconds, $T_{reception}$, was subtracted from the unique timestamp to account for the incrementing of the timestamp as time passed, as shown in the following equation:

$$\tau = |Timestamp - T_{reception}|.$$

The value τ was placed into a data frame to pass to the edge weighting function.

The edge weighting function takes the unique τ for the first beacon frame, τ_1 , and the second τ , τ_2 , and performs the following function:

$$\Delta\tau = |\tau_1 - \tau_2|.$$

If $\Delta\tau$ was less than or equal to 500 microseconds, the edge would be given a weight of one. Otherwise, it would be given an edge weight of zero. The time period of 500 microseconds was chosen to allow for small variations in the timestamps that may arise from the unknown accuracy of the collection equipment.

2. Reception Time

The reception time for receiving the beacon frame, as reported by Wireshark, was parsed and turned into a whole number representing the time in microseconds that the beacon frame was received. The edge weighting function selected the first reception time, RT_1 , and the second reception time, RT_2 , and performed the following function:

$$\Delta RT = |RT_1 - RT_2|.$$

The edge weight, W , was determined by the following rule:

$$W = \begin{cases} 1, & \text{if } x_0 < \Delta RT \leq x_1 \\ 3/4, & \text{if } x_1 < \Delta RT \leq x_2 \\ 1/2, & \text{if } x_2 < \Delta RT \leq x_3 \\ 0, & \text{otherwise} \end{cases}.$$

The values were chosen to weight edges strongly if they were received very closely together and to weight edges less strongly if they were received farther apart.

3. RSSI

The first RSSI, $RSSI_1$, and the second RSSI, $RSSI_2$, were placed into the following function:

$$\Delta RSSI = |RSSI_1 - RSSI_2|.$$

In order to determine edge weight, $\Delta RSSI$, was placed into another function:

$$W = \frac{0.15}{\Delta RSSI},$$

where 0.15 is a tuning constant and W is the resulting edge weight. The value of the numerator was determined experimentally by plotting the resulting edges in Gephi and calculating the modularity measure of the graph. This weight function was used to assign a greater edge weight to beacon frames with RSSIs close to one another. If $\Delta RSSI$ was greater than a cutoff value of 1dBm, represented by x , then the edge would be assigned a weight of zero. This value was selected to prevent assigning edge weights to beacon frames that were likely originating from different APs.

I. ANALYSIS OF RESULTING GRAPHS

The edges along with their assigned weights were then written into another CSV file. The CSV file containing the beacon frame data was then loaded into Gephi as a nodes table and the edge weight CSV was loaded as an edges table. Gephi would automatically remove the edges that were assigned a weight of zero and only graph the edges that had a value greater than zero assigned to them. This data was then graphed in the overview window and the built-in modularity measure feature was utilized to determine the number of modularity classes in the graph and which modularity class each beacon frame was assigned.

As an additional measure, the data was run through a Python program to determine the number of true positives (TPs), false positives (FPs), true negatives (TNs), and false negatives (FNs). TPs were defined as beacon frames that were from the same AP and had an edge connecting them. TNs were defined as beacon frames that were from different APs and did not have an edge connecting them. FPs were defined as beacon frames that were from different APs and had edge connecting them. FNs were defined as beacon frames that were from the same AP and did not have an edge connecting them.

TPs, FPs, TNs, and FNs were calculated by dividing the edges into two groups, one assigned an edge weight greater than zero, and the other assigned an edge weight of zero. TP, FP, TN, and FN were counted according to the following rule:

If $W \neq 0$:

TP iff Beacon Frame₁ and Beacon Frame₂ come from the same AP

FP iff Beacon Frame₁ and Beacon Frame₂ come from different APs

If $W = 0$:

FN iff Beacon Frame₁ and Beacon Frame₂ come from the same AP

TN iff Beacon Frame₁ and Beacon Frame₂ come from different APs

where membership to the same AP was determined by the ground truth obtained from the technology department.

In order to determine the effectiveness of each method, the accuracy, recall, and precision were calculated based off the number of TPs, TNs, FPs, and FNs present in the graph. Accuracy is the ratio of fraction of correctly classified beacon frames out of every single beacon frame in the data set. Accuracy is calculated via the following formula:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}.$$

Recall is defined as the fraction of correctly correlated beacon frames over all beacon frames that were actually in that group. Recall is calculated via the following formula:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

Precision is defined as the fraction of correctly correlated beacon frames over the total of beacon frames that were classified in that group. Precision is calculated via the following formula:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

F-score, also known as F1 score, is the scaled geometric mean of both precision and recall. A score of one means perfect precision and recall, while zero means it provides neither precision nor recall. F-score is calculated via the following formula:

$$\text{F-Score} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}.$$

J. ETHICAL CONSIDERATIONS

Prior to commencing this experiment, a Human Subject Research Determination Request was submitted to the institution review board (IRB) along with the proposal for this study. The determination of the IRB was that since this study was collecting information that is continuously broadcast by APs and wireless users during normal WLAN activity, the study did not fall under human research restrictions.

IV. EXPERIMENT RESULTS

This chapter presents the results of each of the methods proposed in this study and reviews their accuracy, precision, recall, and F-Score. In addition, the graphs and modularity measures generated by Gephi using data from the proposed methods are reviewed.

A. PREPARATION OF THE GRAPHS IN GEPHI

Following the edge weighting method, the edges were written to a separate CSV file. The beacon frame file was loaded into Gephi as a nodes table and the CSV with the edge weights was loaded as an edges file. This resulted in a weighted, undirected graph in Gephi. The graph was then displayed as a Fruchterman Reingold graph and the vertices were colored by AP. AP one was colored blue, AP two was colored red, AP three was colored green, and AP four was colored yellow. The Fruchterman Reingold graph was chosen because it displays all communities, both connected and unconnected, making visual interpretation of the graph easier. After exporting the graph from Gephi, the modularity measure in Gephi was selected to calculate the modularity of the graph as well as the number of communities present in the graph. The vertices were then recolored by modularity class and exported from Gephi.

B. TIMESTAMP METHOD

The timestamp method had an accuracy of 0.900, precision of 1.0, a recall of 0.639, and a F-Score of 0.779. The high F-Score due to this method having a high number of TP (4431650) and TN (18064444) as well as zero FP and a relatively small amount of FN (2508906). This resulted in the weighted, undirected graphs seen in Figure 10 and Figure 11.

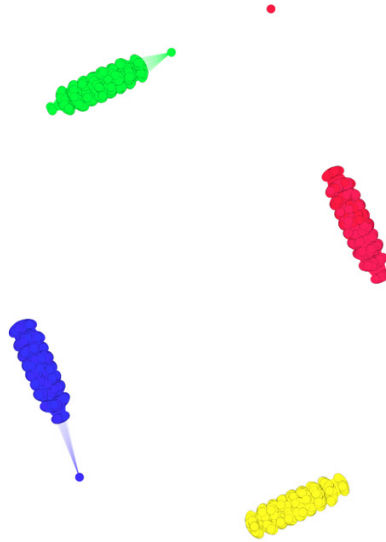


Figure 10. Weighted, undirected graph of beacon frames colored by AP and correlated by timestamp

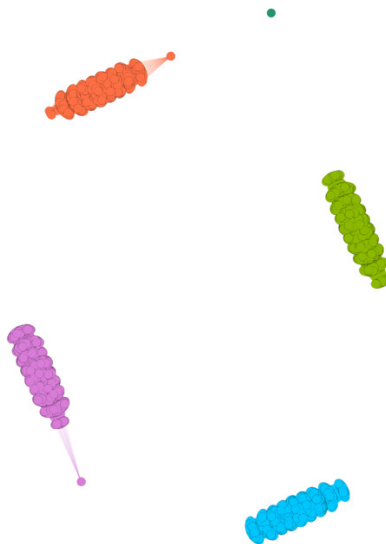


Figure 11. Weighted, undirected graph of beacon frames colored by modularity class correlated by timestamp

The modularity measure function in Gephi calculated that the modularity of the graph was 0.661 with a total of five communities present. Figure 12 depicts the size distribution of the number of vertices, or nodes, within each modularity class.

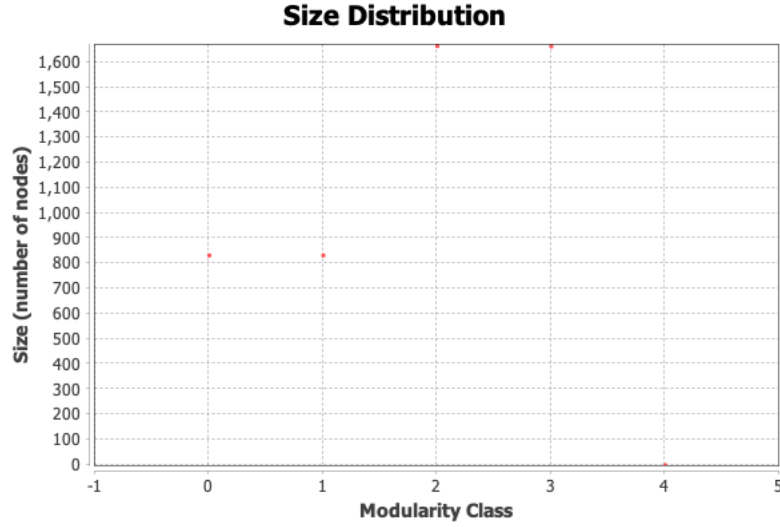


Figure 12. Size distribution for beacon frames correlated by timestamp

In order to determine why AP two had a small community of vertices unconnected to the much larger community, τ versus reception time was graphed for each AP. Due to the large number of beacon frames from each AP, only one in nine data points were graphed for Figures 13 through 16.

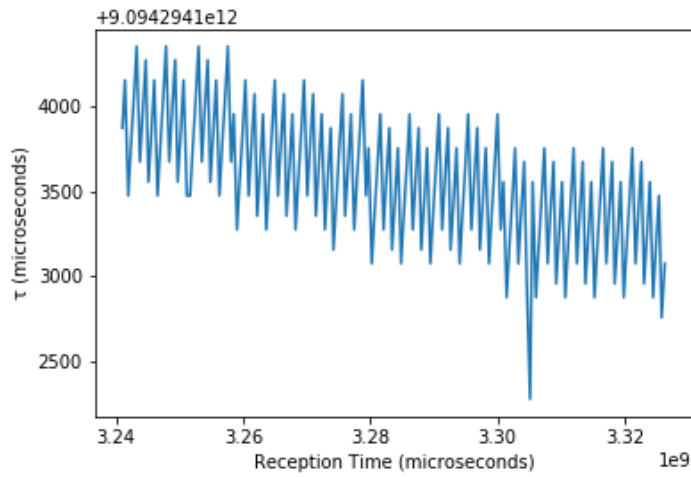


Figure 13. AP one τ versus reception time

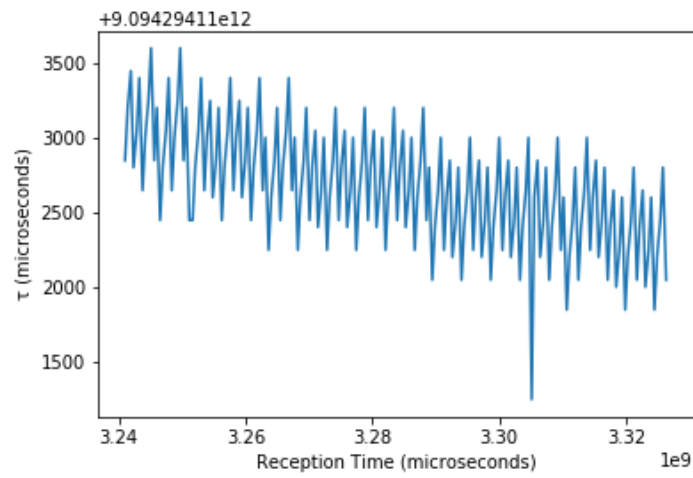


Figure 14. AP two τ versus reception time

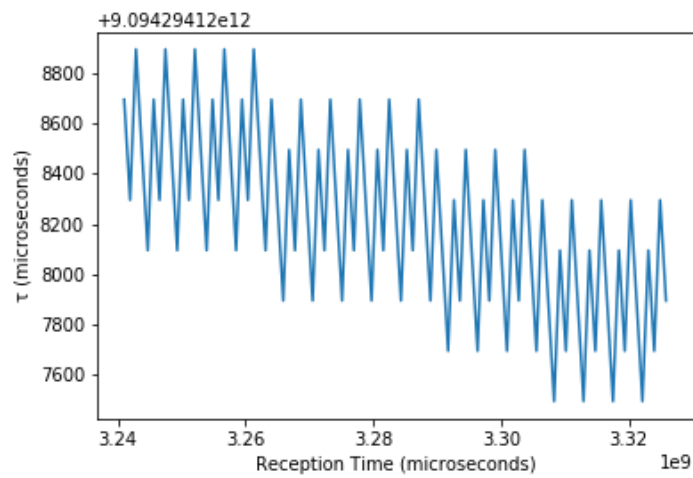


Figure 15. AP three τ versus reception time

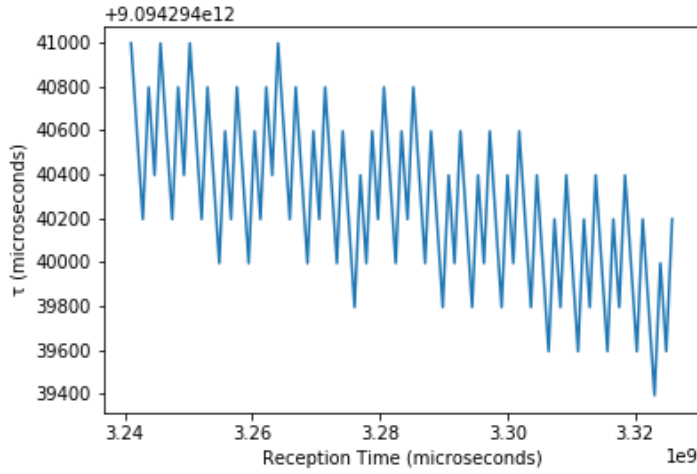


Figure 16. AP four τ versus reception time

The large spike for AP two occurring after 3300000000 microseconds accounts for this small community of beacon frames. This is because the weighting function for the timestamp method creates a weighted edge only if $\Delta\tau$ for two beacon frames is within 500 microseconds. The spike just after 3300000000 microseconds is greater than 500 microseconds apart from the nearest timestamp. There is a similar spike for AP one after 3300000000 microseconds, however, it was within 500 microseconds of the nearest timestamp. The jitter observed in all four figures is possibly the result of inaccuracies introduced by the times recorded by Wireshark,

C. RECEPTION TIME METHOD

The reception time method had an accuracy of 0.722, precision of 0.133, a recall of 0.0005, and a F-Score of 0.0009. The reason the accuracy is relatively high while the F-Score is poor is due to the method's high number of TN (18042782) combined with its TP (3330) generated by the edge weighting function. The resulting weighted and undirected graph as seen in Figure 17.

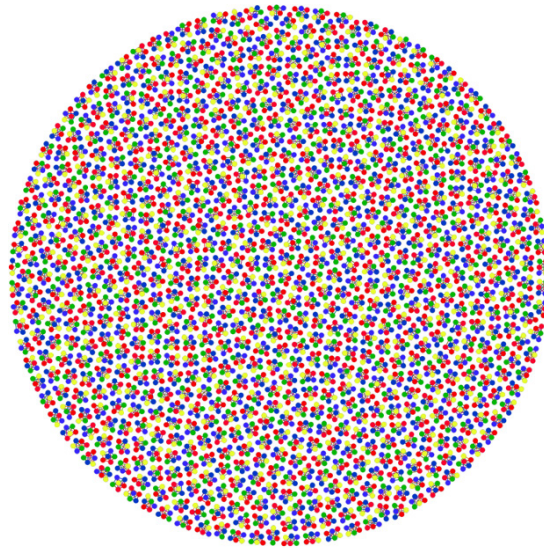


Figure 17. Weighted and undirected graph of beacon frames correlated by reception time

When the modularity measure function was initiated in Gephi, the program calculated that the modularity of the weighted undirected graph was 0.999 and that there were 5,252 communities present. This was due to the edge weighting function assigning large weights to beacon frames received closely together and small or no weights to beacon frames received further apart, thereby creating many small communities of beacon frames. As a result, Gephi calculated 5,252 communities with a modularity of 0.999. The size distribution within each modularity class can be seen in Figure 18.

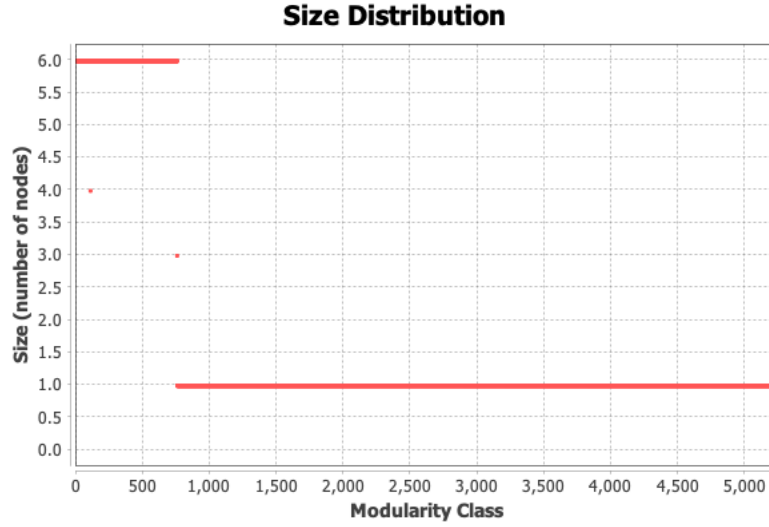


Figure 18. Size Distribution for beacon frames correlated by reception time

D. RSSI METHOD

The RSSI method had an accuracy of 0.506, precision of 0.284, a recall of 0.512, and a F-Score of 0.365. Much like the reception time method, the RSSI method was negatively impacted by its high number of FP (8968134) and FN (3389020) generated by the edge weighting function. The RSSI method's F-Score was improved compared to the reception time method due to the larger amount of TP (3551536) and TN (9096310). This resulted in the weighted, undirected graphs seen in Figure 19 and Figure 20.

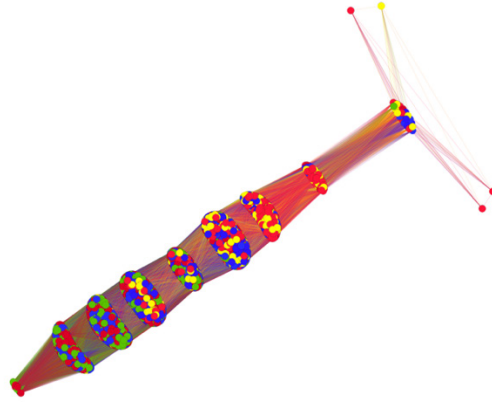


Figure 19. Weighted, undirected graph of beacon frames colored by AP and correlated by RSSI

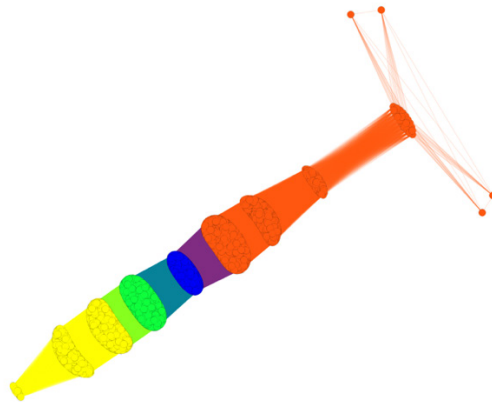


Figure 20. Weighted, undirected graph of beacon frames colored by modularity class and correlated by RSSI

The modularity measure function in Gephi calculated that the modularity of the graph was 0.549 with a total of four communities present. The size distribution of the number of vertices, or nodes, within each modularity class for the RSSI method is seen in Figure 21.

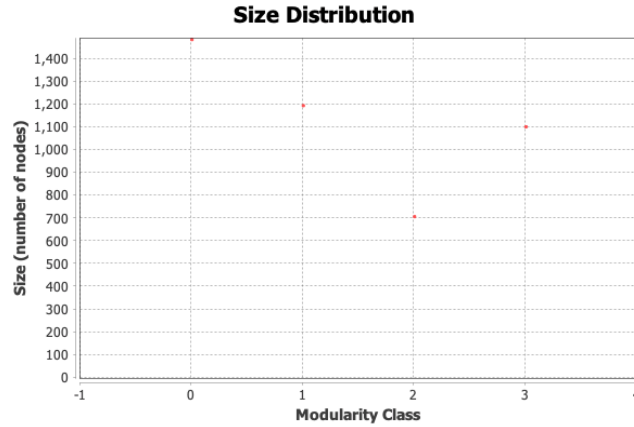


Figure 21. Size distribution of beacon frames correlated by RSSI

This method is better able to create communities of vertices that reflect the ground truth that the data set consists of beacon frames from four APs, however, within those communities, there exists a mix of beacon frames which hinders definitive correlation of a VAP to the source AP. One factor that significantly impacts the ability of the RSSI method is the fact that beacon frames received by Wireshark from the same AP have different RSSIs due to the placement of the physical APs and multipath interference within the building where the collection was conducted. A histogram of the RSSIs of every beacon frame collected from its physical AP can be seen in Figure 22.

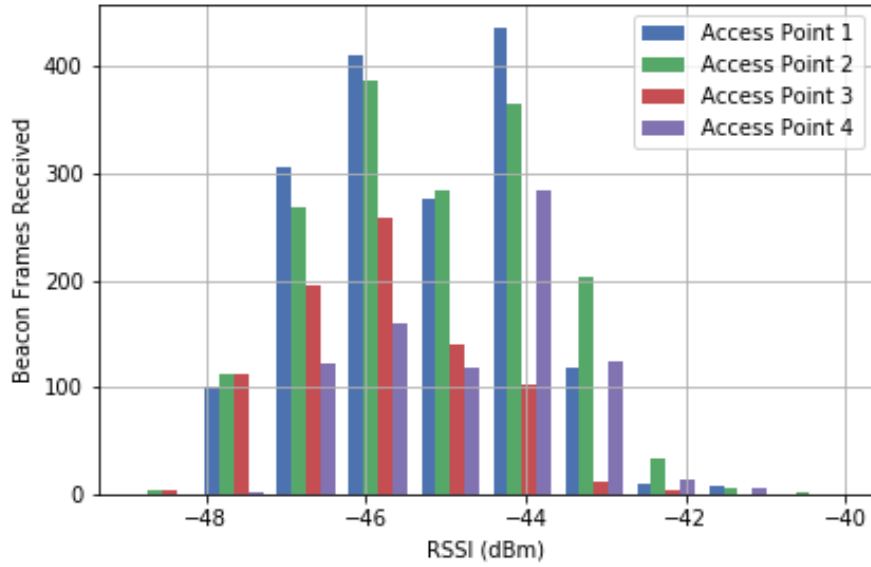


Figure 22. Histogram of RSSI for all APs

Due to the fact that there are multiple overlaps in RSSI for each physical AP, the edge weighting function is unable to adequately differentiate one AP from another based on the RSSI of the beacon frame received. If there were more definitive delineations between the RSSIs of the APs, it is possible that the RSSI method would have a higher F-Score.

E. CONCLUSIONS

Of the three methods examined in this study, the timestamp method was the most promising, with a F-Score of 0.779. The RSSI was the second most promising method, albeit with a much lower F-Score of 0.365. The reception time was demonstrated to have little ability to correlate beacon frames and VAPs to parent APs and had a F-Score of 0.0009. The TP, FP, TN, FN, accuracy, precision, recall, and F-Score for all three methods examined in this study are displayed in Table 1.

Table 1. Summary of experiment results

Method	TP	FP	TN	FN	Accuracy	Precision	Recall	F-Score
Timestamp	4431650	0	18064444	2508906	0.900	1.0	0.639	0.779
RSSI	3551536	8968134	9096310	3389020	0.506	0.284	0.512	0.365
Reception Time	3330	21662	18042782	6937226	0.722	0.133	0.0005	0.0009

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND FUTURE WORK

A. FINDINGS AND CONCLUSIONS

In examining the three proposed methods for correlating VAPs through beacon frames, it turns out that if APs in a WLAN have individual timestamp values, then the VAPs can be correlated almost perfectly. In the absence of a unique timestamp, the RSSI method provides mediocre to poor results and does not adequately correlate VAPs to the point where a malicious actor could definitively correlate a VAP to an AP via this method alone. The reception time method performed so poorly that it does not provide any correlation capability.

B. CONTRIBUTIONS OF THIS STUDY

The contributions of this study include:

- Evaluating the concepts in [14] of using RSSI fingerprinting to correlate VAPs to an AP. Due to the RF environment inherent to an indoor location with multiple APs, it is not possible to reliably fingerprint an AP for the purposes of VAP correlation.
- Advancing the concepts in [1] by examining some of the propositions advanced by the authors on beacon frames. While the authors' seventh proposition that VAPs can be correlated through reception time of probe responses did not carry over to beacon frames, this study was able to confirm that the timestamp field in beacon frames can be used effectively to correlate VAPs.
- Examining the depth of the vulnerability of DoD WLANs to correlation of VAPs.
- Proposing techniques to reduce or eliminate the effectiveness of correlation-style attacks on WLANs.

C. STEPS TO REDUCE THE EFFECTIVENESS OF VAP CORRELATION EFFORTS

As demonstrated by this study, if APs in a WLAN do not have unique timestamp values, then the timestamp method does not work at all. With that in mind, it is advisable for security professionals to cycle power to the WLAN so that all APs come online simultaneously. APs that are located indoors are inherently more resistant to correlation via the RSSI method due to multipath interference and no special measures need to be taken to defeat this method.

D. RECOMMENDATIONS FOR FUTURE WORK

There are multiple avenues for further study of WLAN security:

- This study was conducted in a homogeneous environment with APs sourced from the same manufacturer. Future studies should address heterogeneous environments consisting of different makes and models of APs. It is also worth examining whether different particular makes and models of APs present different attributes which would enable or enhance VAP correlation.
- Another avenue of investigation is to combine the work of this study and [1] to determine whether it is possible to use probe response and beacon frame attributes to achieve better correlation results than those presented in this thesis.
- This study's confirmation of the timestamp was dependent upon making the timestamp fields artificially unique by subtracting a random value between one and 10,000,000. Future work can examine WLANs which have APs with unique timestamps and determine whether this method continues to be effective and whether any steps can be taken to improve upon it.

APPENDIX A. TIMESTAMP PYTHON CODE

This program was written in Jupyter Notebook on a computer with Python 3.7.0 installed. Comments have been added to the code to provide further information to the reader. All of the BSSIDs for the VAPs have been replaced with “XX:XX:XX:XX:XX:XX.”

```
#Program for timestamp edge creation, results analysis, and  $\tau$  graphs
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from random import randint
import datetime
from datetime import timedelta
import decimal
import itertools
from datetime import datetime
from pandas import DataFrame, Series
from itertools import combinations
import matplotlib.pyplot as plt
import seaborn as sns

#Import CSV of beacon frames
nodes_file = 'BeaconFrames.csv'

#Declare name of edges file
edges_file = 'Edges.csv'

#Read entire csv file into dataframe 'df_nodes'
df = pd.read_csv(nodes_file)

#Create a dataframe with ID, reception time, timestamp, and MAC Address
df_time_stamp = df[['ID', 'Reception Time', 'Timestamp', 'MAC Address']].copy()

#Create a “source list” for edges
source_list = df['ID'].values.tolist()

#Create a “target list” for edges
target_list = df['ID'].values.tolist()

#Create edges between all nodes (combine source and target list)
new_source = []
new_dest = []
```

```

for a, b in itertools.product(source_list, target_list):

    #Avoid self-loops (edges to/from the same node)
    if a == b:
        continue
    else:
        new_source.append(a)
        new_dest.append(b)

#Create a dataframe from the combination of source/target Ids
combination_df = pd.DataFrame({'Source': new_source, 'Target': new_dest})

#Generate unique timestamp if timestamps are increasing uniformly
#Row Counter
i = 0

#Holds unique timestamp-Reception Time(microseconds)|
ap_1 = []

#Holds unique timestamp-Reception Time(microseconds)|
ap_2 = []

#Holds unique timestamp-Reception Time(microseconds)|
ap_3 = []

#Holds unique timestamp-Reception Time(microseconds)|
ap_4 = []

#Holds the reception time in microseconds for every beacon frame
ap_total = []

#Holds the unique timestamp for AP one
ap_1_time = []

#Holds the unique timestamp for AP two
ap_2_time = []

#Holds the unique timestamp for AP three
ap_3_time = []

#Holds the unique timestamp for AP four
ap_4_time = []

#Holds total time for every single beacon frame
ap_total_time = []

```

```

#Holds  $\tau$  for every AP one
ap_1_unique_time = []

#Holds  $\tau$  for every AP two
ap_2_unique_time = []

#Holds  $\tau$  for every AP three
ap_3_unique_time = []

#Holds  $\tau$  for every AP four
ap_4_unique_time = []

#Holds reception time for every AP one
ap_1_reception_time = []

#Holds reception time for every AP two
ap_2_reception_time = []

#Holds reception time for every AP three
ap_3_reception_time = []

#Holds reception time for every AP four
ap_4_reception_time = []

#Time format for parsing date time from a string to microseconds
time_format = '%b %d, %Y %H:%M:%S.%f000 %Z'

#Generate Random Number for AP one
ap1_value = randint(1, 10000000)

#Generate Random Number for AP two
ap2_value = randint(1, 10000000)

#Generate Random Number for AP three
ap3_value = randint(1, 10000000)

#Generate Random Number for AP four
ap4_value = randint(1, 10000000)

#While loop to go through every row of the dataframe
while i < len(df_time_stamp):

    #Turn data frame row into a tuple so ID, timestamp, and MAC Address can be accessed
    ap_tuple = df_time_stamp.iloc[i]

```

```

#ID is the 0th element
ap_id = ap_tuple[0]

#Reception time (string format) is the 1st element
reception_time = ap_tuple[1]

#Timestamp is the 2nd element
ap_str_time = ap_tuple[2]

#Timestamp is a string in hexadecimal format and needs to be converted to an integer
#in base 10 format
ap_mac_time = int(ap_str_time, 10)

#Mac Address is the 3rd element
ap_mac_addr = ap_tuple[3]

#Parse reception time using strptime
date_time_object = datetime.strptime(reception_time, time_format)

#Take minutes from parsed time and convert to microseconds
minute = 60000000*date_time_object.minute

#Take seconds from parsed time and convert to microseconds
second = 1000000*date_time_object.second

#Take microseconds from parsed time (no conversion needed)
microsecond = date_time_object.microsecond

#Round microseconds to the closest millisecond because the accuracy of the system
#clock is unknown
rounded_microsecond = round(microsecond,-3)

#Calculate total time for timestamp (microseconds)
total_time = minute + second + rounded_microsecond

#Perform a string match against the BSSID for the first VAP for AP one
if ap_mac_addr == "XX:XX:XX:XX:XX:XX":

    #Subtract random value from timestamp to get a unique value (ap1_time)
    ap_1_time = ap_mac_time - ap1_value

    #Subtract reception time in microseconds to account for time passage
    unique_time = abs(ap_1_time - total_time)

```

```

#Append "unique time" to list ap_1
ap_1.append(unique_time)

#Append "unique time" to list ap_total, this will go into a DF for edge weighting
#function
ap_total.append(unique_time)

#Append AP one time (timestamp-random value) to list ap_1_time
ap_1_unique_time.append(unique_time)

#Append total time (minute + seconds + microseconds) in microseconds to
#ap_1_reception_time
ap_1_reception_time.append(total_time)

#Perform a string match against the second BSSID for the VAP for AP one
elif ap_mac_addr == "XX:XX:XX:XX:XX:XX":
    ap_1_time = ap_mac_time - ap1_value
    unique_time = abs(ap_1_time - total_time)
    ap_1.append(unique_time)
    ap_total.append(unique_time)
    ap_1_unique_time.append(unique_time)
    ap_1_reception_time.append(total_time)

#Perform a string match against the first BSSID for the VAP for AP two
elif ap_mac_addr == "XX:XX:XX:XX:XX:XX":
    ap_2_time = ap_mac_time - ap2_value
    unique_time = abs(ap_2_time - total_time)
    ap_2.append(unique_time)
    ap_total.append(unique_time)
    ap_2_unique_time.append(unique_time)
    ap_2_reception_time.append(total_time)

#Perform a string match against the second BSSID for the VAP for AP two
elif ap_mac_addr == "XX:XX:XX:XX:XX:XX":
    ap_2_time = ap_mac_time - ap2_value
    unique_time = abs(ap_2_time - total_time)
    ap_2.append(unique_time)
    ap_total.append(unique_time)
    ap_2_unique_time.append(unique_time)
    ap_2_reception_time.append(total_time)

#Perform a string match against the first BSSID for the VAP for AP three
elif ap_mac_addr == "XX:XX:XX:XX:XX:XX":
    ap_3_time = ap_mac_time - ap3_value
    unique_time = abs(ap_3_time - total_time)

```

```

    ap_3.append(unique_time)
    ap_total.append(unique_time)
    ap_3_unique_time.append(unique_time)
    ap_3_reception_time.append(total_time)

#Perform a string match against the second BSSID for the VAP for AP three
elif ap_mac_addr == "XX:XX:XX:XX:XX:XX":
    ap_3_time = ap_mac_time - ap3_value
    unique_time = abs(ap_3_time - total_time)
    ap_3.append(unique_time)
    ap_total.append(unique_time)
    ap_3_unique_time.append(unique_time)
    ap_3_reception_time.append(total_time)

#Perform a string match against the first BSSID for the VAP for AP 4
elif ap_mac_addr == "XX:XX:XX:XX:XX:XX":
    ap_4_time = ap_mac_time - ap4_value
    unique_time = abs(ap_4_time - total_time)
    ap_4.append(unique_time)
    ap_total.append(unique_time)
    ap_4_unique_time.append(unique_time)
    ap_4_reception_time.append(total_time)

#Perform a string match against the second BSSID for the VAP for AP 4
elif ap_mac_addr == "XX:XX:XX:XX:XX:XX":
    ap_4_time = ap_mac_time - ap4_value
    unique_time = abs(ap_4_time - total_time)
    ap_4.append(unique_time)
    ap_total.append(unique_time)
    ap_4_unique_time.append(unique_time)
    ap_4_reception_time.append(total_time)

#Catch any cases that do not match any of the above
else:
    print("Not Match")

# 'i' increments through the rows
i += 1

#Create data frame consisting solely of timestamps
mac_time_df= pd.DataFrame({'Timestamp': ap_total})

#Edge weight function to determine edge weight for each potential connection generated
# 'row_count' counts the row number and is incremented
row_count = 0

```

```

# 'output' is a list to store edge weight function output
output = []

# While row_count is less than the length of combination_df,
# execute the edge weight function
while row_count < len(combination_df):

    # Retrieve source and target tuple from 'combination_df'
    src_tgt_tuple = combination_df.iloc[row_count]

    # Source node is the first element
    src = src_tgt_tuple[0]

    # Target node is the second element
    tgt = src_tgt_tuple[1]

    # Pull the source node timestamp from the mac_time_df dataframe
    src_mac_time = mac_time_df.iloc[src]

    # Pull the target node timestamp from the mac_time_df dataframe
    tgt_mac_time = mac_time_df.iloc[tgt]

    # Compute the absolute value difference between the source and target timestamp
    time_delta = abs(src_mac_time - tgt_mac_time)

    # If the time delta is less than/equal to 500, then it is likely the same time
    # and an edge weight of 1 is assigned
    if time_delta.item() <= 500:
        edge_weight = 1
        output.append(edge_weight)

    # If the time delta is greater than 500, then it is not likely the same time
    # and an edge weight of 0 is assigned (i.e., no edge exists between the source
    # and target nodes)
    else:
        edge_weight = 0
        output.append(edge_weight)

    # Increment row count
    row_count += 1

# Convert output list into a data frame
df_weights = pd.DataFrame({'Weight': output})

```

```

#Create a dataframe of all source nodes and destination nodes and the edge weight of those
#connections
final_df = pd.DataFrame({'Source': new_source, 'Target': new_dest, 'Weight': output})

#Place "ID" over first column to prevent column title shifting to the left when it is imported
#into Gephi
final_df.index.name = 'ID'

#Write to a CSV
final_df.to_csv(edges_file, sep=',')

#Analysis of TP, FP, TN, FN, accuracy, precision, recall, and F-Score
#Declare variables for BSSIDs for all VAPs for APs 1–4
ap_one_one = "XX:XX:XX:XX:XX:XX"
ap_one_two = "XX:XX:XX:XX:XX:XX"
ap_two_one = "XX:XX:XX:XX:XX:XX"
ap_two_two = "XX:XX:XX:XX:XX:XX"
ap_three_one = "XX:XX:XX:XX:XX:XX"
ap_three_two = "XX:XX:XX:XX:XX:XX"
ap_four_one = "XX:XX:XX:XX:XX:XX"
ap_four_two = "XX:XX:XX:XX:XX:XX"

#Maintain count of row in Dataframe
row_count = 0

#Maintains count of TP
true_positive = 0

#Maintains count of FP
false_positive = 0

#Maintains count of TN
true_negative = 0

#Maintains count of FN
false_negative = 0

#While loop to go through every row of the dataframe
while row_count < len(final_df):

    #Unpacks row from final_df into a 3 tuple (src, dest, and weight)
    edge_tuple = final_df.iloc[row_count]
    src = int(edge_tuple[0])
    tgt = int(edge_tuple[1])
    edge_weight = edge_tuple[2]

```



```
#Unpacks row from df into a tuple (mac addr of the edge source and mac addr of the
#edge target)
```

```
beacon_frame_tuple_src = df.iloc[src]
mac_addr_src = beacon_frame_tuple_src[6]
beacon_frame_tuple_tgt = df.iloc[tgt]
mac_addr_tgt = beacon_frame_tuple_tgt[6]
```

```
#Connection exists: Evaluate whether correct (True Positive) or incorrect (False
#Positive)
```

```
#If edge weight is anything but zero, it has a weight
if edge_weight != 0:
```

```
    #If source and target MAC Address match, it is a true positive (Connection should
    #exist and does)
```

```
    #AP one
```

```
    if (mac_addr_src == ap_one_one and mac_addr_tgt == ap_one_one):
        true_positive += 1
    elif (mac_addr_src == ap_one_one and mac_addr_tgt == ap_one_two):
        true_positive += 1
    elif (mac_addr_src == ap_one_two and mac_addr_tgt == ap_one_one):
        true_positive += 1
    elif (mac_addr_src == ap_one_two and mac_addr_tgt == ap_one_two):
        true_positive += 1
```

```
    #AP two
```

```
    elif (mac_addr_src == ap_two_one and mac_addr_tgt == ap_two_one):
        true_positive += 1
    elif (mac_addr_src == ap_two_one and mac_addr_tgt == ap_two_two):
        true_positive += 1
    elif (mac_addr_src == ap_two_two and mac_addr_tgt == ap_two_one):
        true_positive += 1
    elif (mac_addr_src == ap_two_two and mac_addr_tgt == ap_two_two):
        true_positive += 1
```

```
    #AP three
```

```
    elif (mac_addr_src == ap_three_one and mac_addr_tgt == ap_three_one):
        true_positive += 1
    elif (mac_addr_src == ap_three_one and mac_addr_tgt == ap_three_two):
        true_positive += 1
    elif (mac_addr_src == ap_three_two and mac_addr_tgt == ap_three_one):
        true_positive += 1
    elif (mac_addr_src == ap_three_two and mac_addr_tgt == ap_three_two):
        true_positive += 1
```

```

#AP four
elif (mac_addr_src == ap_four_one and mac_addr_tgt == ap_four_one):
    true_positive += 1
elif (mac_addr_src == ap_four_one and mac_addr_tgt == ap_four_two):
    true_positive += 1
elif (mac_addr_src == ap_four_two and mac_addr_tgt == ap_four_one):
    true_positive += 1
elif (mac_addr_src == ap_four_two and mac_addr_tgt == ap_four_two):
    true_positive += 1

#If source and target MAC Address do not match, it is a false positive (Connection
#exists and shouldn't)
else:
    false_positive += 1

#Connection does not exist: Evaluate whether correct (True Negative) or incorrect (False
#Negative)
elif edge_weight == 0:

    #If source and target MAC Address match, it is a false negative (Connection should
    #exist but doesn't)
    #AP one
    if (mac_addr_src == ap_one_one and mac_addr_tgt == ap_one_one):
        false_negative += 1
    elif (mac_addr_src == ap_one_one and mac_addr_tgt == ap_one_two):
        false_negative += 1
    elif (mac_addr_src == ap_one_two and mac_addr_tgt == ap_one_one):
        false_negative += 1
    elif (mac_addr_src == ap_one_two and mac_addr_tgt == ap_one_two):
        false_negative += 1

    #AP two
    elif (mac_addr_src == ap_two_one and mac_addr_tgt == ap_two_one):
        false_negative += 1
    elif (mac_addr_src == ap_two_one and mac_addr_tgt == ap_two_two):
        false_negative += 1
    elif (mac_addr_src == ap_two_two and mac_addr_tgt == ap_two_one):
        false_negative += 1
    elif (mac_addr_src == ap_two_two and mac_addr_tgt == ap_two_two):
        false_negative += 1

    #AP three
    elif (mac_addr_src == ap_three_one and mac_addr_tgt == ap_three_one):
        false_negative += 1

```

```

elif (mac_addr_src == ap_three_one and mac_addr_tgt == ap_three_two):
    false_negative += 1
elif (mac_addr_src == ap_three_two and mac_addr_tgt == ap_three_one):
    false_negative += 1
elif (mac_addr_src == ap_three_two and mac_addr_tgt == ap_three_two):
    false_negative += 1

#AP four
elif (mac_addr_src == ap_four_one and mac_addr_tgt == ap_four_one):
    false_negative += 1
elif (mac_addr_src == ap_four_one and mac_addr_tgt == ap_four_two):
    false_negative += 1
elif (mac_addr_src == ap_four_two and mac_addr_tgt == ap_four_one):
    false_negative += 1
elif (mac_addr_src == ap_four_two and mac_addr_tgt == ap_four_two):
    false_negative += 1

#If source and target MAC Address do not match, it is a true negative (Connection
#shouldn't exist and doesn't)
else:
    true_negative += 1

#Increment row count
row_count += 1

#Calculate the accuracy of the method
artificial_timestamp_accuracy=(true_positive+true_negative)/(true_positive+
false_positive + false_negative + true_negative)
print(artificial_timestamp_accuracy)

#Calculate recall of the method
artificial_timestamp_recall = true_positive/(true_positive + false_negative)
print(artificial_timestamp_recall)

#Calculate the precision of the method
artificial_timestamp_precision = true_positive/(true_positive+false_positive)
print(artificial_timestamp_precision)

#Calculate the F-Score of the method
artificial_timestamp_f_score=true_positive/(true_positive+0.5*(false_positive+
false_negative))
print(artificial_timestamp_f_score)

#Plot "jitter" in AP times
#Step 1: Create a dataframe from data generated from Step 7 above

```

```
ap_1_unique_df=pd.DataFrame({'ReceptionTime (microseconds)': ap_1_reception_time,  
' $\tau$  (microseconds)': ap_1_unique_time})
```

#Step 2: Select every 9th row to plot to reduce number of data points and

#make the graph easier to interpret

```
ap_1_jitter = ap_1_unique_df[ap_1_unique_df.index % 9 ==0]
```

#Step 3: Create a line graph from the dataframe created above and save it as a .png file

```
sns_fig = sns.lineplot(x= "Reception Time (microseconds)," y= " $\tau$  (microseconds)," data=  
ap_1_jitter)
```

#Save graph

```
fig = sns_fig.get_figure()
```

```
fig.savefig('AP one:  $\tau$  v.s. Time.png')
```

#Step 4: Repeat steps 1–3 for AP two

```
ap_2_unique_df=pd.DataFrame({'ReceptionTime (microseconds)': ap_2_reception_time,  
' $\tau$  (microseconds)': ap_2_unique_time})
```

```
ap_2_jitter = ap_2_unique_df[ap_2_unique_df.index % 9 ==0]
```

```
sns_fig = sns.lineplot(x= "Reception Time (microseconds)," y= " $\tau$  (microseconds)," data=  
ap_2_jitter)
```

#Save graph

```
fig = sns_fig.get_figure()
```

```
fig.savefig('AP two:  $\tau$  v.s. Time.png')
```

#Step 5: Repeat steps 1–3 for AP three

```
ap_3_unique_df=pd.DataFrame({'ReceptionTime (microseconds)': ap_3_reception_time,  
' $\tau$  (microseconds)': ap_3_unique_time})
```

```
ap_3_jitter = ap_3_unique_df[ap_3_unique_df.index % 9 ==0]
```

```
sns_fig = sns.lineplot(x= "Reception Time (microseconds)," y= " $\tau$  (microseconds)," data=  
ap_3_jitter)
```

#Save figure

```
fig = sns_fig.get_figure()
```

```
fig.savefig('AP three:  $\tau$  v.s. Time.png')
```

#Step 6: Repeat steps 1–3 for AP 4

```
ap_4_unique_df=pd.DataFrame({'ReceptionTime (microseconds)': ap_4_reception_time,  
' $\tau$  (microseconds)': ap_4_unique_time})
```

```
ap_4_jitter = ap_4_unique_df[ap_4_unique_df.index % 9 ==0]
```

```
sns_fig = sns.lineplot(x= "Reception Time (microseconds)," y= " $\tau$  (microseconds)," data=  
ap_4_jitter)
```

```
#Save figure  
fig.savefig('AP 4:  $\tau$  v.s. Time.png')
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. RECEPTION TIME PYTHON CODE

This program was written in Jupyter Notebook on a computer with Python 3.7.0 installed. Comments have been added to the code to provide further information to the reader. All of the BSSIDs for the VAPs have been replaced with “XX:XX:XX:XX:XX:XX.”

```
#Program for reception time edge creation and results analysis
import pandas as pd
import numpy as np
from datetime import datetime
from pandas import DataFrame, Series
from itertools import combinations
from functools import reduce
import itertools

#Import CSV of beacon frames
nodes_file = 'BeaconFrames.csv'

#Declare name of edges file
edges_file = 'Edges.csv'

#Open Node file, create node dataframe named 'df_node' of ID and reception time
df = pd.read_csv(nodes_file)

#Create a dataframe consisting of Id and Time
df_nodes = df[['ID', 'Reception Time']]

#Create source list for edges
source_list = df['ID'].values.tolist()

#Create target list for edges
target_list = df['ID'].values.tolist()

#Convert values in the dataframe to a list
weight_list = df['Reception Time'].values.tolist()

#Create a dataframe of time values
df_Time_value = df['Reception Time']

#Create edges between all nodes
new_source = []
new_dest = []
for a, b in itertools.product(source_list, target_list):
    if a == b:
```

```

        #Avoid self-loops (edges to/from the same node)
        continue
    else:
        new_source.append(a)
        new_dest.append(b)

#Create a dataframe of source and target nodes
new_df = pd.DataFrame({'Source': new_source, 'Target': new_dest})

#Edge weight function
# 'row_count' counts the row number and is incremented
row_count = 0

# 'output' is a list to store edge weight function output
output = []

#Time format for parsing date time
time_format = '%b %d, %Y %H:%M:%S.%f000 %Z'

#While loop to go through every row of the dataframe
while row_count < len(new_df):

    #Turn dataframe row into a tuple so that the ID for the edge nodes can be accessed
    src_tgt_tuple = new_df.iloc[row_count]

    #Source node is the 0th element
    src = src_tgt_tuple[0]

    #Target node is the 1st element
    tgt = src_tgt_tuple[1]

    #Pull the source node time from the df_Time_value data frame
    src_time = df_Time_value[src]

    #Pull the target node time from the df_Time_value data frame
    tgt_time = df_Time_value[tgt]

    #Convert the source node's reception time into a date time object
    date_time_object_src = datetime.strptime(src_time, time_format)

    #Take minutes from parsed time and convert to microseconds
    src_minute = 60000000*date_time_object_src.minute

    #Take seconds from parsed time and convert to microseconds
    src_second = 1000000*date_time_object_src.second

```



```

#Take microseconds from parsed time
src_microsecond = date_time_object_src.microsecond

#Round microseconds to the closest millisecond because the accuracy of the system
#clock is unknown
src_reception_time = src_minute + src_second + src_microsecond

#Convert the target node's reception time into a date time object
date_time_object_tgt = datetime.strptime(tgt_time, time_format)

#Take minutes from parsed time and convert to microseconds
tgt_minute = 60000000*date_time_object_tgt.minute

#Take seconds from parsed time and convert to microseconds
tgt_second = 1000000*date_time_object_tgt.second

#Take microseconds from parsed time
tgt_microsecond = date_time_object_tgt.microsecond

#Calculate target node's reception time using seconds and milliseconds
tgt_reception_time = tgt_minute + tgt_second + tgt_microsecond

#Calculate absolute value of source node reception minus target node reception time
reception_time_func = abs(src_reception_time-tgt_reception_time)

#Determine if  $\Delta RT$  is less than/equal to 621 microseconds
if reception_time_func <= 621:
    edge_weight = 1.0

#Determine if  $\Delta RT$  is less than/equal to 700 microseconds
elif reception_time_func <= 700:
    edge_weight = 0.75

#Determine if  $\Delta RT$  is less than/equal to 1024 microseconds
elif reception_time_func <= 1024:
    edge_weight = 0.50

#Determine if  $\Delta RT$  is greater than 1024 microseconds
elif reception_time_func > 1024:
    edge_weight = 0

#Place weight into output list
output.append(edge_weight)
#Increment row
row_count += 1

```

```

#Add df_weights to 'newer_df' (master dataframe)
final_df = pd.DataFrame({'Source': new_source, 'Target': new_dest, 'Weight': output})

#Place "ID" over first column to prevent column title shifting to the left in Gephi
final_df.index.name = 'ID'

#Write to a CSV
final_df.to_csv(edges_file, sep=',')

#Analysis of TP, FP, TN, FN, accuracy, precision, recall, and F-Score
#Declare variables for BSSIDs for all VAPs for APs 1-4
ap_one_one = "XX:XX:XX:XX:XX:XX"
ap_one_two = "XX:XX:XX:XX:XX:XX"
ap_two_one = "XX:XX:XX:XX:XX:XX"
ap_two_two = "XX:XX:XX:XX:XX:XX"
ap_three_one = "XX:XX:XX:XX:XX:XX"
ap_three_two = "XX:XX:XX:XX:XX:XX"
ap_four_one = "XX:XX:XX:XX:XX:XX"
ap_four_two = "XX:XX:XX:XX:XX:XX"

#Maintain count of row in Dataframe
row_count = 0

#Maintains count of TP
true_positive = 0

#Maintains count of FP
false_positive = 0

#Maintains count of TN
true_negative = 0

#Maintains count of FN
false_negative = 0

#While loop to go through every row of the dataframe
while row_count < len(final_df):

    #Unpacks row from final_df into a 3 tuple (src, dest, and weight)
    edge_tuple = final_df.iloc[row_count]
    src = int(edge_tuple[0])
    tgt = int(edge_tuple[1])
    edge_weight = edge_tuple[2]

```

```
#Unpacks row from df into a tuple (mac addr of the edge source and mac addr of the
#edge target)
```

```
beacon_frame_tuple_src = df.iloc[src]
mac_addr_src = beacon_frame_tuple_src[6]
beacon_frame_tuple_tgt = df.iloc[tgt]
mac_addr_tgt = beacon_frame_tuple_tgt[6]
```

```
#Connection exists: Evaluate whether correct (True Positive) or incorrect (False
#Positive)
```

```
if edge_weight != 0:
```

```
    #If source and target MAC Address match, it is a true positive (Connection should
    #exist and does)
```

```
    #AP one
```

```
    if (mac_addr_src == ap_one_one and mac_addr_tgt == ap_one_one):
        true_positive += 1
    elif (mac_addr_src == ap_one_one and mac_addr_tgt == ap_one_two):
        true_positive += 1
    elif (mac_addr_src == ap_one_two and mac_addr_tgt == ap_one_one):
        true_positive += 1
    elif (mac_addr_src == ap_one_two and mac_addr_tgt == ap_one_two):
        true_positive += 1
```

```
    #AP two
```

```
    elif (mac_addr_src == ap_two_one and mac_addr_tgt == ap_two_one):
        true_positive += 1
    elif (mac_addr_src == ap_two_one and mac_addr_tgt == ap_two_two):
        true_positive += 1
    elif (mac_addr_src == ap_two_two and mac_addr_tgt == ap_two_one):
        true_positive += 1
    elif (mac_addr_src == ap_two_two and mac_addr_tgt == ap_two_two):
        true_positive += 1
```

```
    #AP three
```

```
    elif (mac_addr_src == ap_three_one and mac_addr_tgt == ap_three_one):
        true_positive += 1
    elif (mac_addr_src == ap_three_one and mac_addr_tgt == ap_three_two):
        true_positive += 1
    elif (mac_addr_src == ap_three_two and mac_addr_tgt == ap_three_one):
        true_positive += 1
    elif (mac_addr_src == ap_three_two and mac_addr_tgt == ap_three_two):
        true_positive += 1
```

```
    #AP four
```

```
    elif (mac_addr_src == ap_four_one and mac_addr_tgt == ap_four_one):
```

```

    true_positive += 1
elif (mac_addr_src == ap_four_one and mac_addr_tgt == ap_four_two):
    true_positive += 1
elif (mac_addr_src == ap_four_two and mac_addr_tgt == ap_four_one):
    true_positive += 1
elif (mac_addr_src == ap_four_two and mac_addr_tgt == ap_four_two):
    true_positive += 1

#If source and target MAC Address do not match, it is a false positive (Connection
#exists and shouldn't)
else:
    false_positive += 1

#Connection does not exist: Evaluate whether correct (True Negative) or incorrect (False
#Negative)
elif edge_weight == 0:

    #If source and target MAC Address match, it is a false negative (Connection should
    #exist but doesn't)
    #AP one
    if (mac_addr_src == ap_one_one and mac_addr_tgt == ap_one_one):
        false_negative += 1
    elif (mac_addr_src == ap_one_one and mac_addr_tgt == ap_one_two):
        false_negative += 1
    elif (mac_addr_src == ap_one_two and mac_addr_tgt == ap_one_one):
        false_negative += 1
    elif (mac_addr_src == ap_one_two and mac_addr_tgt == ap_one_two):
        false_negative += 1

    #AP two
    elif (mac_addr_src == ap_two_one and mac_addr_tgt == ap_two_one):
        false_negative += 1
    elif (mac_addr_src == ap_two_one and mac_addr_tgt == ap_two_two):
        false_negative += 1
    elif (mac_addr_src == ap_two_two and mac_addr_tgt == ap_two_one):
        false_negative += 1
    elif (mac_addr_src == ap_two_two and mac_addr_tgt == ap_two_two):
        false_negative += 1

    #AP three
    elif (mac_addr_src == ap_three_one and mac_addr_tgt == ap_three_one):
        false_negative += 1
    elif (mac_addr_src == ap_three_one and mac_addr_tgt == ap_three_two):
        false_negative += 1
    elif (mac_addr_src == ap_three_two and mac_addr_tgt == ap_three_one):

```

```

        false_negative += 1
    elif (mac_addr_src == ap_three_two and mac_addr_tgt == ap_three_two):
        false_negative += 1

#AP 4
    elif (mac_addr_src == ap_four_one and mac_addr_tgt == ap_four_one):
        false_negative += 1
    elif (mac_addr_src == ap_four_one and mac_addr_tgt == ap_four_two):
        false_negative += 1
    elif (mac_addr_src == ap_four_two and mac_addr_tgt == ap_four_one):
        false_negative += 1
    elif (mac_addr_src == ap_four_two and mac_addr_tgt == ap_four_two):
        false_negative += 1

    #If source and target MAC Address do not match, it is a true negative (Connection
    #shouldn't exist and doesn't)
    else:
        true_negative += 1

#Increment row
    row_count += 1

#Calculate the accuracy of the method
    reception_time_accuracy = (true_positive + true_negative)/(true_positive + false_positive
    + false_negative + true_negative)
    print(reception_time_accuracy)

#Calculate recall of the method
    reception_time_recall = true_positive/(true_positive + false_negative)
    print(reception_time_recall)

#Calculate precision of the method
    reception_time_precision = true_positive/(true_positive+false_positive)
    print(reception_time_precision)

#Calculate F-Score of the method
    reception_time_f_score=true_positive/(true_positive+0.5*(false_positive+
    false_negative))
    print(reception_time_f_score)

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. RSSI PYTHON CODE

This program was written in Jupyter Notebook on a computer with Python 3.7.0 installed. Comments have been added to the code to provide further information to the reader. All of the BSSIDs for the VAPs have been replaced with “XX:XX:XX:XX:XX:XX.”

```
#Program for RSSI edge creation and results analysis
import pandas as pd
import numpy as np
from datetime import datetime
from pandas import DataFrame, Series
from itertools import combinations
from functools import reduce
import itertools

#Import CSV of beacon frames
nodes_file = 'BeaconFrames.csv'

#Declare name of edges file
edges_file = 'Edges.csv'

# Open Node file and create a dataframe called “df”
df = pd.read_csv(nodes_file)

#Create a dataframe called “df_nodes” consisting of the beacon frame ID number and RSSI
df_nodes = df[['Id', 'dBm']]

#Create a list of edge sources from beacon frame IDs
source_list = df['ID'].values.tolist()

#Create a list of edge destinations from beacon frame Ids
target_list = df['ID'].values.tolist()

#Create a list called “weight_list” from dataframe consisting of beacon frame RSSIs
weight_list = df['dBm'].values.tolist()

#Create a dataframe called “df_dBm_value” from a dataframe consisting of beacon frame
#RSSIs
df_dBm_value = df['dBm']

#Create edges between all nodes
new_source = []
```

```

new_dest = []
for a, b in itertools.product(source_list, target_list):

    #Avoid self-loops (edges to/from the same node)
    if a == b:
        continue
    else:
        new_source.append(a)
        new_dest.append(b)

#Create a dataframe of source and target nodes
new_df = pd.DataFrame({'Source': new_source, 'Target': new_dest})

#Edge weighting function
# 'row_count' counts the row number and is incremented
row_count = 0

# 'output' is a list to store edge weighting function output
output = []

#While loop to go through every row of the dataframe
while row_count < len(new_df):

    #Retrieve source and dest tuple from 'new_df'
    src_tgt_tuple = new_df.iloc[row_count]

    #Source node is the 0th element
    src = src_tgt_tuple[0]

    #Target node is the 1st element
    tgt = src_tgt_tuple[1]

    #Pull the source node dBm from the df_dBm_value data frame
    src_dBm = df_dBm_value[src]

    #Pull the target node dBm from the df_dBm_value data frame
    tgt_dBm = df_dBm_value[tgt]

    #Get denominator for edge weighting function; |Src-Tgt|
    func_denom = abs(src_dBm-tgt_dBm)

    #If the absolute value difference for "func_denom" is not equal to zero, divide 0.15 by
    #the value in that variable
    if func_denom != 0:

```



```

#If RSSI difference is greater than 1 dBm, do not assign an edge weight. This serves
#as a cut-off.
if func_denom > 1:
    edge_weight = 0

#If RSSI difference is not greater than 1, then divide 0.15 by that value and store result
#in the variable "func_denom."
else:
    edge_weight = (0.15/func_denom)

#If the RSSIs are identical equal then weight the edge 1.5.
else:
    edge_weight = 0.80

#Append edge weight to the list "output"
output.append(edge_weight)

#Increment the row
row_count += 1

#Add df_weights to 'newer_df' (master dataframe)
final_df = pd.DataFrame({'Source': new_source, 'Target': new_dest, 'Weight': output})

#Place "ID" over first column to prevent column title shifting to the left in Gephi
final_df.index.name = 'ID'

#Write to a CSV
final_df.to_csv(edges_file, sep=',')

#Analysis of TP, FP, TN, FN, accuracy, precision, recall, and F-Score
#Declare variables for BSSIDs for all VAPs for APs 1-4
ap_one_one = "XX:XX:XX:XX:XX:XX"
ap_one_two = "XX:XX:XX:XX:XX:XX"
ap_two_one = "XX:XX:XX:XX:XX:XX"
ap_two_two = "XX:XX:XX:XX:XX:XX"
ap_three_one = "XX:XX:XX:XX:XX:XX"
ap_three_two = "XX:XX:XX:XX:XX:XX"
ap_four_one = "XX:XX:XX:XX:XX:XX"
ap_four_two = "XX:XX:XX:XX:XX:XX"

#Maintain count of row in dataframe
row_count = 0

#Maintains count of TP
true_positive = 0

```

```

#Maintains count of FP
false_positive = 0

#Maintains count of TN
true_negative = 0

#Maintains count of FN
false_negative = 0

#While loop to go through every row of the dataframe
while row_count < len(final_df):

    #Unpacks row from final_df into a 3 tuple (src, dest, and weight)
    edge_tuple = final_df.iloc[row_count]
    src = int(edge_tuple[0])
    tgt = int(edge_tuple[1])
    edge_weight = edge_tuple[2]

    #Unpacks row from df into a tuple (mac address of the edge source and mac address of
    #the edge target)
    beacon_frame_tuple_src = df.iloc[src]
    mac_addr_src = beacon_frame_tuple_src[6]
    beacon_frame_tuple_tgt = df.iloc[tgt]
    mac_addr_tgt = beacon_frame_tuple_tgt[6]

    #Connection exists: Evaluate whether correct (True Positive) or incorrect (False
    #Positive)
    if edge_weight != 0:

        #If source and target MAC address match, it is a true positive (Connection should
        #exist and does)
        #AP one
        if (mac_addr_src == ap_one_one and mac_addr_tgt == ap_one_one):
            true_positive += 1
        elif (mac_addr_src == ap_one_one and mac_addr_tgt == ap_one_two):
            true_positive += 1
        elif (mac_addr_src == ap_one_two and mac_addr_tgt == ap_one_one):
            true_positive += 1
        elif (mac_addr_src == ap_one_two and mac_addr_tgt == ap_one_two):
            true_positive += 1

        #AP two
        elif (mac_addr_src == ap_two_one and mac_addr_tgt == ap_two_one):
            true_positive += 1
        elif (mac_addr_src == ap_two_one and mac_addr_tgt == ap_two_two):

```

```

    true_positive += 1
elif (mac_addr_src == ap_two_two and mac_addr_tgt == ap_two_one):
    true_positive += 1
elif (mac_addr_src == ap_two_two and mac_addr_tgt == ap_two_two):
    true_positive += 1

#AP three
elif (mac_addr_src == ap_three_one and mac_addr_tgt == ap_three_one):
    true_positive += 1
elif (mac_addr_src == ap_three_one and mac_addr_tgt == ap_three_two):
    true_positive += 1
elif (mac_addr_src == ap_three_two and mac_addr_tgt == ap_three_one):
    true_positive += 1
elif (mac_addr_src == ap_three_two and mac_addr_tgt == ap_three_two):
    true_positive += 1

#AP 4
elif (mac_addr_src == ap_four_one and mac_addr_tgt == ap_four_one):
    true_positive += 1
elif (mac_addr_src == ap_four_one and mac_addr_tgt == ap_four_two):
    true_positive += 1
elif (mac_addr_src == ap_four_two and mac_addr_tgt == ap_four_one):
    true_positive += 1
elif (mac_addr_src == ap_four_two and mac_addr_tgt == ap_four_two):
    true_positive += 1

#If source and target MAC address do not match, it is a false positive (Connection
#exists and shouldn't)
else:
    false_positive += 1

#Connection does not exist: Evaluate whether correct (True Negative) or incorrect (False
#Negative)
elif edge_weight == 0:

    #If source and target MAC address match, it is a false negative (Connection should
    #exist but doesn't)
    #AP one
    if (mac_addr_src == ap_one_one and mac_addr_tgt == ap_one_one):
        false_negative += 1
    elif (mac_addr_src == ap_one_one and mac_addr_tgt == ap_one_two):
        false_negative += 1
    elif (mac_addr_src == ap_one_two and mac_addr_tgt == ap_one_one):
        false_negative += 1
    elif (mac_addr_src == ap_one_two and mac_addr_tgt == ap_one_two):

```

```

    false_negative += 1

#AP two
elif (mac_addr_src == ap_two_one and mac_addr_tgt == ap_two_one):
    false_negative += 1
elif (mac_addr_src == ap_two_one and mac_addr_tgt == ap_two_two):
    false_negative += 1
elif (mac_addr_src == ap_two_two and mac_addr_tgt == ap_two_one):
    false_negative += 1
elif (mac_addr_src == ap_two_two and mac_addr_tgt == ap_two_two):
    false_negative += 1

#AP three
elif (mac_addr_src == ap_three_one and mac_addr_tgt == ap_three_one):
    false_negative += 1
elif (mac_addr_src == ap_three_one and mac_addr_tgt == ap_three_two):
    false_negative += 1
elif (mac_addr_src == ap_three_two and mac_addr_tgt == ap_three_one):
    false_negative += 1
elif (mac_addr_src == ap_three_two and mac_addr_tgt == ap_three_two):
    false_negative += 1

#AP four
elif (mac_addr_src == ap_four_one and mac_addr_tgt == ap_four_one):
    false_negative += 1
elif (mac_addr_src == ap_four_one and mac_addr_tgt == ap_four_two):
    false_negative += 1
elif (mac_addr_src == ap_four_two and mac_addr_tgt == ap_four_one):
    false_negative += 1
elif (mac_addr_src == ap_four_two and mac_addr_tgt == ap_four_two):
    false_negative += 1

#If source and target MAC address do not match, it is a true negative (Connection
#shouldn't exist and doesn't)
else:
    true_negative += 1

#Increment row
row_count += 1

#Calculate the accuracy of the method
reception_time_accuracy = (true_positive + true_negative)/(true_positive + false_positive
+ false_negative + true_negative)
print(signal_strength_accuracy)

```

```
#Calculate recall of the method
reception_time_recall = true_positive/(true_positive + false_negative)
print(signal_strength_recall)

#Calculate precision of the method
reception_time_precision = true_positive/(true_positive+false_positive)
print(signal_strength_precision)

#Calculate F-Score of the method
reception_time_f_score=true_positive/(true_positive+0.5*(false_positive+
false_negative))
print(reception_time_f_score)
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. RSSI GRAPHING AND ANALYSIS PYTHON CODE

This program was written in Jupyter Notebook on a computer with Python 3.7.0 installed. Comments have been added to the code to provide further information to the reader. All of the BSSIDs for the VAPs have been replaced with “XX:XX:XX:XX:XX:XX.”

```
#Program to perform analysis on the RSSIs for all of the collected beacon frames
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import scipy.stats
from scipy.stats import norm
```

```
#Declare nodes file
nodes_file = 'BeaconFrame.csv'
```

```
#Read entire csv file into dataframe 'df_nodes'
df = pd.read_csv(nodes_file)
```

```
#Plot histogram of signal strength across all APs
plt.hist(df['dBm'])
```

```
#Label y axis
plt.ylabel("Number of beacon frames")
```

```
#Label x axis
plt.xlabel("dBm")
```

```
#Label graph with title
plt.title("Over All Distribution of Signal Strength Across all APs")
```

```
#Display graph to screen
plt.show()
```

```
#Sort RSSI by AP
#List for AP one information
ap_1_bin = []
```

```
#List for AP one RSSI
ap_1_dBm = []
```

```
#List for AP two information
ap_2_bin = []
```

```
#List for AP two RSSI
```

```
ap_2_dBm = []
```

```
#List for AP three information
```

```
ap_3_bin = []
```

```
#List for AP three RSSI
```

```
ap_3_dBm = []
```

```
#List for AP four information
```

```
ap_4_bin = []
```

```
#List for AP four RSSI
```

```
ap_4_dBm = []
```

```
#Counter for beacon frames for AP one
```

```
ap_1_beacon_num = 0
```

```
#Counter for beacon frames for AP two
```

```
ap_2_beacon_num = 0
```

```
#Counter for beacon frames for AP three
```

```
ap_3_beacon_num = 0
```

```
#Counter for beacon frames for AP four
```

```
ap_4_beacon_num = 0
```

```
#Counter for beacon frames for all APs
```

```
ap_dBm_total = []
```

```
#Row counter
```

```
i = 0
```

```
#Turn data frame into a tuple so ID, timestamp, and MAC address can be accessed
```

```
while i < len(df):
```

```
    ap_tuple = df.iloc[i]
```

```
    #RSSI is the 5th element in the tuple
```

```
    ap_dBm = ap_tuple[5]
```

```
    #BSSID is the 7th element in the tuple
```

```
    ap_mac_addr = ap_tuple[7]
```

```
    #Perform a string match with the first AP one BSSID
```

```
    if ap_mac_addr == "XX:XX:XX:XX:XX:XX":
```



```

#Place "ap_tuple" into "ap_1_bin" list
ap_1_bin.append(ap_tuple)

#Place "ap_dBm" into "ap_1_dBm" list
ap_1_dBm.append(ap_dBm)

#Place "ap_dBm" into "ap_dBm_total" list
ap_dBm_total.append(ap_dBm)

#Increment beacon frame counter for AP one
ap_1_beacon_num += 1

#Perform a string match with the second AP one BSSID
elif ap_mac_addr == "XX:XX:XX:XX:XX:XX":
    ap_1_bin.append(ap_tuple)
    ap_1_dBm.append(ap_dBm)
    ap_dBm_total.append(ap_dBm)
    ap_1_beacon_num += 1

#Perform a string match with the first AP two BSSID
elif ap_mac_addr == "XX:XX:XX:XX:XX:XX":
    ap_2_bin.append(ap_tuple)
    ap_2_dBm.append(ap_dBm)
    ap_dBm_total.append(ap_dBm)
    ap_2_beacon_num += 1

#Perform a string match with the second AP two BSSID
elif ap_mac_addr == "XX:XX:XX:XX:XX:XX":
    ap_2_bin.append(ap_tuple)
    ap_2_dBm.append(ap_dBm)
    ap_dBm_total.append(ap_dBm)
    ap_2_beacon_num += 1

#Perform a string match with the first AP three BSSID
elif ap_mac_addr == "XX:XX:XX:XX:XX:XX":
    ap_3_bin.append(ap_tuple)
    ap_3_dBm.append(ap_dBm)
    ap_dBm_total.append(ap_dBm)
    ap_3_beacon_num += 1

#Perform a string match with the second AP three BSSID
elif ap_mac_addr == "XX:XX:XX:XX:XX:XX":
    ap_3_bin.append(ap_tuple)
    ap_3_dBm.append(ap_dBm)
    ap_dBm_total.append(ap_dBm)

```

```

    ap_3_beacon_num += 1

#Perform a string match with the first AP four BSSID
elif ap_mac_addr == "XX:XX:XX:XX:XX:XX":
    ap_4_bin.append(ap_tuple)
    ap_4_dBm.append(ap_dBm)
    ap_dBm_total.append(ap_dBm)
    ap_4_beacon_num += 1

#Perform a string match with the second AP four BSSID
elif ap_mac_addr == "XX:XX:XX:XX:XX:XXX":
    ap_4_bin.append(ap_tuple)
    ap_4_dBm.append(ap_dBm)
    ap_dBm_total.append(ap_dBm)
    ap_4_beacon_num += 1

#Catch any beacon frames that aren't a part of APs one through four
else:
    print("Not Match")

#Increment row counter
i += 1

#Create dataframe for AP one
df_ap_1_dBm = pd.DataFrame({'Access Point 1 dBm': ap_1_dBm})

#Create dataframe for AP two
df_ap_2_dBm = pd.DataFrame({'Access Point 2 dBm': ap_2_dBm})

#Create dataframe for AP three
df_ap_3_dBm = pd.DataFrame({'Access Point 3 dBm': ap_3_dBm})

#Create dataframe for AP four
df_ap_4_dBm = pd.DataFrame({'Access Point 4 dBm': ap_4_dBm})

#Create dataframe for all APs
df_dBm_total = pd.DataFrame({'All Access Points dBm': ap_dBm_total})

#Calculate mean, median for histogram for RSSIs for AP one
mu, sigma = norm.fit(ap_1_dBm)
print("Mu is," mu)
print("Sigma is," sigma)
#Plot histogram using "ap_1_dBm" list
plt.hist(ap_1_dBm, 20, facecolor='green', alpha=0.75)

```

```

#Label x axis
plt.xlabel('Signal Strength (dBm)')

#Label y axis
plt.ylabel('Frequency')

#Create title for graph
plt.title("Access Point 1")

#Show grid on plot
plt.grid(True)

#Display graph to screen
plt.show()

#Calculate mean, median for histogram for RSSIs for AP two
mu, sigma = norm.fit(ap_2_dBm)
print("Mu is," mu)
print("Sigma is," sigma)
plt.hist(ap_2_dBm, 20, facecolor='green', alpha=0.75)
plt.xlabel('Signal Strength (dBm)')
plt.ylabel('Frequency')
plt.title("Access Point 2")
plt.grid(True)
plt.show()

#Calculate mean, median for histogram for RSSIs for AP three
mu, sigma = norm.fit(ap_3_dBm)
print("Mu is," mu)
print("Sigma is," sigma)
plt.hist(ap_3_dBm, 20, facecolor='green', alpha=0.75)
plt.xlabel('Signal Strength (dBm)')
plt.ylabel('Frequency')
plt.title("Access Point 3")
plt.grid(True)
plt.show()

#Calculate mean, median for histogram for RSSIs for AP four
mu, sigma = norm.fit(ap_4_dBm)
print("Mu is," mu)
print("Sigma is," sigma)
plt.hist(ap_4_dBm, 20, facecolor='green', alpha=0.75)
plt.xlabel('Signal Strength (dBm)')
plt.ylabel('Frequency')
plt.title("Access Point 4")

```

```

plt.grid(True)
plt.show()

#Calculate mean, median for histogram for RSSIs for all APs
mu, sigma = norm.fit(ap_dBm_total)
print("Mu is," mu)
print("Sigma is," sigma)
plt.hist(ap_dBm_total, 20, facecolor='green', alpha=0.75)
plt.xlabel('Signal Strength (dBm)')
plt.ylabel('Frequency')
plt.title("All Access Points")
plt.grid(True)
plt.show()

#Plot histogram of RSSIs for all APs, broken down by originating AP
#Use Seaborn Deep graph style
plt.style.use('seaborn-deep')

#Plot the histogram using data from "ap_1_dBm," "ap_2_dBm," "ap_3_dBm,"
"ap_4_dBm"

#Create a key on the graph labeling APs one through four
fig = plt.hist([ap_1_dBm, ap_2_dBm, ap_3_dBm, ap_4_dBm], histtype = 'bar', align =
'mid', label = ['Access Point 1', 'Access Point 2', 'Access Point 3', 'Access Point 4'])

#Label x axis
plt.xlabel('RSSI (dBm)')

#Label y axis
plt.ylabel('Beacon Frames Received')

#Place graph key in the upper right of the graph
plt.legend(loc='upper right')

#Show grids on plot
plt.grid(True)

#Tighten graph layout
plt.tight_layout()

#Save the graph as "dBm Histogram.png"
plt.savefig('dBm Histogram.png')

#Display graph to screen
plt.show()

```

LIST OF REFERENCES

- [1] J. D. Roth, J. Martin, and T. Mayberry, “A graph-theoretic approach to virtual access point correlation,” in *2017 IEEE Conference on Communications and Network Security (CNS)*, Las Vegas, NV, USA, 2017, pp. 1–9. [Online]. Available: <https://ieeexplore.ieee.org/document/8228645>
- [2] Department of Homeland Security. “A guide to securing networks for Wi-Fi (IEEE 802.11 Family),” Department of Homeland Security, Washington, DC, USA, 2017. [Online]. Available: https://www.us-cert.gov/sites/default/files/publications/A_Guide_to_Securing_Networks_for_Wi-Fi.pdf
- [3] K. A. Scarfone, D. Dicoi, M. Sexton, and C. Tibbs, “Guide to securing legacy IEEE 802.11 wireless networks,” National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST SP 800–48r1, 2008. [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-48r1.pdf>
- [4] M. P. Singh, M. M. Mishra, and M. P. N. Barwal, “Analysis of security issues and their solutions,” in *National Conference on Computational and Mathematical Sciences*, 2014. pp 1–7. [Online]. <https://doi.org/10.13140/2.1.1061.3448>
- [5] M. Gast, *802.11 Wireless Networks: The Definitive Guide*, 2nd ed. Sebastapol, CA USA: O’Reilly, 2005.
- [6] *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*. IEEE Standard 802.11-2016. Accessed February 13, 2019. [Online]. Available: https://standards.ieee.org/standard/802_11-2016.html.
- [7] *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*. IEEE Standard 802.11-2016 (Revision of IEEE Std 802.11-2012), pp.1-3534, 14 Dec. 2016, <https://doi.org/10.1109/IEEESTD.2016.7786995>
- [8] M. Gast, “802.11 framing in detail,” in *802.11 Wireless Networks: The Definitive Guide*, 2nd Edition. Sebastapol, CA, USA: O’Reilly, 2005, pp. 69–116. [Online]. Available: <https://www.oreilly.com/library/view/80211-wireless-networks/0596100523/ch04.html>
- [9] D. West, *Introduction to Graph Theory*, 2nd ed. New York, NY, USA: Pearson, 2001.

- [10] N. Biggs, *Algebraic Graph Theory*, 2nd edition. New York, NY, USA: Cambridge University Press, 1993. [Online]. Kindle edition.
- [11] M. Girvan and M. E. J. Newman, “Community structure in social and biological networks,” *Proc. Natl. Acad. Sci.*, vol. 99, no. 12, pp. 7821–7826, Jun. 2002. [Online]. Available: <https://www.pnas.org/content/99/12/7821>
- [12] M. E. J. Newman, “Modularity and community structure in networks,” *Proc. Natl. Acad. Sci.*, vol. 103, no. 23, p. 8577, Jun. 2006. [Online]. Available: <https://www.pnas.org/content/103/23/8577>
- [13] A. Clauset, M. E. J. Newman, and C. Moore, “Finding community structure in very large networks,” *Phys. Rev. E*, vol. 70, no. 6, Dec. 2004. [Online]. Available: <https://arxiv.org/abs/cond-mat/0408187>
- [14] S. He, T. Hu, and S.-H. G. Chan, “Toward Practical deployment of fingerprint-based indoor localization,” *IEEE Pervasive Comput.*, vol. 16, no. 2, pp. 76–83, Apr. 2017. [Online]. Available: <https://ieeexplore.ieee.org/document/7891107>
- [15] Wireshark. “Wireshark.” Accessed January 13, 2019. [Online]. Available: <https://www.wireshark.org/>
- [16] Wireshark. “tshark—The Wireshark network analyzer 2.6.6.” Accessed January 13, 2019. [Online]. Available: <https://www.wireshark.org/docs/man-pages/tshark.html>
- [17] M. Bastian, S. Heymann, and M. Jacomy, “Gephi : An Open source software for exploring and manipulating networks,” p. 2. [Online]. Available: <https://gephi.org/publications/gephi-bastian-feb09.pdf>
- [18] V. Blondel, J. L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” *J Stat Mech: Theory Exp*, vol 2008, pp 1–12, Jul. 2008. [Online]. Available: <https://iopscience.iop.org/article/10.1088/1742-5468/2008/10/P10008/meta>
- [19] Python. “About.” Accessed January 13, 2019. [Online]. Available: <https://www.python.org/about/>
- [20] IEEE. “Guidelines for use of extended unique identifier (EUI), organizationally unique identifier (OUI), and company ID (CID),” 2017. [Online]. Available: <https://standards.ieee.org/content/dam/ieee-standards/standards/web/documents/tutorials/eui.pdf>
- [21] Wireshark. “CaptureSetup/WLAN—The Wireshark Wiki.” [Online]. Accessed January 13, 2019. Available: <https://wiki.wireshark.org/CaptureSetup/WLAN>

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California