# **Project Report**

**A Real-Time Network Traffic Monitoring System Using Python and WebSockets**

**Author:** Rishabh Gaurish Shetye  
**Github:** <https://github.com/Rishabh-G-Shetye/py-netmon>

## **Abstract**

This report details the design, implementation, and functionality of a real-time network traffic monitoring system. The project's primary objective is to capture network packets from a host machine, process them to extract key information, and display this data dynamically on a web-based dashboard.

The system architecture consists of three main components:

1. A Python-based packet sniffer utilizing the **Pyshark** library.
2. A backend web server built with **Flask** and **Flask-SocketIO**.
3. A frontend interface created using **HTML, CSS, and JavaScript**.

The sniffer captures live traffic, sends structured data to a REST API endpoint on the Flask server, which then pushes the data in real time to the web dashboard via WebSockets. The result is a lightweight, accessible tool for network monitoring that provides immediate visual feedback on network activity.

The report concludes by discussing system limitations and outlining future work, including the integration of machine learning for anomaly detection to evolve the system into a basic Intrusion Detection System (IDS).

## **1. Introduction**

### **1.1 Background**

Network traffic analysis is a fundamental practice in computer networking and cybersecurity. It involves the inspection of data packets traversing a network to troubleshoot performance issues, debug applications, and detect malicious activities.

While professional tools like **Wireshark** provide comprehensive analysis, they can be overly complex for lightweight monitoring. This project aims to build an accessible, high-level monitoring tool using Python for packet capture and modern web technologies for real-time visualization.

### **1.2 Project Objectives**

The project objectives are:

* Capture live network packets from a host machine.
* Extract essential metadata (IP addresses, ports, protocols, timestamps).
* Transmit processed data to a central server application.
* Visualize traffic dynamically on a real-time web dashboard.

### **1.3 Scope**

The system is designed for single-host monitoring, focusing on **IP, TCP, UDP, and ICMP** protocols. It is a **proof-of-concept**, not a production-grade monitoring solution.

## **2. System Architecture and Design**

The system uses a **three-component architecture**, ensuring modularity and separation of concerns. Components communicate via **HTTP** and **WebSockets**.

### **2.1 Component 1: The Packet Sniffer (sniffer.py)**

* **Technology:** Python, **pyshark** (wrapper for Tshark).
* **Functions:**
  + Detects active network interface (via netifaces).
  + Captures live packets.
  + Extracts fields (timestamp, IPs, ports, transport layer, highest layer).
  + Sends structured JSON data to the backend via HTTP POST.

### **2.2 Component 2: The Backend Server (server.py)**

* **Technology:** Flask + Flask-SocketIO.
* **Functions:**
  + Provides a REST API (/report) for incoming sniffer data.
  + Caches recent packets in memory for new clients.
  + Broadcasts data in real time via WebSockets.
  + Hosts the frontend dashboard (index.html).

### **2.3 Component 3: The Frontend Dashboard (index.html)**

* **Technology:** HTML, CSS, JavaScript, Socket.IO client.
* **Functions:**
  + Establishes a WebSocket connection with server.
  + Loads cached packet history on connection.
  + Dynamically updates a data table with live packets.
  + Provides status indicators and protocol color-coding.

## **3. Implementation Details**

### **3.1 Packet Capture and Filtering**

* Core capture via pyshark.LiveCapture.
* Requires **root/administrator privileges**.
* Filtering avoids capturing the sniffer’s own traffic to 127.0.0.1:5000.

### **3.2 Real-Time Communication Protocol**

* Choose **WebSockets** over HTTP polling for low-latency push communication.
* Used socketio.emit('new\_packet', data) to broadcast to all connected dashboards.

### **3.3 Frontend Interface**

* Table format with **protocol-based color coding**.
* Event-driven JavaScript updates table dynamically.
* Performance capped at **100 most recent entries**.

## **4. Results and Discussion**

The system achieved its objectives: capturing traffic, transmitting it, and displaying it in real time.

### **4.1 Limitations**

* **Privilege Requirement:** Requires root/admin rights.
* **Performance Bottleneck:** High traffic may overload Python or HTTP requests.
* **Data Persistence:** Logs are ephemeral and lost on restart.
* **Security:** Development-grade Flask server, no HTTPS, no authentication for WebSockets.

## **5. Conclusion and Future Work**

### **5.1 Conclusion**

The project demonstrates a functional real-time network monitoring tool that integrates **Pyshark, Flask, and WebSockets**. It is effective as an **educational tool** for visualizing network traffic and a foundation for advanced IDS applications.

### **5.2 Future Work**

Planned enhancements include:

* **Intrusion Detection Integration:**
  + Logging packets with feature extraction.
  + Training ML models (e.g., Random Forest, Gradient Boosting, Neural Networks).
  + Real-time classification of malicious traffic.
  + Dashboard alerts for anomalies.
* **Deployment and Scalability:**
  + Docker containerization.
  + Production-grade server (Gunicorn/uWSGI + Nginx).
  + Database integration (PostgreSQL/InfluxDB).
* **Enhanced Dashboard:**
  + Filters by IP or protocol.
  + Graphical statistics (protocol distribution, bandwidth usage).

