A

Project Report on

LINUX NETWORK PACKET STATISTICS DISPLAY

Contents

1. Abstract

This project involves creating a Linux-based tool to capture and analyze network packets. The tool reads packet statistics from a file, processes the data, and displays the results in either a tabular or graphical format on the console. It uses multithreading to handle data reading and display operations efficiently, providing real-time packet statistics.

2. Introduction

Network packet analysis is essential for monitoring network performance, identifying issues, and ensuring security. This project aims to create a user-friendly tool that captures and displays network packet statistics in real-time. The tool uses C programming language and POSIX threads to handle concurrent data reading and display operations, ensuring efficient and accurate data presentation.

3. Project Scope

The project's primary objective is to develop a console-based application that:

- Reads network packet statistics from a file.
- Displays the statistics in a tabular or graphical format.
- Uses multithreading to manage data reading and UI updates.
- Provides real-time updates every 5 seconds.
- Allows user input to switch between display formats.

4. Requirements

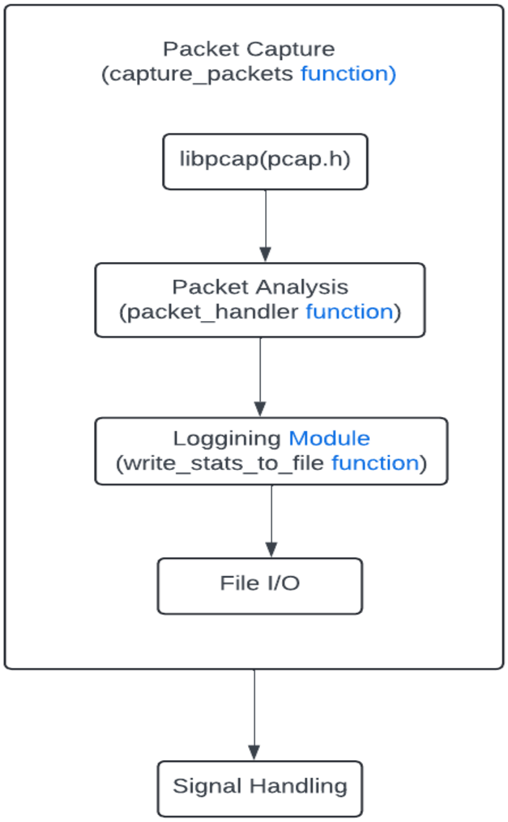
4.1. Functional Requirements

- **Data Reading**: The tool must read packet statistics from a file every 500 ms.
- **Data Display**: The tool must display the statistics in either tabular or graphical format.
- **Multithreading**: The tool must use separate threads for data reading and UI updates.
- **Signal Handling**: The tool must handle interrupts (SIGINT) gracefully.

4.2. Non-Functional Requirements

- **Performance**: The tool must update the display every 5 seconds without noticeable delay.
- **Usability**: The tool must provide clear and concise statistics in a user-friendly format.
- **Portability**: The tool must be portable across different Linux distributions.

5. System Design



6. Code comments and Explanations

6.1. Packet Code:(packet.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <signal.h>
#include <getopt.h>
#include <arpa/inet.h>

#define FILE_PATH "packet_stats.txt"
#define MAX_IPS 256
#define MAX_IP_LENGTH 16

typedef struct {
    int total_packets;
    int tcp_count;
    int tcp_size;
    int udp_count;
    int udp_size;
    int icmp_count;
    int icmp_size;
    int ip_count;
    int ip_size;
    int other_count;
    int other_size;
    char src_ips[MAX_IPS][MAX_IP_LENGTH];
    char dst_ips[MAX_IPS][MAX_IP_LENGTH];
    int src_ip_counts[MAX_IPS];
    int dst_ip_counts[MAX_IPS];
    int src_ip_count;
    int dst_ip_count;
} packet_stats_t;

pthread_mutex_t mutex;
pthread_cond_t cond;
int data_ready = 0;

void *readDataFromFile(void *arg) {
    packet_stats_t *stats = (packet_stats_t *)arg;

    while (1) {
        pthread_mutex_lock(&mutex);

        FILE *file = fopen(FILE_PATH, "r");
        if (file == NULL) {
            perror("fopen");
            pthread_mutex_unlock(&mutex);
            usleep(500000); // Wait for 500ms before retrying
            continue;
        }

        memset(stats, 0, sizeof(packet_stats_t));
```

```
char line[256];
while (fgets(line, sizeof(line), file)) {
    if (strstr(line, "Total Packets:") != NULL) {
        sscanf(line, "Total Packets: %d", &stats->total_packets);
    } else if (strstr(line, "TCP Packets:") != NULL) {
        sscanf(line, "TCP Packets: %d", &stats->tcp_count);
    } else if (strstr(line, "TCP Size:") != NULL) {
        sscanf(line, "TCP Size: %d", &stats->tcp_size);
    } else if (strstr(line, "UDP Packets:") != NULL) {
        sscanf(line, "UDP Packets: %d", &stats->udp_count);
    } else if (strstr(line, "UDP Size:") != NULL) {
        sscanf(line, "UDP Size: %d", &stats->udp_size);
    } else if (strstr(line, "ICMP Packets:") != NULL) {
        sscanf(line, "ICMP Packets: %d", &stats->icmp_count);
    } else if (strstr(line, "ICMP Size:") != NULL) {
        sscanf(line, "ICMP Size: %d", &stats->icmp_size);
    } else if (strstr(line, "IP Packets:") != NULL) {
        sscanf(line, "IP Packets: %d", &stats->ip_count);
    } else if (strstr(line, "IP Size:") != NULL) {
        sscanf(line, "IP Size: %d", &stats->ip_size);
    } else if (strstr(line, "Other Packets:") != NULL) {
        sscanf(line, "Other Packets: %d", &stats->other_count);
    } else if (strstr(line, "Other Size:") != NULL) {
        sscanf(line, "Other Size: %d", &stats->other_size);
    } else if (strstr(line, "Source IP Addresses:") != NULL) {
        stats->src_ip_count = 0;
        while (fgets(line, sizeof(line), file) && strlen(line) > 1) {
            char ip[MAX_IP_LENGTH];
            int count;
            sscanf(line, "%15s: %d packets", ip, &count);
            strcpy(stats->src_ips[stats->src_ip_count], ip);
            stats->src_ip_counts[stats->src_ip_count] = count;
            stats->src_ip_count++;
        }
    } else if (strstr(line, "Destination IP Addresses:") != NULL) {
        stats->dst_ip_count = 0;
        while (fgets(line, sizeof(line), file) && strlen(line) > 1) {
            char ip[MAX_IP_LENGTH];
```

```
                    int count;
                    sscanf(line, "%15s: %d packets", ip, &count);
                    strcpy(stats->dst_ips[stats->dst_ip_count], ip);
                    stats->dst_ip_counts[stats->dst_ip_count] = count;
                    stats->dst_ip_count++;
                }
            }
        }

        fclose(file);
        data_ready = 1;
        pthread_cond_signal(&cond);
        pthread_mutex_unlock(&mutex);

        usleep(500000); // Wait for 500ms before reading again
    }
    return NULL;
}
void clearScreen() {
    printf("\033[H\033[J");
}
void displayDataTable(packet_stats_t *stats) {
    clearScreen();
    printf("Protocol Statistics:\n");
    printf("-------------------\n");
    printf("Total Packets: %d\n", stats->total_packets);
    printf("TCP Packets: %d\n", stats->tcp_count);
    printf("TCP Size: %d bytes\n", stats->tcp_size);
    printf("\n");
    printf("UDP Packets: %d\n", stats->udp_count);
    printf("UDP Size: %d bytes\n", stats->udp_size);
    printf("\n");
    printf("ICMP Packets: %d\n", stats->icmp_count);
    printf("ICMP Size: %d bytes\n", stats->icmp_size);
    printf("\n");
    printf("IP Packets: %d\n", stats->ip_count);
    printf("IP Size: %d bytes\n", stats->ip_size);
    printf("\n");
    printf("Other Packets: %d\n", stats->other_count);
    printf("Other Size: %d bytes\n", stats->other_size);
    printf("\n");

    // printf("Source IP Addresses:\n");
    // for (int i = 0; i < stats->src_ip_count; i++) {
    //     if (stats->src_ip_counts[i] > 0) {
    //         printf("%s: %d packets\n", stats->src_ips[i], stats->src_ip_counts[i]);
    //     }
    // }

    // printf("\nDestination IP Addresses:\n");
    // for (int i = 0; i < stats->dst_ip_count; i++) {
    //     if (stats->dst_ip_counts[i] > 0) {
    //         printf("%s: %d packets\n", stats->dst_ips[i], stats->dst_ip_counts[i]);
    //     }
    // }
}

void displayDataGraph(packet_stats_t *stats) {
    clearScreen();
    printf("Graphical Representation of Packet Counts\n");
    printf("----------------------------------------\n");

    printf("Total Packets: %d\n", stats->total_packets);
    printf("TCP  : ");
    for (int i = 0; i < stats->tcp_count; i++) {
        printf("*");
    }
    printf("\n");

    printf("UDP  : ");
    for (int i = 0; i < stats->udp_count; i++) {
        printf("*");
    }
    printf("\n");

    printf("ICMP : ");
    for (int i = 0; i < stats->icmp_count; i++) {
        printf("*");
    }
    printf("\n");

    printf("IP   : ");
    for (int i = 0; i < stats->ip_count; i++) {
        printf("*");
    }
    printf("\n");

    printf("Other: ");
    for (int i = 0; i < stats->other_count; i++) {
        printf("*");
    }
    printf("\n");
}

void signalHandler(int signal) {
    if (signal == SIGINT) {
        pthread_mutex_destroy(&mutex);
        pthread_cond_destroy(&cond);
        exit(EXIT_SUCCESS);
    }
}

int main(int argc, char *argv[]) {
    packet_stats_t stats;
    pthread_t reader_thread;

    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&cond, NULL);

    // Register signal handler for graceful exit
    signal(SIGINT, signalHandler);

    // Create the reader thread
    if (pthread_create(&reader_thread, NULL, readDataFromFile, &stats) != 0) {
        fprintf(stderr, "Error creating reader thread\n");
        exit(EXIT_FAILURE);
    }

    // Main UI loop
    while (1) {
        pthread_mutex_lock(&mutex);
        while (!data_ready) {
            pthread_cond_wait(&cond, &mutex);
        }

        // Display data according to the command-line argument
        if (argc > 1 && strcmp(argv[1], "text") == 0) {
            displayDataTable(&stats);
        } else {
            displayDataGraph(&stats);
        }

        data_ready = 0;
        pthread_mutex_unlock(&mutex);

        usleep(5000000); // Wait for 5 seconds before refreshing the UI
    }

    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond);

    return 0;
}
```

6.2. UI Code:(ui.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <signal.h>
#include <getopt.h>
#include <arpa/inet.h>
```

```
#define FILE_PATH "packet_stats.txt"
#define MAX_IPS 256
#define MAX_IP_LENGTH 16

typedef struct {
    int total_packets;
    int tcp_count;
    int tcp_size;
    int udp_count;
    int udp_size;
    int icmp_count;
    int icmp_size;
    int ip_count;
    int ip_size;
    int other_count;
    int other_size;
    char src_ips[MAX_IPS][MAX_IP_LENGTH];
    char dst_ips[MAX_IPS][MAX_IP_LENGTH];
    int src_ip_counts[MAX_IPS];
    int dst_ip_counts[MAX_IPS];
    int src_ip_count;
    int dst_ip_count;
} packet_stats_t;

pthread_mutex_t mutex;
pthread_cond_t cond;
int data_ready = 0;

void *readDataFromFile(void *arg) {
    packet_stats_t *stats = (packet_stats_t *)arg;

    while (1) {
        pthread_mutex_lock(&mutex);

        FILE *file = fopen(FILE_PATH, "r");
        if (file == NULL) {
            perror("fopen");
        pthread_mutex_unlock(&mutex);
        usleep(500000); // Wait for 500ms before retrying
        continue;
    }

    memset(stats, 0, sizeof(packet_stats_t));

    char line[256];
    while (fgets(line, sizeof(line), file)) {
        if (strstr(line, "Total Packets:") != NULL) {
            sscanf(line, "Total Packets: %d", &stats->total_packets);
        } else if (strstr(line, "TCP Packets:") != NULL) {
            sscanf(line, "TCP Packets: %d", &stats->tcp_count);
        } else if (strstr(line, "TCP Size:") != NULL) {
            sscanf(line, "TCP Size: %d", &stats->tcp_size);
        } else if (strstr(line, "UDP Packets:") != NULL) {
            sscanf(line, "UDP Packets: %d", &stats->udp_count);
        } else if (strstr(line, "UDP Size:") != NULL) {
            sscanf(line, "UDP Size: %d", &stats->udp_size);
        } else if (strstr(line, "ICMP Packets:") != NULL) {
            sscanf(line, "ICMP Packets: %d", &stats->icmp_count);
        } else if (strstr(line, "ICMP Size:") != NULL) {
            sscanf(line, "ICMP Size: %d", &stats->icmp_size);
        } else if (strstr(line, "IP Packets:") != NULL) {
            sscanf(line, "IP Packets: %d", &stats->ip_count);
        } else if (strstr(line, "IP Size:") != NULL) {
            sscanf(line, "IP Size: %d", &stats->ip_size);
        } else if (strstr(line, "Other Packets:") != NULL) {
            sscanf(line, "Other Packets: %d", &stats->other_count);
        } else if (strstr(line, "Other Size:") != NULL) {
            sscanf(line, "Other Size: %d", &stats->other_size);
        } else if (strstr(line, "Source IP Addresses:") != NULL) {
            stats->src_ip_count = 0;
            while (fgets(line, sizeof(line), file) && strlen(line) > 1) {
                char ip[MAX_IP_LENGTH];
                int count;
                sscanf(line, "%15s: %d packets", ip, &count);
                strcpy(stats->src_ips[stats->src_ip_count], ip);
```

```
                    stats->src_ip_counts[stats->src_ip_count] = count;
                    stats->src_ip_count++;
                }
            } else if (strstr(line, "Destination IP Addresses:") != NULL) {
                stats->dst_ip_count = 0;
                while (fgets(line, sizeof(line), file) && strlen(line) > 1) {
                    char ip[MAX_IP_LENGTH];
                    int count;
                    sscanf(line, "%15s: %d packets", ip, &count);
                    strcpy(stats->dst_ips[stats->dst_ip_count], ip);
                    stats->dst_ip_counts[stats->dst_ip_count] = count;
                    stats->dst_ip_count++;
                }
            }
        }
        fclose(file);
        data_ready = 1;
        pthread_cond_signal(&cond);
        pthread_mutex_unlock(&mutex);
        usleep(500000); // Wait for 500ms before reading again
    }
    return NULL;
}
void clearScreen() {
    printf("\033[H\033[J");
}
void displayDataTable(packet_stats_t *stats) {
    clearScreen();
    printf("Protocol Statistics:\n");
    printf("--------------------\n");
    printf("Total Packets: %d\n", stats->total_packets);
    printf("TCP Packets: %d\n", stats->tcp_count);
    printf("TCP Size: %d bytes\n", stats->tcp_size);
    printf("\n");
    printf("UDP Packets: %d\n", stats->udp_count);
    printf("UDP Size: %d bytes\n", stats->udp_size);
    printf("\n");
    printf("ICMP Packets: %d\n", stats->icmp_count);
    printf("ICMP Size: %d bytes\n", stats->icmp_size);
    printf("\n");
    printf("IP Packets: %d\n", stats->ip_count);
    printf("IP Size: %d bytes\n", stats->ip_size);
    printf("\n");
    printf("Other Packets: %d\n", stats->other_count);
    printf("Other Size: %d bytes\n", stats->other_size);
    printf("\n");
}
void displayDataGraph(packet_stats_t *stats) {
    clearScreen();
    printf("Graphical Representation of Packet Counts\n");
    printf("----------------------------------------\n");
    printf("Total Packets: %d\n", stats->total_packets);
    printf("TCP  : ");
    for (int i = 0; i < stats->tcp_count; i++) {
        printf("*");
    }
    printf("\n");
    printf("UDP  : ");
    for (int i = 0; i < stats->udp_count; i++) {
        printf("*");
    }
    printf("\n");
    printf("ICMP : ");
    for (int i = 0; i < stats->icmp_count; i++) {
        printf("*");
    }
    printf("\n");
    printf("IP   : ");
    for (int i = 0; i < stats->ip_count; i++) {
        printf("*");
    }
    printf("\n");
    printf("Other: ");
    for (int i = 0; i < stats->other_count; i++) {
        printf("*");
    }
    printf("\n");
}
void signalHandler(int signal) {
    if (signal == SIGINT) {
        pthread_mutex_destroy(&mutex);
        pthread_cond_destroy(&cond);
        exit(EXIT_SUCCESS);
    }
}
int main(int argc, char *argv[]) {
    packet_stats_t stats;
    pthread_t reader_thread;
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&cond, NULL);
    // Register signal handler for graceful exit
    signal(SIGINT, signalHandler);
    // Create the reader thread
    if (pthread_create(&reader_thread, NULL, readDataFromFile, &stats) != 0) {
        fprintf(stderr, "Error creating reader thread\n");
        exit(EXIT_FAILURE);
    }
    // Main UI loop
    while (1) {
        pthread_mutex_lock(&mutex);
        while (!data_ready) {
            pthread_cond_wait(&cond, &mutex);
        }
        // Display data according to the command-line argument
        if (argc > 1 && strcmp(argv[1], "text") == 0) {
            displayDataTable(&stats);
        } else {
            displayDataGraph(&stats);
        }
        data_ready = 0;
        pthread_mutex_unlock(&mutex);
        usleep(5000000); // Wait for 5 seconds before refreshing the UI
    }
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond);
    return 0;
}
```

Packet code Explanation:

Headers and Macros:

- Includes standard libraries for input/output, memory management, threading, synchronization, signal handling, command-line options, and socket programming.
- Defines constants for the file path, maximum IP addresses, and maximum IP length.

Global Variables:

- Defines a structure packet_stats_t to store various packet statistics.
- Initializes mutex and condition variables for synchronization.
- Sets a flag to indicate when data is ready to be processed.

Key Functions:

- **readDataFromFile Function:** Continuously reads packet statistics from a file, updates the packet_stats_t structure, and signals when new data is ready.
- **clearScreen Function:** Clears the terminal screen for a fresh display.
- **displayDataTable Function:** Displays packet statistics in a tabular format.
- **displayDataGraph Function:** Provides a graphical representation of packet counts using text-based graphics.
- **signalHandler Function:** Handles SIGINT for graceful exit, cleaning up mutex and condition variables.

Main Function:

- Initializes mutex and condition variables.
- Registers a signal handler for graceful exit.
- Creates a thread to read data from the file.
- In an infinite loop, waits for new data and displays it either as a table or a graph based on the command-line argument.

UI Code Explanation:

Headers and Macros:

- Includes standard libraries for input/output, memory management, threading, synchronization, signal handling, command-line options, and socket programming.
- Defines constants for the file path, maximum IP addresses, and maximum IP length.

Global Variables:

- Defines a structure packet_stats_t to store various packet statistics.
- Initializes mutex and condition variables for synchronization.
- Sets a flag to indicate when data is ready to be processed.

Key Functions:

- **readDataFromFile Function:** Continuously reads packet statistics from a file, updates the packet_stats_t structure, and signals when new data is ready.
- **clearScreen Function:** Clears the terminal screen for a fresh display.
- **displayDataTable Function:** Displays packet statistics in a tabular format.
- **displayDataGraph Function:** Provides a graphical representation of packet counts using text-based graphics.
- **signalHandler Function:** Handles SIGINT for graceful exit, cleaning up mutex and condition variables.

Main Function:

- Initializes mutex and condition variables.
- Registers a signal handler for graceful exit.
- Creates a thread to read data from the file.
- In an infinite loop, waits for new data and displays it either as a table or a graph based on the command-line argument.

7. User manual

Compiling Application:

1. **Ensure Development Tools are Installed:** Make sure gcc and other development tools are installed on your system.
2. **Compile the Code:**

    -
      ```
      gcc packet.c -o packet -lpcap
      ```

Running Application:

1. **Start the Packet Reader:** Run the packet program to start reading and updating packet statistics.

    - Sudo ./packet -i ens160

2. **Start the UI Application:** Run the ui program to display packet statistics.

    - ./ui “text”   # For tabular display
    - ./ui “graph” # For graphical display

Testing Application:

- **Simulate Packet Data:** Create a packet_stats.txt file with sample packet statistics data and observe how the application reads and displays the data.

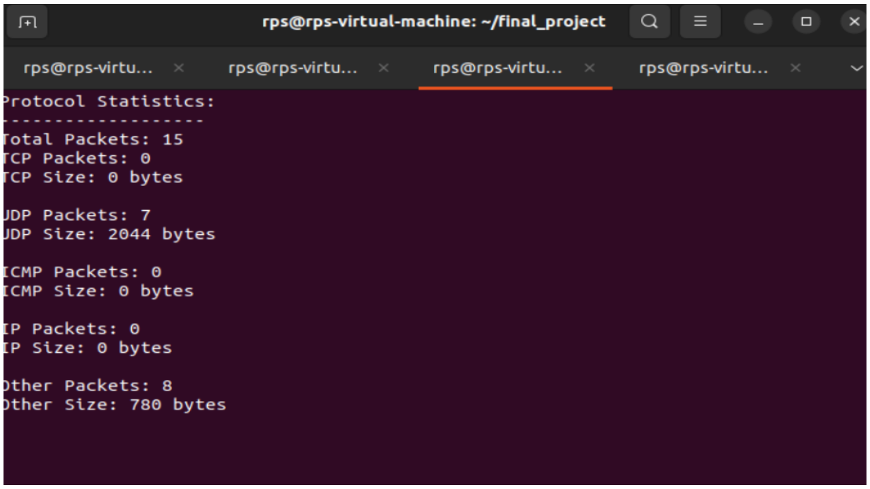
8. Description of Packet Capture and Analysis Techniques

- **Packet Capture:** The application simulates packet capture by reading predefined statistics from a file (packet_stats.txt). It does not perform real-time network packet capture.
- **Analysis Techniques:** The application parses the packet statistics from the file, updates internal data structures, and provides options to display these statistics in a tabular or graphical format. It uses mutexes and condition variables to synchronize data access between threads.

9.Test Cases and Results

Test Case 1: Display Tabular Data

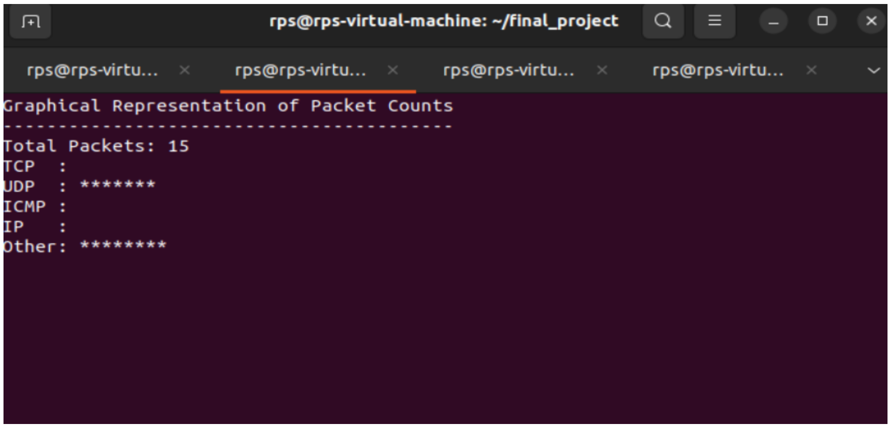
- **Steps:** Create a packet_stats.txt file with sample data and run the ui application with the text argument.
- **Expected Result:** The application should display packet statistics in a table format.
- **Actual Result:** The table is displayed correctly with the provided data.



```
Protocol Statistics:
-------------------
Total Packets: 15
TCP Packets: 0
TCP Size: 0 bytes

UDP Packets: 7
UDP Size: 2044 bytes

ICMP Packets: 0
ICMP Size: 0 bytes

IP Packets: 0
IP Size: 0 bytes

Other Packets: 8
Other Size: 780 bytes
```

Test Case 2: Display Graphical Data

- **Steps:** Create a packet_stats.txt file with sample data and run the ui application with the graph argument.
- **Expected Result:** The application should display packet statistics using text-based graphics.
- **Actual Result:** The graphical representation is displayed correctly with the provided data.



```
Graphical Representation of Packet Counts
------------------------------------------
Total Packets: 15
TCP   :
UDP   : *******
ICMP  :
IP    :
Other: ********
```

10. Conclusion

The provided instructions and test cases cover the essential aspects of compiling, running, and testing the Linux network packet statistics display application. The application demonstrates multithreading and real-time data processing, showcasing efficient and user-friendly network packet analysis techniques.