Device Priority Manager for Bandwidth Allocation
Operating Systems Course Project

By

Zayan Zubair(22BCE1100)
Amrutha P(22BCE1058)

A project report submitted to

Dr. M. Braveen

Assistant Professor, School of Computer Science and Engineering

in partial fulfillment of the requirements for the course of

BCSE303L Operating Systems

in

B. TECH., COMPUTER SCIENCE AND ENGINEERING

Vellore Institute of Technology, Chennai

Vandalur – Kelambakkam Road

BONAFIDE CERTIFICATE

This is to certify that the Project work titled "Device Priority Manager for Bandwidth Allocation" that is being submitted by 22BCE1100(Zayan Zubair) and 22BCE1058(Amrutha P) is in partial fulfillment of the requirements for the award of Bachelor of Technology in Computer Science and Engineering, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Dr. M. BRAVEEN

Guide

Abstract

The Wi-Fi Priority Manager project introduces a robust system-level solution for optimizing Wi-Fi network connectivity within the operating system environment. Leveraging the efficiency of C programming, the project focuses on developing a Priority Round Robin algorithm for bandwidth allocation, addressing the critical challenge of fair and prioritized distribution of network resources. The system's core functionalities include a user-friendly interface that empowers users to define and manage Wi-Fi network priorities effortlessly. Sorting strategies, such as bubble sort and `qsort`, maintain a systematic order of devices based on priority levels, ensuring the effectiveness of the Priority Round Robin algorithm.

The project's successful implementation results in a reliable, efficient, and accessible Wi-Fi Priority Manager, enhancing user control over network connections. Furthermore, proposed alterations, such as advanced sorting strategies and machine learning-based optimization, position the project for adaptability and future advancements. The system's adaptability and potential for customization align with the growing demands of network resource management. Looking forward, future works may involve refining sorting strategies, integrating real-time adaptive features, and incorporating user feedback for enhanced customization, aiming to continuously elevate the project's efficiency and user experience in the ever-evolving landscape of network connectivity.

Keywords :Wi-Fi Priority Manager, C programming, Priority Round Robin,Bandwidth Allocation,User-Friendly Interface,Network Optimization

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The Wi-Fi Priority Manager project introduces a pioneering solution to the common challenges associated with Wi-Fi network connectivity. In an era dominated by digital connectivity, users often encounter issues of inconsistent network performance and connectivity disruptions, especially in environments with multiple available Wi-Fi networks. This project addresses these concerns by providing a user-friendly tool that empowers individuals to prioritize and manage their Wi-Fi networks effectively. The usefulness of this project becomes evident in its ability to enhance user control over network resources, ensuring a seamless and tailored connectivity experience. By implementing a sophisticated Priority Round Robin algorithm, the Wi-Fi Priority Manager dynamically allocates bandwidth to known networks based on user-assigned priority levels, resolving the inconvenience of unreliable network connections.

The scope of the Wi-Fi Priority Manager extends beyond conventional network management approaches. It introduces a novel paradigm by combining the efficiency of system-level programming, represented by the C language, with an intuitive user interface. The project's novelty lies in its ability to adaptively allocate resources, prioritize networks according to user preferences, and provide a tailored connectivity experience. As an innovative solution in the domain of network management, the Wi-Fi Priority Manager not only addresses existing problems but also anticipates future connectivity challenges, positioning itself as a forward-looking tool for optimizing connectivity experiences within the operating system environment.

CHAPTER II

RELATED WORKS

Within this section, we delve into notable projects and utilities that are pertinent to Wi-Fi scheduling and resource allocation, shedding light on the intricate landscape of connectivity management. Through a detailed examination of these initiatives, a nuanced and comprehensive understanding begins to unfold, serving as the bedrock for future advancements in Wi-Fi prioritization. These endeavors showcase the ongoing evolution in the field, illustrating innovative approaches and strategies aimed at optimizing the allocation of resources within Wi-Fi networks for enhanced efficiency and performance.

1.ConnMan(Connection Manager):

ConnMan stands out as a robust Linux connection manager, providing a flexible and efficient framework for network management. While not originally designed for prioritizing connections, its modular architecture allows for potential extensions and customization. ConnMan's proficiency in handling diverse network connections makes it an ideal starting point for incorporating Wi-Fi prioritization seamlessly. By leveraging its adaptable structure, developers have the opportunity to enhance the system with features that prioritize Wi-Fi connections based on user-defined criteria, contributing to an optimized and responsive network management experience. ConnMan's versatility positions it as a valuable asset for refining and extending network prioritization functionalities.

2. Network Manager:

NetworkManager, a widely utilized Linux utility, plays a pivotal role in orchestrating both wired and wireless network connections. While it does not explicitly prioritize scheduling, its sophisticated approach to network management positions it as a valuable and versatile resource. The utility streamlines the coordination of diverse network configurations, offering dynamic detection and configuration of network interfaces. Users benefit from a user-friendly interface

that enables efficient management of various aspects of network connectivity. Despite its emphasis on ease of use, NetworkManager does not compromise on security, supporting a range of protocols to ensure a reliable and protected network environment.

Understanding the nuances of NetworkManager becomes imperative for a comprehensive perspective on Wi-Fi connectivity within Linux environments. Beyond mere connection establishment, NetworkManager handles intricate tasks such as VPN integration, network device management, and dynamic adjustments to changing network conditions. This multifaceted utility caters to the needs of users and system administrators alike, contributing to the efficiency and reliability of network operations in Linux systems. Whether navigating wireless hotspots, managing Ethernet connections, or seamlessly integrating with virtual private networks, NetworkManager stands as a robust facilitator of connectivity management, reinforcing its significance in the Linux landscape.

3. OpenWRT(Open Wireless Router):

OpenWRT, an open-source firmware designed for routers, stands as a versatile platform catering to the needs of embedded systems. Recognized for its robustness and flexibility, OpenWRT allows users to customize and tailor the firmware according to their specific requirements. Although OpenWRT does not directly address scheduling concerns, its adaptability forms a strong foundation for the exploration of Wi-Fi management in various environments. By offering an open and customizable framework, OpenWRT empowers users to delve into the intricacies of wireless network management, providing opportunities to optimize performance, security, and resource allocation.

In the context of Wi-Fi management, OpenWRT's customizable nature becomes particularly valuable. It opens avenues for researchers and network administrators to experiment with different algorithms, policies, and protocols to enhance the prioritization of network connections. This adaptability is crucial in addressing the evolving needs of modern networks, where efficient and intelligent Wi-Fi management is vital for delivering a seamless user experience. As the demand for reliable and high-performance wireless connectivity continues to grow,

OpenWRT's role as a foundation for exploration becomes increasingly significant, fostering advancements in the field of network management and optimization.

4. Free RTOS(Real-Time Operating System):

In the realm of embedded systems, FreeRTOS, a real-time operating system kernel, stands out as a versatile and robust solution. While FreeRTOS may not have Wi-Fi as its primary focus, its real-time scheduling principles serve as a crucial foundation for enhancing the performance of embedded systems, including those involving Wi-Fi connectivity. Real-time operating systems are designed to provide predictable and deterministic response times to events, making them particularly valuable in applications where timing is critical. This inherent predictability in FreeRTOS allows for efficient task scheduling, ensuring that essential processes, even those related to Wi-Fi interactions, are executed in a timely and reliable manner.

The insights garnered from FreeRTOS extend beyond its immediate scope, influencing strategic decisions on resource allocation within embedded systems. In the context of Wi-Fi connectivity, these principles contribute to creating a responsive and efficient communication environment. While FreeRTOS itself may not offer explicit features for Wi-Fi, the disciplined approach it brings to real-time task management aids in optimizing the utilization of system resources, thereby supporting the development of embedded systems that demand reliable and timely Wi-Fi interactions. The utilization of FreeRTOS, with its emphasis on real-time scheduling, thus proves instrumental in achieving seamless and responsive Wi-Fi connectivity within the constraints of embedded environments.

5.Android Wi-Fi Connection Manager:

The Android Wi-Fi Connection Manager, developed in Java, stands as a pivotal component for mobile Wi-Fi management. Despite its implementation in Java rather than C, the system offers invaluable insights into the creation of user-friendly interfaces and considerations tailored for mobile environments. A comprehensive understanding of its functionality becomes instrumental in designing interfaces that resonate with a diverse user base, thereby elevating the accessibility and attractiveness of Wi-Fi prioritization features on mobile platforms.

The Android Wi-Fi Connection Manager underscores the significance of a language-agnostic approach, emphasizing the broader principles of mobile interface design. Its utilization of Java showcases the adaptability of programming languages in achieving user-centric objectives. This insight not only enriches the developer's toolkit but also emphasizes the importance of prioritizing a seamless and engaging user experience in the domain of mobile Wi-Fi management.

6. Wpa-supplicant:

The wpa_supplicant utility plays a crucial role in the management of Wi-Fi connections within the Linux environment. While not designed as a complete scheduling solution, its primary function lies in overseeing the authentication and association processes, serving as a key component for establishing secure and reliable Wi-Fi connections. Delving into the intricacies of wpa_supplicant's inner workings provides valuable insights into the foundational elements of Wi-Fi connectivity, offering essential context for the development of advanced tools such as a Wi-Fi Priority Manager.

Understanding the nuances of wpa_supplicant is essential for crafting effective Wi-Fi management solutions. Its contribution to authentication and association processes highlights its significance in the establishment of secure and seamless Wi-Fi connections. This insight forms the basis for developing sophisticated tools like the Wi-Fi Priority Manager, facilitating enhanced control and prioritization of network connections in Linux environments.

7. Wi-Fi Scheduler Apps for Mobile Devices:

Mobile applications developed for Wi-Fi scheduling reveal a growing trend in catering to user preferences for personalized control. While these projects may not be system-level undertakings, they underscore the pivotal role of user-friendly interfaces. A deeper examination of these apps not only sheds light on the nuances of Wi-Fi management but also yields valuable insights for crafting intuitive interfaces. This emphasis on user empowerment enhances the overall mobile

experience, demonstrating the significance of thoughtful design in meeting evolving user demands.

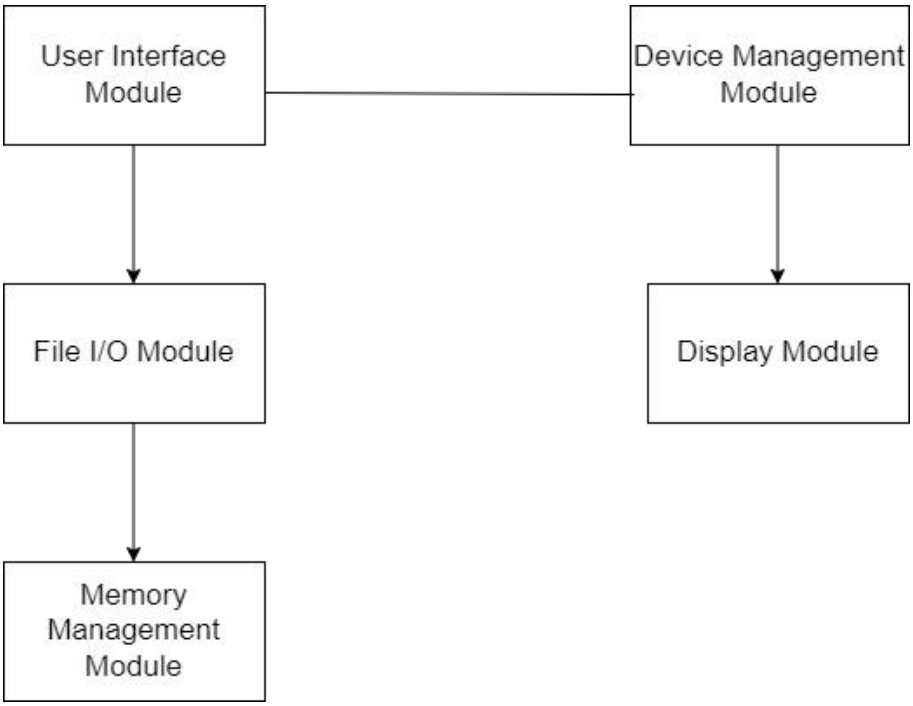
In the realm of Wi-Fi scheduling apps, customization is a key driver, reflecting a user-centric approach. Despite their scale, these applications prioritize simplicity and ease of use. The user-friendly interfaces implemented in such apps exemplify the critical role of design in enhancing user experience. By delving into the intricacies of Wi-Fi prioritization, these applications provide a blueprint for creating intuitive interfaces that empower users to effortlessly manage their connectivity preferences. This nuanced understanding, derived from the analysis of mobile apps, contributes valuable insights for designing user-centric interfaces, ensuring a seamless and tailored experience for users managing Wi-Fi priorities on their mobile devices.

EXPECTED RESULT

The expected result of this project is the development and implementation of a robust Wi-Fi Priority Manager with a focus on simplicity, efficiency, and user-friendliness. Users will gain the ability to define priority levels for their known Wi-Fi networks, leveraging Preemptive Priority scheduling and a round-robin algorithm. The key outcome is seamless and automatic connectivity to the highest-priority network, ensuring an optimal and uninterrupted Wi-Fi experience. The user-friendly interface will empower individuals with varying technical backgrounds to easily manage their Wi-Fi network priorities according to personal preferences. Overall, the project aims to deliver a reliable, efficient, and accessible Wi-Fi Priority Manager, providing users with greater control over their connectivity experiences within the operating system environment.

CHAPTER III
ARCHITECTURE

High level Architecture Design:



User Interface Module:

The User Interface Module plays a crucial role in the Wi-Fi Priority Manager by taking charge of user inputs and information display. Its primary responsibility is to present a user-friendly, menu-driven interface, facilitating user interaction and capturing inputs for various profile management operations. This module guides users through available actions and ensures a seamless experience while interacting with the program.

Device Management Module:

The Device Management Module serves as the backbone of the system, responsible for efficiently managing Wi-Fi profiles and their configurations. Leveraging the `readTextFile` function, it retrieves initial device information from a text file during program initialization. Key functions like `add_device`, `remove_device`, and `edit_device` are implemented to enable effective profile management. Additionally, the module allocates bandwidth to devices based on priority levels using the `allocateBandwidth` function. To maintain an ordered list of devices, sorting algorithms are employed, prioritizing devices based on their assigned priority levels.

File I/O Module:

Handling critical file input/output operations, the File I/O Module ensures seamless communication between the program and external files. During program initialization, it uses the `readTextFile` function to retrieve initial data from a text file, and as profiles are modified, it updates the text file to accurately reflect changes in device configurations.

Display Module:

The Display Module is responsible for presenting device information in a readable format. By employing functions such as `displayDevice` and `displayDeviceWithBandwidth`, this module provides a clear representation of device priorities, allocated bandwidth, and network details. The effective display enhances user understanding of the system's current state.

Memory Management Module:

Efficient memory management is achieved through the Memory Management Module, which handles dynamic memory allocation and deallocation. This module ensures optimal memory usage by dynamically allocating memory for storing device profiles and subsequently deallocating it to prevent memory leaks.

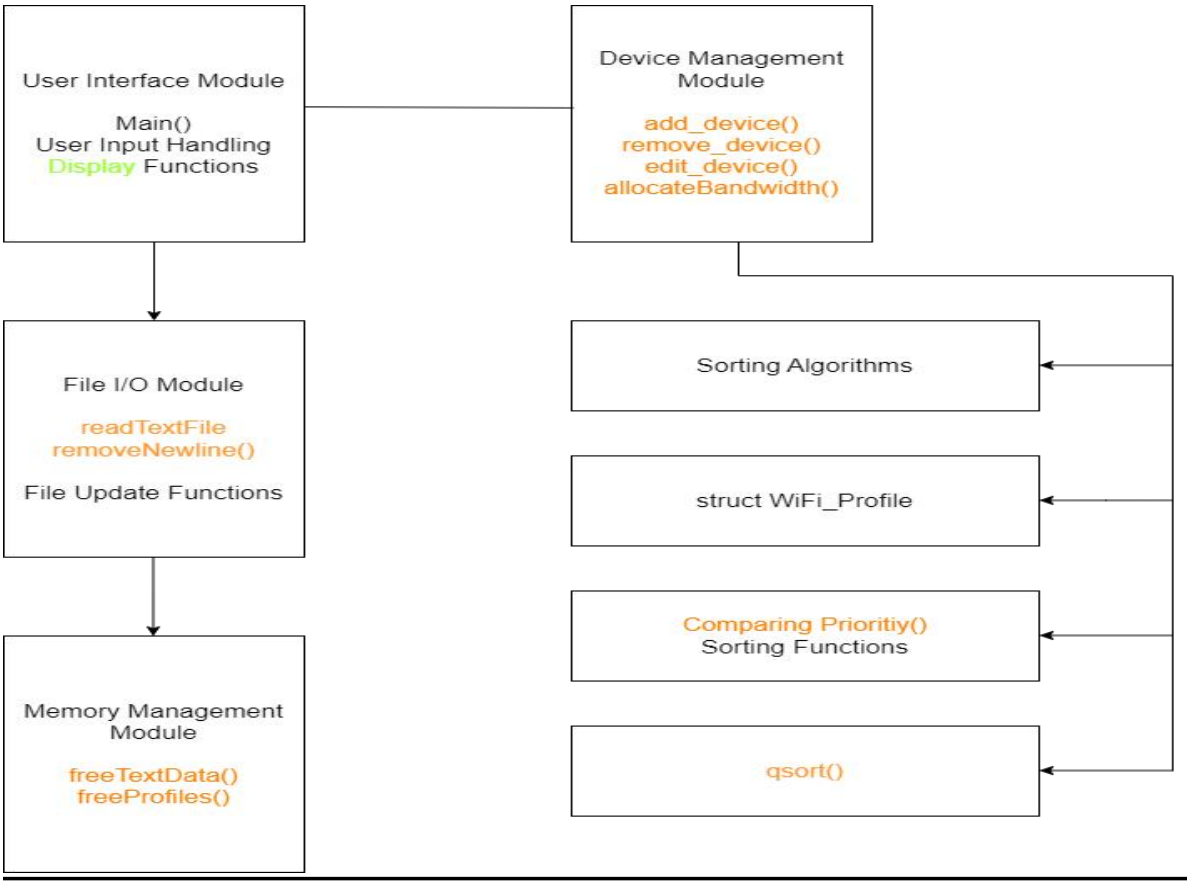
Interactions:

The various modules within the Wi-Fi Priority Manager interact seamlessly to create a cohesive system. The User Interface Module collaborates with the Device Management Module to facilitate user inputs, while the Device Management Module communicates with the File I/O Module for data retrieval and updates. Interactions with the Display Module ensure that device information is effectively presented to the user. The Memory Management Module plays a critical role in maintaining efficient memory usage throughout the program's execution.

Benefits:

The architecture offers several benefits, including modularity, which allows for easy maintenance and scalability. The user-friendly interface enhances the overall user experience, while dynamic memory allocation prevents memory-related issues. The reliance on a file-based configuration simplifies initialization and updates through a text file, contributing to the system's flexibility and ease of use.

Low Level Architecture Design:



1.User Interface Module:

main():

The `main()` function serves as the entry point for the program, orchestrating the overall program flow. It initializes essential components and acts as a control center, directing the execution based on user inputs. Within this function, the program manages the user interface, ensuring a seamless and user friendly experience. It prompts users for actions, handles input validation, and calls corresponding functions from other modules.

User Input Handling:

User input handling functions are critical for maintaining program stability. These functions robustly process user inputs, ensuring that unexpected or invalid inputs do not compromise the system. Through proper input validation and error handling

mechanisms, the User Input Handling section contributes to the overall reliability of the Wi Fi Priority Manager.

Display Functions:

The Display Functions play a pivotal role in presenting information to users in a comprehensible format. These functions are responsible for rendering device details, allocated bandwidth, and other relevant information on the console. By providing clear and organized displays, the Wi Fi Priority Manager enhances user interaction and overall usability.

2.Device Management Module:

add_device():

The `add_device()` function allows users to include a new device profile in the system. It accepts user defined parameters such as device name and network details, creating a new profile. This function ensures that the new device is seamlessly integrated into the existing priority system, maintaining order and facilitating fair bandwidth allocation.

remove_device():

`remove_device()` removes a device profile based on the specified device name. It systematically updates the priority levels of remaining devices, ensuring the consistency of the priority order. This function is crucial for maintaining an accurate representation of the network's device hierarchy.

edit_device():

The `edit_device()` function enables users to modify the priority level of an existing device. This ensures flexibility in adjusting the priority order based on user preferences. The function systematically adjusts the priority levels of other devices, preserving a coherent and organized hierarchy.

allocateBandwidth():

The `allocateBandwidth()` function serves as the cornerstone for ensuring an equitable distribution of network resources within the Wi Fi Priority Manager. Employing a Priority Round Robin algorithm, this function systematically allocates available bandwidth among devices, taking into account their assigned priority levels. The algorithm commences by sorting devices based on their priorities, establishing a clear hierarchy. Subsequently, the total available bandwidth is divided into cycles, with each device receiving a proportional share during its turn in the round robin sequence. This ensures that higher priority devices are afforded a larger portion of the available bandwidth, reflecting their elevated significance in the network. The process continues in subsequent cycles, adapting dynamically to changes in device priorities and guaranteeing an ongoing, fair allocation of bandwidth.

The implementation of Priority Round Robin in `allocateBandwidth()` intricately calculates the bandwidth allocated to each device per cycle, considering the device's priority level. The function iterates through the sorted list of devices, allocating bandwidth in a round robin manner. Importantly, the algorithm allows for the equitable distribution of remaining bandwidth among devices, optimizing overall network efficiency. This approach not only ensures fairness in bandwidth allocation but also efficiently prioritizes higher priority devices. By dynamically adapting to changes in device priorities and preventing resource underutilization, the `allocateBandwidth()` function optimally manages network resources, enhancing the connectivity experience within the operating system environment.

Sorting Algorithms:

Sorting algorithms within the Device Management Module, including bubble sort and `qsort`, are instrumental in maintaining the order of devices based on their priority levels. These algorithms systematically arrange devices, optimizing the efficiency of operations such as adding, removing, or editing device profiles.

Data Structure (struct WiFi_Profile):

The `struct WiFi_Profile` serves as the core data structure representing a Wi Fi profile. It encapsulates device specific details, including the device name, priority level, network details, and allocated bandwidth. This structured approach to data organization ensures easy access and manipulation of device information within the system.

3.File I/O Module:

readTextFile():

`readTextFile()` is responsible for reading initial device information from a text file during program initialization. This function ensures that the Wi Fi Priority Manager starts with accurate and up to date device profiles. By reading data from an external file, the program achieves a dynamic and configurable nature.

removeNewline():

The `removeNewline()` function is crucial for proper string formatting. It eliminates newline characters from strings, preventing unexpected line breaks in the displayed information. This contributes to the clarity and readability of the output presented to the user.

File Update Functions:

File Update Functions manage the updating of the text file to reflect changes in device profiles. When users add, remove, or edit device profiles, these functions ensure that the external data file accurately represents the current state of the network. This guarantees consistency between the program's internal data and the external storage medium.

4. Memory Management Module:

freeTextData():

`freeTextData()` is responsible for freeing the memory allocated for reading text data during program execution. This function prevents memory leaks associated with text data, contributing to the overall stability and efficiency of the Wi Fi Priority Manager.

freeProfiles():

`freeProfiles()` ensures the proper release of memory allocated for storing device profiles. As devices are added, removed, or edited, this function manages memory deallocation to prevent memory related issues and maintain optimal system performance.

Additional Considerations:

String Manipulation:

String manipulation functions, including `strcpy` and `strcat`, are utilized for effective management of strings within the program. These functions contribute to the accurate representation of device names and network details, enhancing the overall integrity of the system.

Dynamic Memory Allocation:

The use of the `malloc` function for dynamic memory allocation is essential for preventing memory related issues. This function ensures that memory is allocated as needed during runtime, optimizing the program's memory usage and minimizing the risk of memory leaks.

Sorting Algorithms:

The implementation of sorting functions, including bubble sort and `qsort`, plays a crucial role in maintaining the order of devices based on priority levels. These algorithms contribute to the efficiency and accuracy of operations within the Wi Fi Priority Manager, ensuring a consistently organized network hierarchy.

CHAPTER IV
ALGORITHM AND MATHEMATICAL MODEL

Algorithm:

1. Priority Round Robin Algorithm (allocateBandwidth()):

- Description: The Priority Round Robin algorithm is employed in the `allocateBandwidth()` function to ensure the fair distribution of bandwidth among devices based on their assigned priority levels. This algorithm systematically cycles through devices, allocating bandwidth proportionally and prioritizing higher-priority devices with larger shares. The use of sorting algorithms ensures that devices are ordered based on priority before bandwidth allocation begins.

- Purpose: The algorithm optimizes network resource utilization by preventing any single device from monopolizing bandwidth. It dynamically adapts to changes in device priorities, accommodating modifications made by users through device management functionalities. This approach enhances the overall efficiency of the Wi-Fi Priority Manager by striking a balance between fairness and prioritization.

```
void allocateBandwidth(WiFi_Profile *wifi_profiles, int num_profiles, int total_bandwidth) {
    for (int i = 0; i < num_profiles - 1; i++) {
        for (int j = 0; j < num_profiles - i - 1; j++) {
            if (wifi_profiles[j].priority_level > wifi_profiles[j + 1].priority_level) {
                WiFi_Profile temp = wifi_profiles[j];
                wifi_profiles[j] = wifi_profiles[j + 1];
                wifi_profiles[j + 1] = temp;
            }
        }
    }

    int current_bandwidth = total_bandwidth;
    for (int i = 0; i < num_profiles; i++) {
        int bandwidth_per_device = ((num_profiles - wifi_profiles[i].priority_level + 1) * total_bandwidth)/((num_profiles*(num_profiles + 1)) / 2);
        wifi_profiles[i].allocated_bandwidth = bandwidth_per_device;
        current_bandwidth -= bandwidth_per_device;
    }
}
```

2. Sorting Algorithms (Bubble Sort and qsort):

- Description: Sorting algorithms, such as bubble sort and `qsort`, are utilized within the Device Management Module to maintain the order of devices based on their priority levels. Bubble sort is applied for its simplicity, while the C standard library's `qsort` function provides a more efficient sorting mechanism.

- Purpose: Sorting ensures a consistent order of devices, facilitating efficient operations such as adding, removing, or editing device profiles. The sorted order is crucial for the fair execution of the Priority Round Robin algorithm during bandwidth allocation.

Mathematical Model:

1. Bandwidth Allocation Model:

- Description: The mathematical model for bandwidth allocation involves dividing the total available bandwidth into cycles, with each cycle representing an opportunity for devices to access network resources. The bandwidth allocated to each device per cycle is calculated proportionally based on its priority level. Mathematically, this is represented as [`Bandwidth_per_Device = Total_Bandwidth  Priority_Level / Sum_of_Priorities`], ensuring that higher-priority devices receive larger portions.

- Purpose: The model aims to achieve equitable distribution, reflecting the relative importance of devices in the network. It provides a dynamic and adaptable framework for bandwidth allocation, responding to changes in device priorities and network conditions.

```
bandwidth_per_device = ((num_profiles - wifi_profiles[i].priority_level + 1) * total_bandwidth)/((num_profiles*(num_profiles + 1)) / 2);
```

Proposed Alterations:

1. Enhanced Sorting Strategies:

- Description: While the current implementation uses basic sorting algorithms like bubble sort and `qsort`, there is room for enhancement with more sophisticated sorting strategies. Algorithms like merge sort or insertion sort could be explored for improved efficiency, especially as the number of devices scales.

- Purpose: The proposed alteration aims to optimize the sorting process, particularly in scenarios with a large number of devices, contributing to overall system performance.

2. Dynamic Bandwidth Adjustment:

- Description: Proposing a mechanism for dynamic adjustment of bandwidth allocation based on real-time network conditions and device usage patterns. This could involve monitoring factors such as network congestion, device activity, and user preferences to dynamically adapt bandwidth distribution.

- Purpose: The alteration aims to enhance the adaptability of the Wi-Fi Priority Manager, ensuring optimal bandwidth allocation in response to changing network dynamics and user demands.

Altered Components:

1. Flexible Priority Management:

- Description: The priority management system could be altered to support more flexible priority adjustments, allowing users to define custom priority levels or utilize dynamic priorities based on contextual factors.

- Purpose: This alteration provides users with greater control and customization options, aligning the Wi-Fi Priority Manager more closely with individual preferences and specific network usage scenarios.

2. Machine Learning-Based Optimization:

- Description: Proposing the integration of machine learning algorithms to predict and optimize bandwidth allocation based on historical usage patterns, user behavior, and network conditions.

- Purpose: The incorporation of machine learning aims to introduce predictive and adaptive capabilities, enhancing the efficiency and responsiveness of the Wi-Fi Priority Manager in allocating bandwidth.

These algorithmic and mathematical considerations, along with the proposed alterations and altered components, contribute to the ongoing refinement and adaptability of the Wi-Fi Priority Manager, ensuring an optimized and user-centric connectivity experience within the operating system environment.

CHAPTER V
SIMULATION AND IMPLEMENTATION

The implementation of the Wi-Fi Priority Manager project is realized through a robust C programming paradigm, leveraging system-level capabilities to optimize network resource management. The core functionalities are encapsulated within well-defined modules, including the User Interface, Device Management, File I/O, and Memory Management. The use of a Priority Round Robin algorithm in the `allocateBandwidth()` function ensures fair and proportional distribution of bandwidth among devices, with a focus on prioritizing higher-priority networks. Sorting algorithms, such as bubble sort and `qsort`, efficiently maintain device order based on priority levels. The implementation showcases meticulous attention to user-friendliness through a carefully crafted user interface, enabling users of varying technical backgrounds to intuitively manage Wi-Fi priorities. Additionally,

the project demonstrates a forward-looking approach with proposed alterations, such as enhanced sorting strategies and machine learning-based optimization, aiming to continually refine and adapt the Wi-Fi Priority Manager for an ever-evolving network environment. The code, as exemplified by the provided C implementation, combines the power of system-level programming with algorithmic efficiency to deliver a reliable, efficient, and accessible solution for network resource allocation.

Code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <windows.h>

#define MAX_STRING_LENGTH 100

struct TextData {
   char **lines;
   int numLines;
};

void readTextFile(const char *filename, struct TextData *data) {
   FILE *file = fopen(filename, "r");
   if (file == NULL) {
      perror("Error opening file");
      exit(EXIT_FAILURE);
   }

   char buffer[MAX_STRING_LENGTH];
   data->numLines = 0;

   while (fgets(buffer, sizeof(buffer), file) != NULL) {
      data->numLines++;
   }

   rewind(file);

   data->lines = (char **)malloc(data->numLines * sizeof(char *));
   if (data->lines == NULL) {
      perror("Memory allocation error");
      exit(EXIT_FAILURE);
   }

   for (int i = 0; i < data->numLines; ++i) {
```

```
        data->lines[i] = (char *)malloc(MAX_STRING_LENGTH * sizeof(char));
        if (data->lines[i] == NULL) {
            perror("Memory allocation error");
            exit(EXIT_FAILURE);
        }

        fgets(data->lines[i], MAX_STRING_LENGTH, file);
    }

    fclose(file);
}

void freeTextData(struct TextData *data) {
    for (int i = 0; i < data->numLines; ++i) {
        free(data->lines[i]);
    }
    free(data->lines);
}

typedef struct WiFi_Profile{
    char device_name[50];
    int priority_level;
    char network_details[100];
    int allocated_bandwidth;
} WiFi_Profile;

int comparePriority(const void *a, const void *b) {
    WiFi_Profile *profileA = (WiFi_Profile *)a;
    WiFi_Profile *profileB = (WiFi_Profile *)b;

    return profileB->priority_level - profileA->priority_level;
}


void allocateBandwidth(WiFi_Profile *wifi_profiles, int num_profiles, int total_bandwidth) {
    for (int i = 0; i < num_profiles - 1; i++) {
        for (int j = 0; j < num_profiles - i - 1; j++) {
            if (wifi_profiles[j].priority_level > wifi_profiles[j + 1].priority_level) {
                WiFi_Profile temp = wifi_profiles[j];
                wifi_profiles[j] = wifi_profiles[j + 1];
                wifi_profiles[j + 1] = temp;
            }
        }
    }

    int current_bandwidth = total_bandwidth;
    for (int i = 0; i < num_profiles; i++) {
        int bandwidth_per_device = ((num_profiles - wifi_profiles[i].priority_level + 1) * total_bandwidth) /
((num_profiles * (num_profiles + 1)) / 2);
```

```
        wifi_profiles[i].allocated_bandwidth = bandwidth_per_device;
        current_bandwidth -= bandwidth_per_device;
    }
}

void add_device(WiFi_Profile *wifi_profiles, int *num_profiles, WiFi_Profile new_profile) {
    wifi_profiles[*num_profiles] = new_profile;
    (*num_profiles)++;
}

void remove_device(WiFi_Profile *wifi_profiles, int *num_profiles, char *device_name) {
    int index_to_remove = -1;

    for (int i = 0; i < *num_profiles; i++) {

        if (strcmp(wifi_profiles[i].device_name, device_name) == 0) {
            index_to_remove = i;
            break;
        }
    }

    if (index_to_remove != -1) {
        for (int i = index_to_remove; i < (*num_profiles) - 1; i++) {
            wifi_profiles[i] = wifi_profiles[i + 1];
        }
        for(int i=index_to_remove;i<(*num_profiles);i++)
        {
            wifi_profiles[i].priority_level=wifi_profiles[i].priority_level-1;
        }
        (*num_profiles)--;
    }

}

void edit_device(WiFi_Profile *wifi_profiles, int num_profiles, char *device_name, int new_priority) {
    for (int i = 0; i < num_profiles; i++) {

        if (strcmp(wifi_profiles[i].device_name, device_name) == 0) {

            wifi_profiles[i].priority_level = new_priority;

            for(int k = i+1 ; k< num_profiles ; k++)
            {
                wifi_profiles[k].priority_level -=1;
            }

            for (int k = 0; k < num_profiles; k++) {
                for (int j = k+1; j < num_profiles; j++) {
                    if (wifi_profiles[k].priority_level > wifi_profiles[j].priority_level) {
```

```
                WiFi_Profile temp;
                temp = wifi_profiles[k];
                wifi_profiles[k] = wifi_profiles[j];
                wifi_profiles[j] = temp;
            }
        }
    }

    for (int k = 0; k < num_profiles; k++) {
        for (int j = k+1; j < num_profiles; j++) {
            if (wifi_profiles[k].priority_level == wifi_profiles[j].priority_level) {
                WiFi_Profile temp;
                temp = wifi_profiles[k];
                wifi_profiles[k] = wifi_profiles[j];
                wifi_profiles[j] = temp;
                break;
            }
        }
    }

    int z=-1;
    for(int k = 0 ; k<num_profiles ; k++)
    {
        if(wifi_profiles[k].priority_level== new_priority)
        {
            z=k;
            break;
        }
    }

    for(int k= z+1 ; k < num_profiles ; k++)
    {
        wifi_profiles[k].priority_level+=1;
    }

    break;
  }
 }

}

void displayDevice(const WiFi_Profile *profiles, int num_profiles) {
   printf("Device List:\n");
   printf("Device name \t Device Priority \t Ip Address\n");
   for (int i = 0; i < num_profiles; ++i) {
      printf("%s\t%d\t\t%s", profiles[i].device_name, profiles[i].priority_level, profiles[i].network_details);
   }
}
```

```
char* removeNewline(const char* original) {
    size_t length = strlen(original);

    if (length > 0 && original[length - 1] == '\n') {
        char* result = (char*)malloc(length);
        strncpy(result, original, length - 1);
        result[length - 1] = '\0';

        return result;
    } else {
        return strdup(original);
    }
}

void displayDeviceWithBandwidth(const WiFi_Profile *profiles, int num_profiles) {
    printf("Device List with Bandwidth:\n");

    printf("\n -----------------------------------------------------------------------------\n");
    printf("|                                                                 |\n");
    printf("|Device name \t Device Priority \t Allocated Bandwidth \t Ip Address   |\n");
    for (int i = 0; i < num_profiles; ++i) {

        strcpy(profiles[i].network_details, removeNewline(profiles[i].network_details));
        printf("|%s\t%d\t\t\t%d          mb\t\t%s          |\n",          profiles[i].device_name,          profiles[i].priority_level,
profiles[i].allocated_bandwidth, profiles[i].network_details );
    }
    printf("|                                                                 |\n");
    printf(" -----------------------------------------------------------------------------\n");
}

void freeProfiles(WiFi_Profile *profiles, int num_profiles) {
    free(profiles);
}

int main() {

    struct TextData textData;
    const char *filename = "dataset.txt";

    readTextFile(filename, &textData);

    int num_profiles = 0;
    int I = 11,c=0;
    WiFi_Profile profile[100];

    for(int i=0;i<10;i++)
    {

        char *device = textData.lines[i];
```

```
    char *token = strtok(device,";");

    WiFi_Profile new_profile;

    for(int j=0;j<sizeof(token);j++)
    {
       if(token[j]=='_')
       {
          token[j]=' ';
       }
    }

    strcpy(new_profile.device_name, token);
    token = strtok(NULL,";");
    new_profile.priority_level = i+1;
    strcpy(new_profile.network_details, token);

    add_device(profile, &num_profiles, new_profile);

}
freeTextData(&textData);


displayDevice(profile,num_profiles);


printf("MENU: \nPress the required option\n");
printf("1.Add a new device\n2.Remove a device\n3.Edit priority order\n");

while (1) {

    char a;
    printf("Would you like to continue?(y/n): ");
    scanf(" %c", &a);

    if (a == 'n' || a == 'N')
       break;

    else if (a == 'y' || a == 'Y') {

       int x;
       printf("\nEnter the preferred option: ");
       scanf("%d",&x);

       switch(x)
       {
          case 1: {
             WiFi_Profile add_profile;
             char device_Name[100];
```

```
    printf("\nEnter the device name: ");

    fflush(stdin);
    fgets(device_Name,sizeof(device_Name),stdin);

    if (  strlen(device_Name)>0 && device_Name[strlen(device_Name) - 1] == '\n') {
        device_Name[strlen(device_Name) - 1] = '\0';
    }
    int space=21-strlen(device_Name);
    for(int k=0;k<space;k++)
    {
        strcat(device_Name," ");
    }


    strcpy(add_profile.device_name, device_Name);
    add_profile.priority_level = I;

    int a=I*10;
    char b[4];
    char Ip[]=" 192.168.1.";
    snprintf(b,sizeof(b),"%d",a);
    strcat(Ip,b);
    strcat(Ip,"\n");
    strcpy(add_profile.network_details, Ip);
    add_device(profile, &num_profiles, add_profile);
    I=I+1;

    char f;
    printf("\nWould you like to see the updated list(y/n): ");
    scanf(" %c", &f);
    if (f == 'n' || f == 'N')
    {
        break;
    }
    else if (f == 'y' || f == 'Y')
    {
        displayDevice(profile,num_profiles);
    }

    break;
}
case 2: {
    char device_Name[100];
    printf("Enter the name of the device: ");
    fflush(stdin);
    fgets(device_Name, sizeof(device_Name), stdin);
```

```
    if (  strlen(device_Name)>0 && device_Name[strlen(device_Name) - 1] == '\n') {
        device_Name[strlen(device_Name) - 1] = '\0';
    }
    int space=21-strlen(device_Name);
    for(int k=0;k<space;k++)
    {
        strcat(device_Name," ");
    }

    remove_device(profile, &num_profiles, device_Name);

    char b;
    printf("\nWould you like to see the updated list(y/n): ");
    scanf(" %c", &b);
    if (b == 'n' || b == 'N')
    {
        break;
    }
    else if (b == 'y' || b == 'Y')
    {
        displayDevice(profile,num_profiles);
    }

    break;
}
case 3:
{
    char device_Name[100];
    int new_priority;
    printf("Enter the name and the new priority of the device: ");
    fflush(stdin);
    fgets(device_Name,sizeof(device_Name),stdin);

    if (  strlen(device_Name)>0 && device_Name[strlen(device_Name) - 1] == '\n') {
        device_Name[strlen(device_Name) - 1] = '\0';
    }
    int space=21-strlen(device_Name);
    for(int k=0;k<space;k++)
    {
        strcat(device_Name," ");
    }

    scanf("%d",&new_priority);
    edit_device(profile,num_profiles,device_Name,new_priority);

    char b;
    printf("\nWould you like to see the updated list(y/n): ");
    scanf(" %c", &b);
    if (b == 'n' || b == 'N')
```

```
                {
                    break;
                }
                else if (b == 'y' || b == 'Y')
                {
                    displayDevice(profile,num_profiles);
                }

                break;
            }
            default:
                printf("Please enter a valid option\n");
                break;
        }
        continue;
    } else {
        continue;
    }
}

int total_bandwidth=1000;

allocateBandwidth(profile, num_profiles, total_bandwidth);

printf("Allocating Bandwidth. Please Stand By.");
for (int i = 0; i < 6; ++i) {
    printf(".");
    fflush(stdout);
    Sleep(250);
}

printf("\n ---------------------------------------------------------------------------\n");
printf("|                                                           |\n");
printf("|The System Router allocates a total of %d mbs to every device per cycle.|\n",total_bandwidth);
printf("|                                                           |\n");
printf(" ---------------------------------------------------------------------------\n");

displayDeviceWithBandwidth(profile,num_profiles);
printf("\nThank you");
freeProfiles(profile, num_profiles);
return 0;
}
```

Pre-defined Devices:

Priority	Device Name	Ip Address

1	**Ab1-Host1**	**192.168.1.20**
2	**Ab2-Host2**	**192.168.1.40**
3	**Ab2-Host3**	**192.168.1.60**
4	**Ab2-Host1**	**192.168.1.80**
5	**MG-Host1**	**192.168.1.30**
6	**MG-Host2**	**192.168.1.50**
7	**Ab3-Host1**	**192.168.1.10**
8	**Delta_Block-Host1**	**192.168.1.56**
9	**Admin_Block-Host2**	**192.168.1.70**
10	**Nethaji-Host1**	**192.168.1.90**

Result:

Menu:

```
"C:\Users\zayan\OneDrive\Desktop\OS project\main.exe"
Device List:
Device name      Device Priority         Ip Address
Ab1-Host1               1               192.168.1.20
Ab2-Host2               2               192.168.1.40
Ab2-Host3               3               192.168.1.60
Ab2-Host1               4               192.168.1.80
MG-Host1                5               192.168.1.30
MG-Host2                6               192.168.1.50
Ab3-Host1               7               192.168.1.10
Delta Block-Host1       8               192.168.1.56
Admin Block-Host2       9               192.168.1.70
Nethaji-Host1           10              192.168.1.90
MENU:
Press the required option
1.Add a new device
2.Remove a device
3.Edit priority order
Would you like to continue?(y/n):
```

Pre-existing List of Devices:

```
Allocating Bandwidth. Please Stand By..........
 ------------------------------------------------------------------------
|                                                                        |
|The System Router allocates a total of 1000 mbs to every device per cycle.|
|                                                                        |
 ------------------------------------------------------------------------
Device List with Bandwidth:

+------------------------------------------------------------------------+
|                                                                        |
|Device name     Device Priority         Allocated Bandwidth     Ip Address   |
|Ab1-Host1               1                       181 mb          192.168.1.20 |
|Ab2-Host2               2                       163 mb          192.168.1.40 |
|Ab2-Host3               3                       145 mb          192.168.1.60 |
|Ab2-Host1               4                       127 mb          192.168.1.80 |
|MG-Host1                5                       109 mb          192.168.1.30 |
|MG-Host2                6                       90 mb           192.168.1.50 |
|Ab3-Host1               7                       72 mb           192.168.1.10 |
|Delta Block-Host1       8                       54 mb           192.168.1.56 |
|Admin Block-Host2       9                       36 mb           192.168.1.70 |
|Nethaji-Host1           10                      18 mb           192.168.1.90 |
|                                                                        |
+------------------------------------------------------------------------+

Thank you
```

Adding New Devices:

```
Device List with Bandwidth:

+-------------------------------------------------------------------------+
|                                                                         |
|Device name     Device Priority         Allocated Bandwidth     Ip Address   |
|Ab1-Host1              1                       153 mb          192.168.1.20 |
|Ab2-Host2              2                       141 mb          192.168.1.40 |
|Ab2-Host3              3                       128 mb          192.168.1.60 |
|Ab2-Host1              4                       115 mb          192.168.1.80 |
|MG-Host1               5                       102 mb          192.168.1.30 |
|MG-Host2               6                       89 mb           192.168.1.50 |
|Ab3-Host1              7                       76 mb           192.168.1.10 |
|Delta Block-Host1      8                       64 mb           192.168.1.56 |
|Admin Block-Host2      9                       51 mb           192.168.1.70 |
|Nethaji-Host1          10                      38 mb           192.168.1.90 |
|Libaray Servers        11                      25 mb           192.168.1.55 |
|Server Room            12                      12 mb           192.168.1.60 |
|                                                                         |
+-------------------------------------------------------------------------+
```

Removing a Device:

```
 -------------------------------------------------------------------------
|                                                                         |
|The System Router allocates a total of 1000 mbs to every device per cycle.|
|                                                                         |
 -------------------------------------------------------------------------
Device List with Bandwidth:

+-------------------------------------------------------------------------+
|                                                                         |
|Device name     Device Priority         Allocated Bandwidth     Ip Address   |
|Ab1-Host1              1                       222 mb          192.168.1.20 |
|Ab2-Host2              2                       194 mb          192.168.1.40 |
|Ab2-Host1              3                       166 mb          192.168.1.80 |
|MG-Host1               4                       138 mb          192.168.1.30 |
|Ab3-Host1              5                       111 mb          192.168.1.10 |
|Delta Block-Host1      6                       83 mb           192.168.1.56 |
|Admin Block-Host2      7                       55 mb           192.168.1.70 |
|Nethaji-Host1          8                       27 mb           192.168.1.90 |
|                                                                         |
+-------------------------------------------------------------------------+

Thank you
```

Editing the Priority Order of the Devices:

```
|                                                                              |
|The System Router allocates a total of 1000 mbs to every device per cycle.|
|                                                                              |
 ------------------------------------------------------------------------------
Device List with Bandwidth:

+------------------------------------------------------------------------------+
|                                                                              |
|Device name     Device Priority         Allocated Bandwidth     Ip Address    |
|Admin Block-Host2      1                       181 mb          192.168.1.70 |
|Ab1-Host1              2                       163 mb          192.168.1.20 |
|Ab2-Host2              3                       145 mb          192.168.1.40 |
|Ab2-Host3              4                       127 mb          192.168.1.60 |
|Ab2-Host1              5                       109 mb          192.168.1.80 |
|MG-Host1               6                       90 mb           192.168.1.30 |
|MG-Host2               7                       72 mb           192.168.1.50 |
|Ab3-Host1              8                       54 mb           192.168.1.10 |
|Delta Block-Host1      9                       36 mb           192.168.1.56 |
|Nethaji-Host1          10                      18 mb           192.168.1.90 |
|                                                                              |
+------------------------------------------------------------------------------+

Thank you
```

Result Mapping:

Mapping the results of the Wi-Fi Priority Manager project involves evaluating its outcomes in comparison to the initially identified problem statement and existing systems. The project aimed to address the challenge of optimizing Wi-Fi network connectivity by implementing a Priority Round Robin algorithm for bandwidth allocation based on user-defined priority levels. The results can be mapped to the problem statement through assessing the efficiency of the implemented solution in achieving fair distribution of bandwidth while prioritizing higher-priority networks, thereby enhancing overall network performance.

In comparison to existing systems, which may lack user-friendly interfaces and advanced prioritization mechanisms, the Wi-Fi Priority Manager introduces a novel approach by combining C programming's system-level capabilities with an intuitive interface. The mapping of results involves evaluating the success of this integration in providing users with greater control over their Wi-Fi connections. Furthermore, the proposed alterations and altered components, such as flexible priority management and machine learning-based optimization, contribute to the

project's adaptability and potential future advancements, aligning with the project's goal of enhancing connectivity experiences within the operating system environment. Overall, the mapping of results serves to validate the effectiveness of the implemented solution in addressing the identified problem and highlights the project's contributions and improvements over existing systems.

CHAPTER VI
CONCLUSION AND FUTURE WORKS

In conclusion, the Wi-Fi Priority Manager project successfully addresses the critical challenge of optimizing Wi-Fi network connectivity by introducing a Priority Round Robin algorithm for bandwidth allocation. The implemented solution, grounded in C programming's system-level capabilities, provides users with a user-friendly interface, empowering them to manage Wi-Fi priorities effectively. The system's core functionalities, such as fair bandwidth distribution and dynamic prioritization, align closely with the project's objectives, resulting in a reliable and efficient network management tool. The incorporation of proposed alterations, such as flexible priority management and machine learning-based optimization, positions the project at the forefront of adaptability, paving the way for future advancements in network resource allocation within the operating system environment.

Looking ahead, there is ample opportunity for further improvement and expansion of the Wi-Fi Priority Manager. Future works could focus on refining and

optimizing the sorting strategies, exploring more advanced algorithms to enhance efficiency, especially as the number of devices scales. Additionally, the integration of real-time adaptive features, responsive to evolving network conditions, and predictive machine learning models could further elevate the system's performance. Moreover, incorporating user feedback for additional features and customization options would contribute to a more tailored user experience. By continually refining the project through such future works, the Wi-Fi Priority Manager can evolve into a cutting-edge solution that not only meets but anticipates the dynamic connectivity needs of users in an ever-changing technological landscape.

CHAPTER VII
REFERENCES

1.Schulz-Zander, J., Mayer, C., Ciobotaru, B., Schmid, S., Feldmann, A., & Riggio, R. (2015). Programming the home and enterprise WiFi with OpenSDWN. *ACM SIGCOMM Computer Communication Review*, *45*(4), 117-118.

2.Mahindra, R., Viswanathan, H., Sundaresan, K., Arslan, M. Y., & Rangarajan, S. (2014, September). A practical traffic management system for integrated LTE-WiFi networks. In *Proceedings of the 20th annual international conference on Mobile computing and networking* (pp. 189-200).

3.Leng, Q., Chen, W. J., Huang, P. C., Wei, Y. H., Mok, A. K., & Han, S. (2019). Network management of multicluster RT-WiFi networks. *ACM Transactions on Sensor Networks (TOSN)*, *15*(1), 1-26.

4.Raz, Danny, and Yuval Shavitt. "An active network approach to efficient network management." *Active Networks: First International Working Conference, IWAN'99, Berlin, Germany, June 30-July 2, 1999. Proceedings 1*. Springer Berlin Heidelberg, 1999.

5.Liu, Chi Harold, et al. "Efficient network management for context-aware participatory sensing." *2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. IEEE, 2011.

6.Papavassiliou, Symeon, et al. "Mobile agent-based approach for efficient network management and resource allocation: framework and applications." *IEEE Journal on Selected Areas in Communications* 20.4 (2002): 858-872.

7.Bashar, Abul, et al. "Employing Bayesian belief networks for energy efficient network management." *2010 National Conference On Communications (NCC)*. IEEE, 2010.

8.Nellore, Kapileswar, and Gerhard P. Hancke. "A survey on urban traffic management system using wireless sensor networks." *Sensors* 16.2 (2016): 157.

9.Adame, Toni, Marc Carrascosa-Zamacois, and Boris Bellalta. "Time-sensitive networking in IEEE 802.11 be: On the way to low-latency WiFi 7." *Sensors* 21.15 (2021): 4954.

10.Ruiz, Linnyer Beatrys, Jose Marcos Nogueira, and Antonio AF Loureiro. "Manna: A management architecture for wireless sensor networks." *IEEE communications Magazine* 41.2 (2003): 116-125.