Network Traffic Analysis Visualization and Real-Time Monitoring Using Python

1. Introduction

Modern computer networks generate massive volumes of data every second. Analyzing this traffic is essential for understanding network behavior, optimizing performance, and detecting security threats. Network traffic analysis focuses on examining packet-level information such as protocols, IP addresses, timestamps, and packet sizes to extract meaningful insights.

This project implements a **complete network traffic analysis pipeline** using Python. It includes:

- Offline packet analysis using PCAP files
- Data extraction and storage in CSV format
- Protocol and bandwidth visualization
- Real-time network monitoring
- Rule-based suspicious activity detection (mini IDS)

The project bridges **networking fundamentals, data analysis, and cybersecurity concepts** using practical tools and real traffic data.

2. Tools and Technologies Used

Tool	Purpose
Wireshark	Capture live network traffic and generate PCAP files
PyShark	Parse PCAP files and capture live traffic programmatically

Python	Core programming language
Pandas	Data manipulation and analysis
Matplotlib	Visualization of protocols and bandwidth
OS / Threading / Asyncio	System-level handling and real-time processing

3. Project Architecture and Workflow

The project follows a modular and structured workflow:

1. Capture network traffic using Wireshark
2. Parse PCAP files using PyShark
3. Extract essential packet fields
4. Store processed data in CSV format
5. Perform protocol and bandwidth analysis
6. Visualize traffic patterns
7. Monitor traffic in real time
8. Detect suspicious behavior using predefined rules

This separation improves **readability, scalability, and maintainability**.

4. Traffic Capture and PCAP Processing

4.1 Packet Capture

Network packets are captured using Wireshark and saved as PCAP files. These files contain raw packet-level data, including timestamps, IP addresses, protocols, and packet lengths.

4.2 Reading PCAP Files Using PyShark

PyShark is used to parse PCAP files efficiently using lazy loading. To improve performance:

- Packet storage is disabled (`keep_packets=False`)
- A maximum packet limit is enforced

4.3 Extracted Packet Fields

The following fields are extracted from each packet:

- Time (`sniff_time`)
- Source IP
- Destination IP
- Protocol (`highest_layer`)
- Packet length (bytes)

These fields are sufficient for traffic pattern analysis, bandwidth calculation, and anomaly detection.

4.4 CSV Storage

Extracted packet data is stored in `traffic.csv`, enabling:

- Reusability without reprocessing PCAP files
- Faster analysis using Pandas
- Easy visualization and reporting

5. Offline Traffic Analysis

5.1 Protocol Distribution Analysis

Packet counts are grouped by protocol and visualized using bar charts.

Purpose:

- Identify dominant protocols (e.g., TCP, TLS)
- Detect unusual protocol usage
- Establish baseline network behavior

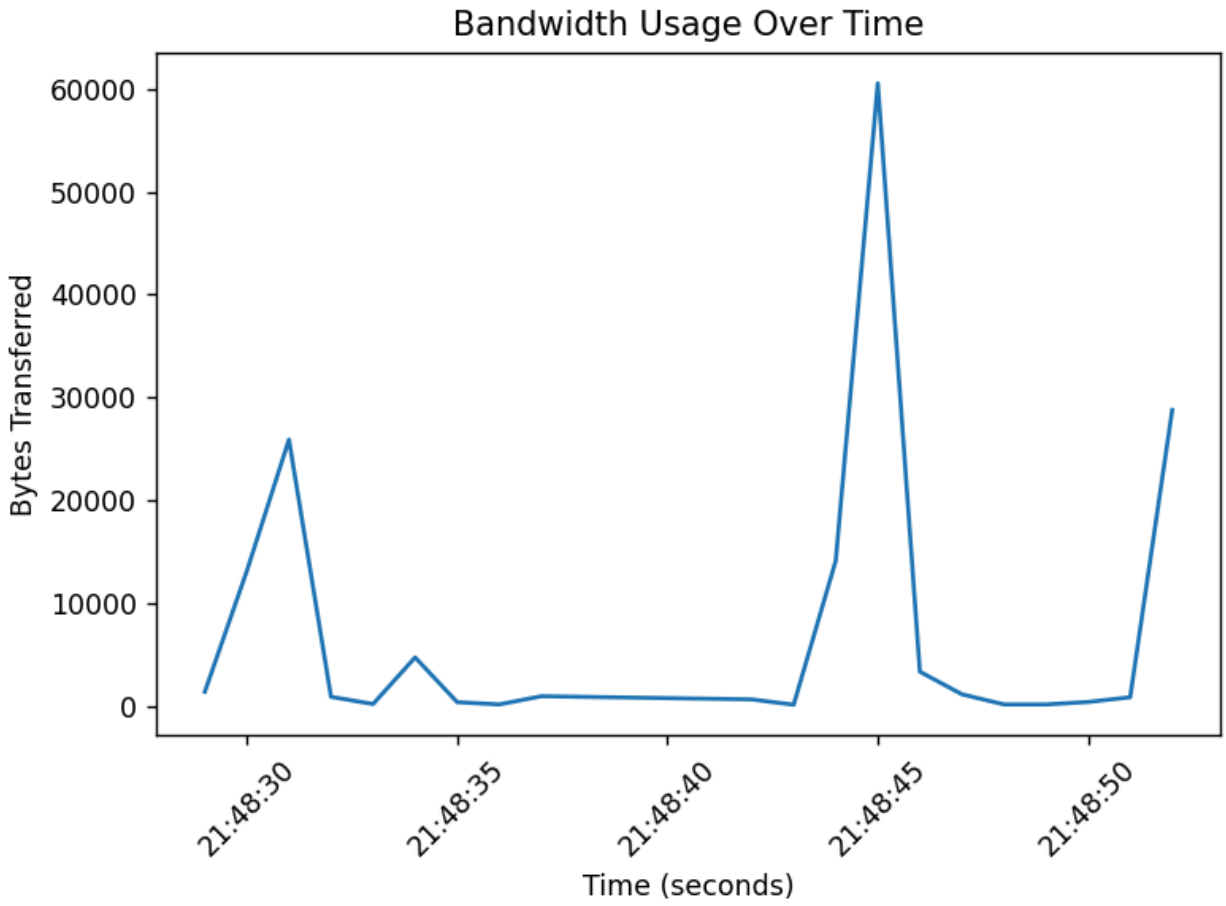
This analysis helps identify anomalies such as excessive ICMP or UDP traffic.

5.2 Bandwidth Usage Analysis

Bandwidth usage is calculated by:

- Converting packet timestamps to seconds
- Grouping packets per second
- Summing packet lengths

The resulting line graph shows **bytes transferred per second over time**.



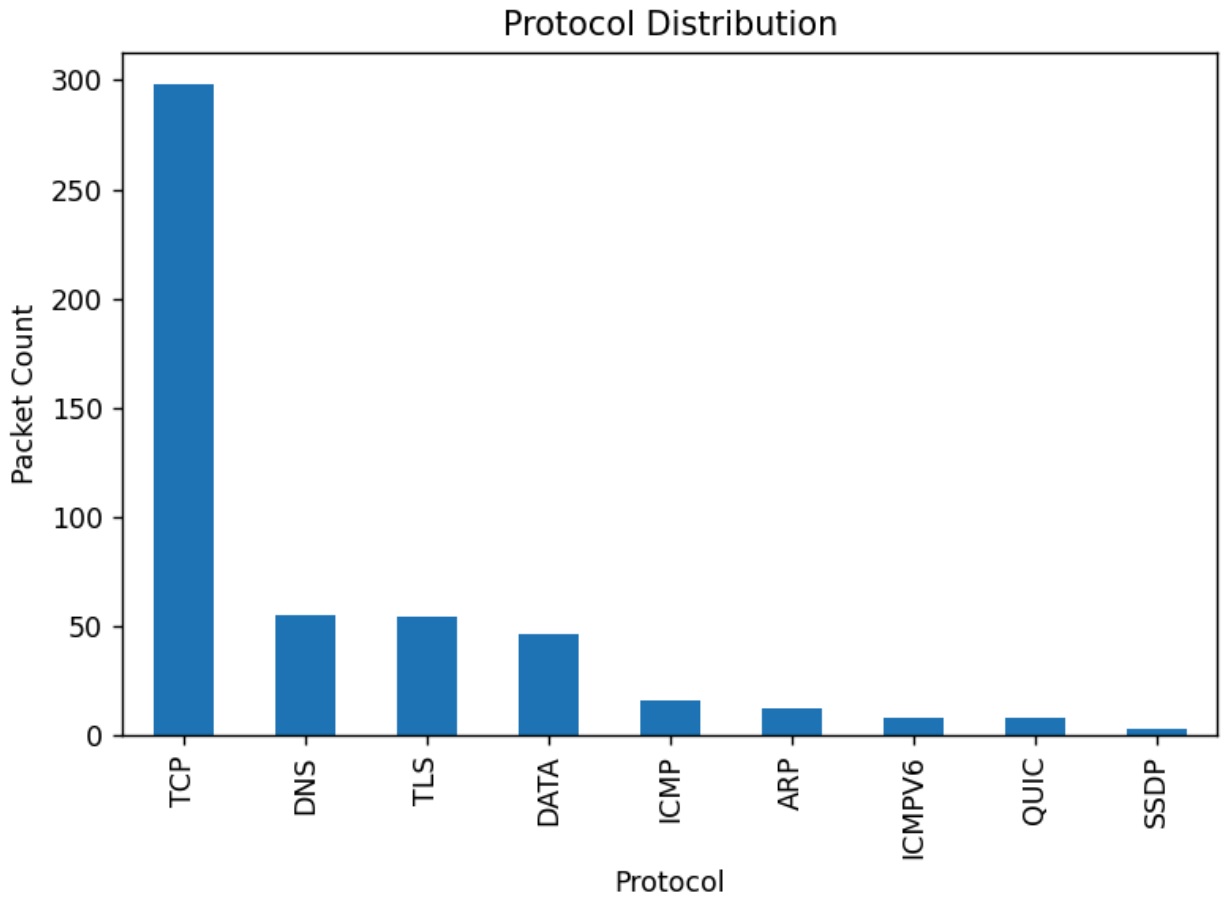
Insights:

- Detect traffic spikes
- Identify congestion or heavy transfers
- Highlight abnormal bursts of traffic

5.3 Protocol-Level Bandwidth Analysis

Bandwidth usage is also analyzed per protocol, allowing comparison of how different protocols consume network resources. This helps detect:

- Protocol abuse
- Flood-based attacks
- Data-heavy encrypted traffic patterns



6. Data Cleaning and Validation

To ensure accurate analysis, the following preprocessing steps are applied:

- Conversion of packet length to numeric values
- Timestamp parsing and validation
- Removal of invalid or missing values
- Filtering valid IPv4 and IPv6 addresses
- Restricting analysis to known protocols

These steps improve **data reliability and visualization clarity**.

7. Visualization Strategy

Multiple visualizations are generated in a **single compact figure** for professional presentation:

- Packet count per protocol
- Bandwidth usage per protocol
- Top source IPs for selected protocols
- Protocol-wise bandwidth over time
- Highlighting suspicious protocols using color coding

This approach allows quick comparison and efficient interpretation.

8. Real-Time Network Monitoring

8.1 Live Packet Capture

Real-time traffic is captured directly from the network interface using PyShark's `LiveCapture`.

Key features:

- Thread-based packet capture
- Manual asyncio event loop handling (Python 3.12 compatibility)
- Sliding window storage for recent packets only

This ensures smooth, continuous monitoring without memory overflow.

8.2 Live Visualization

Two real-time plots are continuously updated:

- Protocol distribution
- Bandwidth usage over time

Bandwidth spikes are highlighted using red markers for immediate visibility.

9. Suspicious Activity Detection (Mini IDS)

9.1 Detection Rules

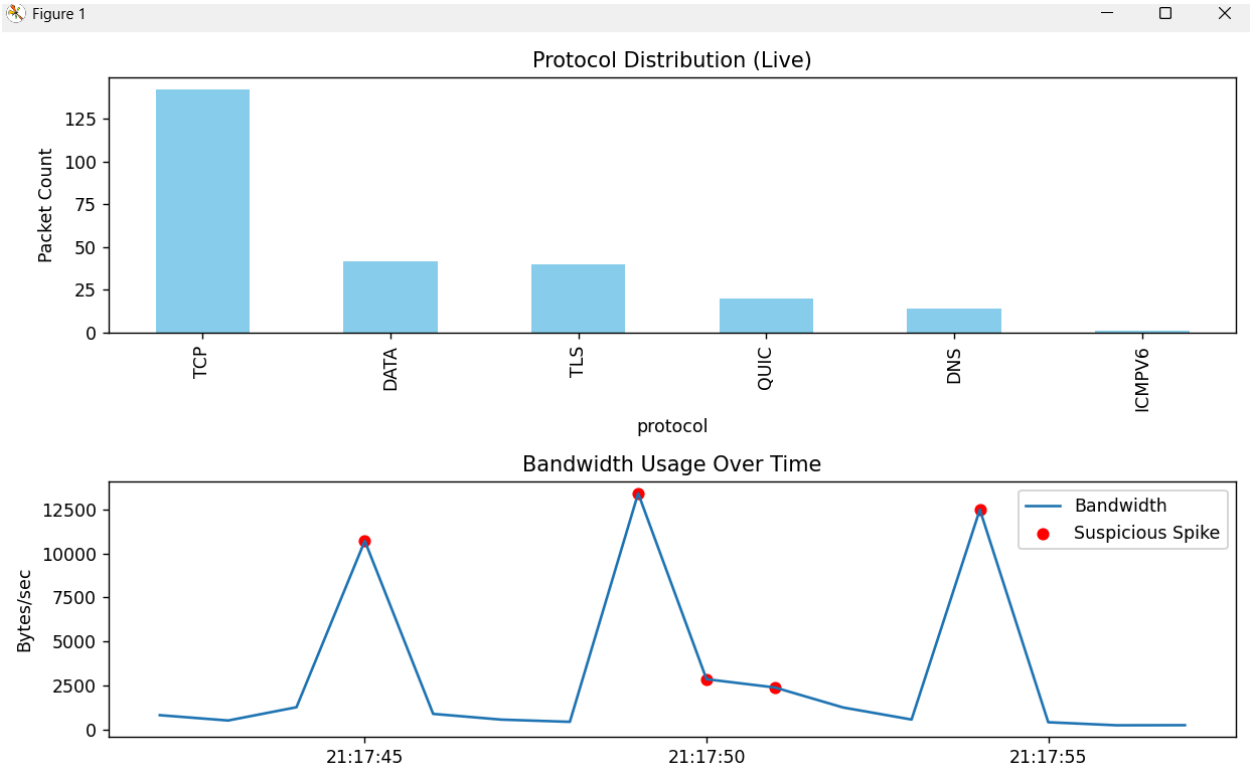
The system applies simple but effective rule-based detection:

Condition	Threshold
High packet rate from one IP	> 50 packets in 10 seconds
Bandwidth spike	> 2000 bytes/sec

9.2 Alerts and Visualization

- Suspicious IPs are printed as console alerts
- Bandwidth spikes are marked visually
- Abnormal protocol usage is highlighted in red

This mimics the core logic of real intrusion detection systems.



10. Protocol-Level Analysis

Traffic was analyzed separately for:

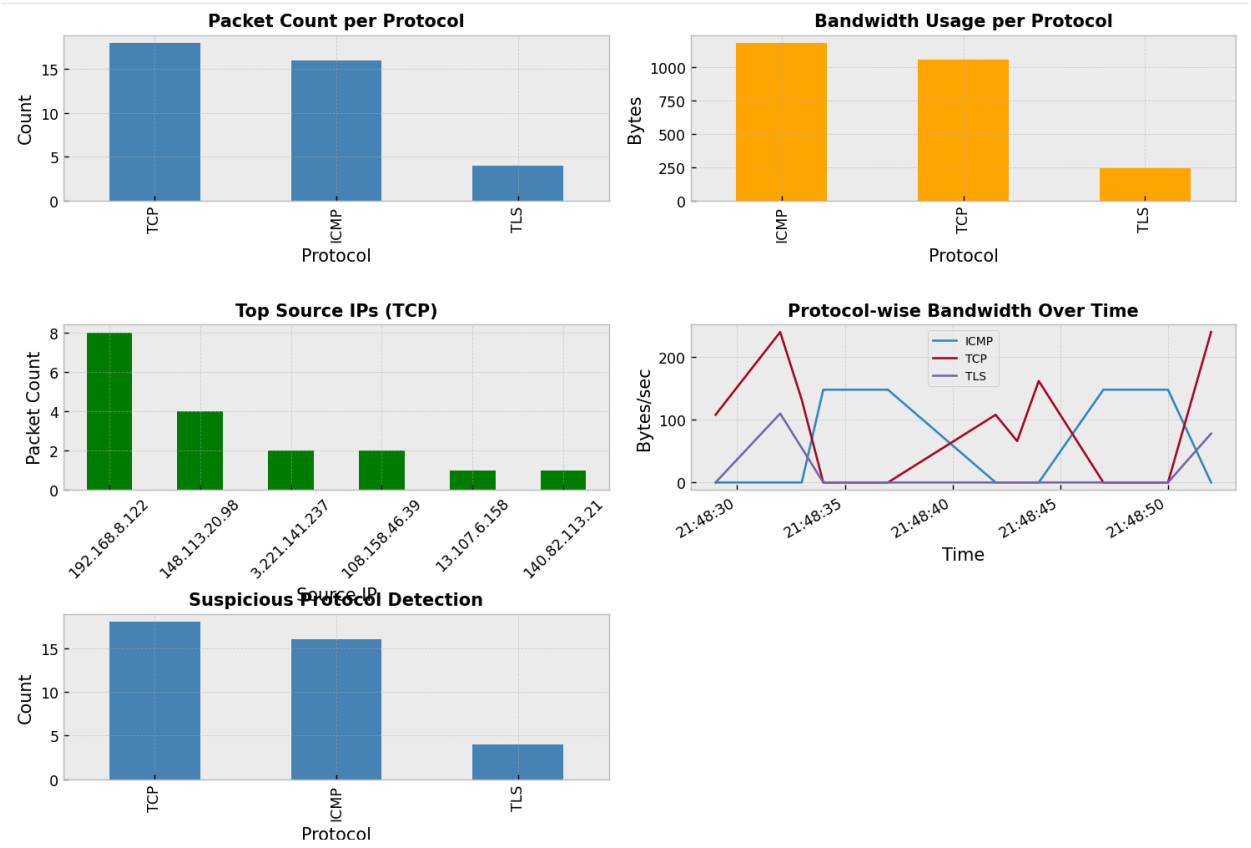
- TCP
- UDP
- TLS
- ICMP

Benefits

- Detect protocol misuse
- Identify flood attacks
- Understand protocol-specific behavior

Example:

- Excessive ICMP → possible ping flood
- Sudden UDP surge → possible UDP flood



11. Performance Optimization and Debugging

Several performance challenges were addressed:

- Limiting packet count in memory
- Processing packets on the fly
- Fixing asyncio conflicts in threaded environments

Resolving these issues demonstrates **strong debugging and system-level understanding**.

12. Conclusion

This project presents a **complete and practical network traffic analysis system**, integrating offline analysis, real-time monitoring, and basic intrusion detection. By combining Python, PyShark, Pandas, and Matplotlib, the system demonstrates how raw network data can be transformed into actionable security insights.

The project goes beyond basic visualization and reflects **intermediate to advanced-level understanding** of networking and cybersecurity concepts.