Project Report

On

NETWORK BANDWIDTH MONITORING TOOL

Submitted in partial fulfilment of the requirements for the award of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

(Artificial Intelligence & Machine Learning)

by

Ms. M PRASANNA (22WH1A6615)

Ms. S DEEPIKA PRAHARSHINI (22WH1A6623)

Ms. B HEMANYA SAI (22WH1A6636)

Ms. P JAHNAVI (22WH1A6639)

Under the esteemed guidance of

Ms. P Anusha

Assistant Professor, CSE(AI&ML)

BVRIT HYDERABAD College of Engineering for Women

(UGC Autonomous Institution | Approved by AICTE | Affiliated to JNTUH)

(NAAC Accredited - A Grade | NBA Accredited B.Tech. (EEE, ECE, CSE and IT)

Bachupally, Hyderabad – 500090

2024-25

ABSTRACT

The network bandwidth monitor tool is a Python-based application that tracks real-time upload and download speeds as well as total network usage. This tool can be run in either a GUI-based environment, utilizing the tkinter library, or in a console-based headless mode, making it versatile for different system setups. The program uses the psutil library to fetch network statistics and continuously updates the network information in a user-friendly format. It is designed to be efficient, with updates occurring periodically, and provides network usage data in human-readable formats such as bytes, KB, MB, GB, and TB. This application helps users monitor their network activity in real-time, making it useful for both casual users and system administrators.

PROBLEM STATEMENT

In today's digital world, network connectivity plays a vital role in everyday computing tasks, whether for personal use, business operations, or system administration. Monitoring network bandwidth, including upload and download speeds, is essential for diagnosing network issues, optimizing resource allocation, and ensuring the smooth operation of network-dependent applications. However, most existing network monitoring tools either provide limited functionality, are overly complex, or are not easily customizable for different user needs and environments.

Users often face challenges in tracking their real-time network activity effectively. For example:

- Users may struggle to understand raw network data, such as the number of bytes transferred, without conversion to more familiar units (e.g., KB, MB, GB).
- In a headless environment (such as remote servers or systems without a graphical user interface), existing tools might not provide a user-friendly way to view live network statistics.
- Many existing network monitoring solutions may be too resource-intensive or overly complicated for basic usage, making them unsuitable for casual users or low-resource environments.

The need for a lightweight, real-time network bandwidth monitor that can function across different environments is evident. This tool must be able to display live upload/download speeds and total network usage in a clear, easy-to-understand format. It should offer flexibility in its operation, running as a GUI-based tool on systems with display support, or falling back to a console-based output in headless environments, making it versatile for various user scenarios.

This application aims to fill that gap by providing an intuitive, lightweight, and efficient solution for monitoring network activity. The key objective is to enable users to effortlessly track network performance in real time, whether they are using it on a local machine with a GUI or remotely on servers without a GUI, ensuring a smooth experience across platforms and environments.

FUNCTIONAL REQUIREMENTS

1. **Network Statistics Monitoring:**
    - The application must fetch real-time network statistics, including bytes sent (upload) and bytes received (download), using the psutil library.
    - It should calculate and display upload and download speeds, as well as total network usage.
2. **Human-readable Output:**
    - The application must convert the network statistics into a human-readable format (KB, MB, GB, TB).
    - Both the raw values (in bytes) and the converted values should be displayed to the user.
3. **Real-time Updates:**
    - The network statistics should be updated at regular intervals (default of 1500ms).
    - The display should update continuously, either in the GUI or console, to reflect the latest network statistics.
4. **User Interface:**
    - The application should provide a GUI using tkinter for environments that support it, displaying the upload, download speeds, and total usage.
    - In headless environments, the application should display the information in a console with periodic updates.
5. **Wraparound Handling:**
    - The application must handle network statistics wraparound (when the counter resets) to avoid errors in calculating the upload and download speeds.
6. **Cross-Environment Support:**
    - The application should function in both GUI environments and headless (command-line) environments without any issues.

NON-FUNCTIONAL REQUIREMENTS

1. **Performance:**
    - The application should not introduce significant delays or system overhead. The update interval for the statistics should be efficient and lightweight, ensuring minimal CPU and memory usage.
    - The system should update the statistics regularly with a delay of 1-2 seconds for the console and 1.5 seconds for the GUI.
2. **Usability:**
    - The GUI version should be simple, with a clear and intuitive layout that allows users to easily monitor the network statistics.
    - The console version should be readable, with clear formatting and separation between different data points.
3. **Reliability:**
    - The application must be reliable, ensuring that network statistics are accurately calculated and displayed in both GUI and console modes.
    - The application should handle errors gracefully, especially in situations such as wraparounds of network counters.
4. **Portability:**
    - The application should work on multiple platforms (e.g., Windows, Linux, macOS) without requiring extensive modifications.
    - It should use Python libraries (psutil and tkinter) that are widely available or easy to install.
5. **Scalability:**
    - While the application is designed for individual users, it should be easily extensible to handle larger-scale deployments, such as server environments, where multiple instances or network interfaces might need to be monitored simultaneously.
6. **Maintainability:**

- The code should be well-documented with clear comments and a consistent structure, making it easy to maintain, debug, and extend.
- It should use modular functions to separate concerns (e.g., network fetching, UI updates, and output formatting).

7. **Security:**
    - While this tool does not directly interact with sensitive data, the application should ensure that it does not leak or misuse any system or network data while fetching and displaying network statistics.

SOURCE CODE

```
import tkinter as tk
from psutil import net_io_counters
import time
import os


# Variables for use in the size() function.
KB = float(1024)
MB = float(KB ** 2)  # 1,048,576
GB = float(KB ** 3)  # 1,073,741,824
TB = float(KB ** 4)  # 1,099,511,627,776


def size(B):
    """Converts bytes to a human-readable format (KB, MB, GB, etc.)"""
    B = float(B)
    if B < KB:
        return f"{B:.2f} Bytes"
    elif KB <= B < MB:
        return f"{B/KB:.2f} KB"
    elif MB <= B < GB:
        return f"{B/MB:.2f} MB"
    elif GB <= B < TB:
        return f"{B/GB:.2f} GB"
    elif TB <= B:
```

```
        return f"{B/TB:.2f} TB"


# Constants
WINDOW_SIZE = (400, 400)  # Width x Height for GUI
WINDOW_RESIZEABLE = False  # If the window is resizeable or not.
REFRESH_DELAY = 1500  # Delay for window update in ms.


# Variables to hold the last known values of upload/download.
last_upload, last_download, upload_speed, down_speed = 0, 0, 0, 0


# Check if display environment variable is set (GUI can only work if display exists)
is_headless = not os.environ.get('DISPLAY')


# GUI-based application (for environments with a GUI)
def create_gui():
    window = tk.Tk()
    window.title("Network Bandwidth Monitor")  # Setting the window title.
    window.geometry(f"{WINDOW_SIZE[0]}x{WINDOW_SIZE[1]}")  # Setting the
window size.
    window.resizable(width=WINDOW_RESIZEABLE, height=WINDOW_RESIZEABLE)
# Locking the window.


    # Labels for the window
    label_total_upload_header = tk.Label(text="Total Upload:", font="Quicksand 12 bold")
    label_total_upload_header.pack()
```

```
    label_total_upload = tk.Label(text="Calculating...", font="Quicksand 12")
    label_total_upload.pack()

    label_total_download_header = tk.Label(text="Total Download:", font="Quicksand 12
bold")
    label_total_download_header.pack()
    label_total_download = tk.Label(text="Calculating...", font="Quicksand 12")
    label_total_download.pack()

    label_total_usage_header = tk.Label(text="Total Usage:", font="Quicksand 12 bold")
    label_total_usage_header.pack()
    label_total_usage = tk.Label(text="Calculating...\n", font="Quicksand 12")
    label_total_usage.pack()

    label_upload_header = tk.Label(text="Upload:", font="Quicksand 12 bold")
    label_upload_header.pack()
    label_upload = tk.Label(text="Calculating...", font="Quicksand 12")
    label_upload.pack()

    label_download_header = tk.Label(text="Download:", font="Quicksand 12 bold")
    label_download_header.pack()
    label_download = tk.Label(text="Calculating...", font="Quicksand 12")
    label_download.pack()

    attribution = tk.Label(text="\n~ WaterrMalann ~", font="Quicksand 11 italic")
```

```
attribution.pack()


# Updating Labels with network statistics
def update():
    global last_upload, last_download, upload_speed, down_speed
    counter = net_io_counters()

    upload = counter.bytes_sent
    download = counter.bytes_recv
    total = upload + download

    # Calculate upload speed
    if last_upload > 0:
        if upload < last_upload:  # Wraparound case
            upload_speed = 0
        else:
            upload_speed = upload - last_upload

    # Calculate download speed
    if last_download > 0:
        if download < last_download:  # Wraparound case
            down_speed = 0
        else:
            down_speed = download - last_download
```

```
        # Update the last known values
        last_upload = upload
        last_download = download

        # Update the labels with the formatted data
        label_total_upload.config(text=f"{size(upload)} ({upload} Bytes)")
        label_total_download.config(text=f"{size(download)} ({download} Bytes)")
        label_total_usage.config(text=f"{size(total)}\n")
        label_upload.config(text=f"{size(upload_speed)}")
        label_download.config(text=f"{size(down_speed)}")

        # Schedule the next update
        window.after(REFRESH_DELAY, update)

    # Start the update process
    window.after(REFRESH_DELAY, update)

    # Start the GUI loop
    window.mainloop()

# Console-based application (for headless environments)
def display_console():
    global last_upload, last_download, upload_speed, down_speed
    while True:
```

```
counter = net_io_counters()

upload = counter.bytes_sent
download = counter.bytes_recv
total = upload + download

# Calculate upload speed
if last_upload > 0:
    if upload < last_upload:  # Wraparound case
        upload_speed = 0
    else:
        upload_speed = upload - last_upload

# Calculate download speed
if last_download > 0:
    if download < last_download:  # Wraparound case
        down_speed = 0
    else:
        down_speed = download - last_download

# Update the last known values
last_upload = upload
last_download = download
```

```
        # Print the updated information to the console
        print(f"Total Upload: {size(upload)} ({upload} Bytes)")
        print(f"Total Download: {size(download)} ({download} Bytes)")
        print(f"Total Usage: {size(total)}")
        print(f"Upload Speed: {size(upload_speed)}")
        print(f"Download Speed: {size(down_speed)}")
        print('-' * 40)

        # Wait for a short period before the next update
        time.sleep(1)
# Main function to check environment and run accordingly
if is_headless:
    print("Running in headless mode, using console output...")
    display_console()  # Console-based display for headless environments
else:
    print("Running with GUI...")
    create_gui()  # GUI-based display
```

OUTPUT

```
Running in headless mode, using console output...
Total Upload: 3.02 MB (3166869 Bytes)
Total Download: 3.07 MB (3215374 Bytes)
Total Usage: 6.09 MB
Upload Speed: 0.00 Bytes
Download Speed: 0.00 Bytes
----------------------------------------
Total Upload: 3.04 MB (3188605 Bytes)
Total Download: 3.09 MB (3234916 Bytes)
Total Usage: 6.13 MB
Upload Speed: 21.23 KB
Download Speed: 19.08 KB
----------------------------------------
Total Upload: 3.06 MB (3205596 Bytes)
Total Download: 3.10 MB (3251441 Bytes)
Total Usage: 6.16 MB
Upload Speed: 16.59 KB
Download Speed: 16.14 KB
----------------------------------------
Total Upload: 3.09 MB (3236443 Bytes)
Total Download: 3.13 MB (3282033 Bytes)
Total Usage: 6.22 MB
Upload Speed: 30.12 KB
Download Speed: 29.88 KB
----------------------------------------
Total Upload: 3.10 MB (3253818 Bytes)
Total Download: 3.15 MB (3299652 Bytes)
Total Usage: 6.25 MB
Upload Speed: 16.97 KB
Download Speed: 17.21 KB
----------------------------------------
```

```
----------------------------------------
Total Upload: 3.12 MB (3270482 Bytes)
Total Download: 3.16 MB (3316969 Bytes)
Total Usage: 6.28 MB
Upload Speed: 16.27 KB
Download Speed: 16.91 KB
----------------------------------------
Total Upload: 3.13 MB (3287146 Bytes)
Total Download: 3.18 MB (3333349 Bytes)
Total Usage: 6.31 MB
Upload Speed: 16.27 KB
Download Speed: 16.00 KB
----------------------------------------
Total Upload: 3.15 MB (3307968 Bytes)
Total Download: 3.20 MB (3354002 Bytes)
Total Usage: 6.35 MB
Upload Speed: 20.33 KB
Download Speed: 20.17 KB
----------------------------------------
Total Upload: 3.17 MB (3324566 Bytes)
Total Download: 3.21 MB (3369445 Bytes)
Total Usage: 6.38 MB
Upload Speed: 16.21 KB
Download Speed: 15.08 KB
----------------------------------------
Total Upload: 3.19 MB (3341296 Bytes)
Total Download: 3.23 MB (3386762 Bytes)
Total Usage: 6.42 MB
Upload Speed: 16.34 KB
Download Speed: 16.91 KB
----------------------------------------
```

```
----------------------------------------
Total Upload: 3.20 MB (3357960 Bytes)
Total Download: 3.25 MB (3403142 Bytes)
Total Usage: 6.45 MB
Upload Speed: 16.27 KB
Download Speed: 16.00 KB
----------------------------------------
Total Upload: 3.22 MB (3374624 Bytes)
Total Download: 3.26 MB (3419522 Bytes)
Total Usage: 6.48 MB
Upload Speed: 16.27 KB
Download Speed: 16.00 KB
----------------------------------------
Total Upload: 3.23 MB (3391288 Bytes)
Total Download: 3.28 MB (3435902 Bytes)
Total Usage: 6.51 MB
Upload Speed: 16.27 KB
Download Speed: 16.00 KB
----------------------------------------
Total Upload: 3.25 MB (3407952 Bytes)
Total Download: 3.29 MB (3452282 Bytes)
Total Usage: 6.54 MB
Upload Speed: 16.27 KB
Download Speed: 16.00 KB
----------------------------------------
Total Upload: 3.27 MB (3424682 Bytes)
Total Download: 3.31 MB (3468728 Bytes)
Total Usage: 6.57 MB
Upload Speed: 16.34 KB
Download Speed: 16.06 KB
----------------------------------------
```