International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 05 | May 2026 www.irjet.net p-ISSN: 2395-0072

RAHU: The Personal Visual Firewall Log

K. Yogesh[1], M. Rithwik[2], K. Abhivamsh[3], P. Mukesh ,Ms. S. Premalatha[4]

[1-4] Department of Computer Science and Engineering, Keshav Memorial Institute of Technology, Hyderabad, India

Under the guidance of Assistant Professor, Department of Computer Science and Engineering, Keshav Memorial Institute of Technology, Hyderabad, India

---

Abstract - *In an era of growing digital privacy concerns, everyday users often lack intuitive tools to monitor and control their computer's network activity. Rahu is a desktop application designed as a user-friendly, host-based application firewall that provides real-time visibility into all outgoing network connections. By mapping each connection to its originating process using OS-level monitoring and packet sniffing techniques, Rahu allows users to identify which applications are communicating externally. The system enriches this information with destination geolocation data via a local GeoIP database, presenting an interactive and continuously updating log through an Electron.js interface. Users can instantly block suspicious or unrecognized connections through seamless integration with the native operating system firewall. Developed with a Python-based backend and a cross-platform graphical interface, Rahu bridges the gap between technical network analysis and user accessibility, enhancing personal cyber security awareness and empowering users with meaningful control over their digital footprint.*

Key Words: Digital Privacy, Host-based Application Firewall, Network Activity Monitoring, Packet Sniffing, GeoIP Visualization, Real-time Log, Process Mapping, OS Firewall Integration, Personal Cyber security, Electron.js

1. INTRODUCTION

Modern computing environments are characterized by applications that continuously communicate with external servers to perform updates, telemetry, analytics, and background synchronization. These communications occur silently and without user awareness or consent. As digital privacy concerns grow, users increasingly demand tools that clearly reveal which applications are making network connections, where those connections are directed, and provide a straightforward mechanism to block suspicious activity.
Traditional firewalls focus on ports and IP addresses, offering minimal clarity regarding application level behavior. Packet analysis tools such as Wireshark require advanced networking expertise. Most existing host-based application firewalls are proprietary, resource-intensive, or depend on cloud analytics that compromise the very privacy they aim to protect.

Rahu addresses this need by acting as a personal visual firewall that merges real-time monitoring, geographic visualization, and one-click blocking into a single accessible interface, bridging the gap between complex technical tools and non-technical end users.

1.1 Problem Statement

Existing firewalls and monitoring tools suffer from several limitations. Most do not provide application-level visibility, making it difficult to determine which process initiated a connection. Packet analysis tools such as Wireshark generate highly technical packet dumps that require expert knowledge and do not map packets to their originating process. Many systems also separate monitoring and firewall control, forcing users to rely on multiple interfaces. Traditional firewalls lack one-click blocking and require manual rule creation. Modern applications frequently make silent outbound connections, yet existing systems do not highlight or alert users about these hidden communications. These shortcomings create a strong need for a simple, unified, non-technical solution.

1.2 Objectives

The major objectives of the proposed system are:

- To develop a real-time personal visual firewall for monitoring outgoing connections
- To map each network connection accurately to its originating application or process
- To provide GeoIP-based visualization of destination servers on an interactive map
- To enable instant one-click application blocking using native OS firewall commands
- To preserve user privacy through local-only processing and encrypted log storage

2. RELATED WORK

Several systems related to network monitoring, application firewalls, and cyber security tooling have been developed and studied. Traditional firewalls such as Windows Firewall, UFW, IPTables, and PF operate on rule-based filtering using ports, protocols, and IP addresses. While effective at blocking unwanted traffic, they do not indicate which application initiated a connection and do not offer real-time process mapping or destination visibility [2].

Packet-level tools such as Wireshark and Netstat provide detailed inspection of network data but suffer from several drawbacks including highly technical interfaces, no application-to-connection mapping, and lack of instant blocking capabilities. These tools are unsuitable for everyday users who simply want to see which applications are communicating externally [1].

Host-based application firewalls such as Little Snitch and GlassWire are closer to the concept of Rahu but still present limitations. Many are commercial products with paid subscriptions, some rely on cloud-based analytics that reduce user privacy, and most do not provide a unified interface combining monitoring, visualization, and one-click blocking.

The analysis of existing systems clearly reveals the absence of a unified, free, and locally operating platform capable of combining real-time process-to-connection mapping, GeoIP visualization, one-click blocking, and non-technical accessibility into a single environment. Rahu addresses this gap directly.

Table -1: Comparison of Existing Systems

System	Process Map	One-Click Block	Unified UI	Local Only
Windows Firewall	No	No	No	Yes
Wireshark	No	No	No	Yes
GlassWire	Partial	Pro Only	Partial	No
Little Snitch	Yes	Yes	Partial	No
Rahu	**Yes**	**Yes**	**Yes**	**Yes**

3. PROPOSED SYSTEM

Rahu is designed as a real-time personal visual firewall that provides complete outgoing network visibility and control through a unified desktop application. The platform provides users with a live connection dashboard Where all outgoing network activity can be monitored directly with process-level detail.

Users can view active connections along with the originating application name, executable path, destination IP, port, protocol, and geographic location. The system performs GeoIP lookups using a local MaxMind GeoLite2 database to present clear visual information about destination servers on an interactive world map powered by Leaflet.js and Open Street Map.

A major feature of the system is the one-click blocking capability, which allows users to instantly block any suspicious or unwanted application from making outgoing connections. The blocking is enforced through native OS firewall commands (netsh ad firewall on Windows, iptables on Linux, pfctl on macOS), requiring no manual firewall configuration from the user.

The system also includes an Alert Engine that evaluates each connection using heuristic scoring rules and assigns a risk score between 0.0 and 1.0. Connections are flagged for unusual destination ports, high connection volumes, unrecognized process names, watchlisted destination countries, and night-time transmissions by non-system processes.

To ensure privacy, all log data is stored locally in an encrypted SQLite database using Fernet symmetric encryption (AES-128-CBC with HMAC-SHA256). No data is ever transmitted to any remote server or cloud service. The system operates entirely on the local device and supports cross-platform deployment across Windows, Linux, and macOS.

4. METHODOLOGY

The methodology of Rahu consists of multiple stages including packet capture, process mapping, GeoIP resolution, risk evaluation, visualization, firewall control, and secure logging. The system is designed using a modular architecture where each component performs a specific task while maintaining efficient real-time communication with other modules.

4.1 Monitoring Engine and Packet Capture

The process begins when the Monitoring Engine class starts packet interception. The engine calls scapy. sniff() with a BPF filter restricting capture to TCP and UDP transport-layer traffic. The sniff function runs on a dedicated daemonthread, passing each

captured packet to the handle_packet() handler where the destination IP and port are extracted. On Windows, the Npcap driver is used; on Linux and macOS, libpcap operates natively. The engine verifies that each captured packet is outgoing by comparing the source IP against local inter face addresses obtainedfrom psutil. net_if_addrs() [4][5].

4.2 Process-to-Connection Mapping

Once a packet is captured, Rahu identifies the originating application by querying the OS socket table using psutil.net_connections(). For each active socket entry, the tuple of (local_address, local_port, remote_address, remote_port) is matched against the captured packet metadata.

When a match is found, the Process ID (PID) is retrieved and used to extract the process name and executable path. An in-memory lookup dictionary caches recent mappings with a 5-second expiry to minimize repeated queries.

4.3 GeoIP Resolution and Visualization

Every connection's remote IP address is resolved to a geographic location using the MaxMind GeoLite2 binary database [3] stored locally on the user's device. The GeoIPResolver class exposes a resolve(ip) method that returns the country name, ISO country code, city name, latitude, and longitude. Private-range IPs are returned as 'Local Network' without a database query.

On the frontend, resolved coordinates are plotted as color-coded circular markers on a Leaflet.js world map: green for low risk (score < 0.3), amber for medium (0.3–0.7), and red for high risk (> 0.7).

4.4 Alert Engine and Risk Scoring

The AlertEngine evaluates each new Connection Record against configurable heuristic rules and assigns a normalized risk score between 0.0 and 1.0. The default rule
set covers: uncommon destination port (+0.25), high-volume sender with more than 50 connections in 60 seconds (+0.30), unknown or unrecognized process (+0.40), destination country on watchlist (+0.35), and night-time transmission by a non-system process (+0.15). Alerts are stored in the encrypted IndexedDB and surfaced to the user through the dashboard panel.

4.5 Firewall Controller

The FirewallController is implemented as an abstract interface with platform-specific concrete implementations. When a user clicks Block, the controller executes native OS firewall commands as subprocess calls with shell=False to prevent injection attacks. The controller checks the return code of each command and raises a Firewall Error exception if the operation fails, which is surfaced to the UI as an error notification. Block rule records are written to the encrypted local database after successful enforcement.

4.6 Secure Storage and Cross-Platform Packaging

All connection records, block rules, and alert histories are stored in a local SQLite database. All data is encrypted using the Python cryptography library's Fernet symmetric encryption scheme with AES-128-CBC and HMAC-SHA256. The encryption key is derived from OS login credentials using PBKDF2-HMAC-SHA256 with 390,000 iterations and a device-specific salt stored in the OS keychain. The Electron UI is packaged using electron-builder and the Python backend is frozen using PyInstaller, removing any system Python dependency.

5. SYSTEM ARCHITECTURE

The architecture of Rahu is designed using a modular four-layer structure consisting of the Monitoring and Packet Capture Layer, Backend Processing Layer, Visualization and User Interface Layer, and Firewall Control Layer. The system ensures efficient real-time communication between packet capture engines, geospatial services, alert modules, and the OS firewall while maintaining local-only operation.

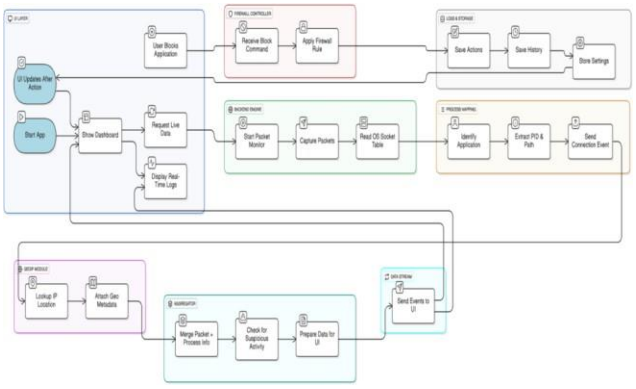

Fig -1 : Architecture workflow of Rahu: The personal Visual Firewall Log

The Monitoring and Packet Capture Layer gathers outgoing packets using Scapy/libpcap/Npcap and maps them to their originating process using psutil. The Backend Processing Layer, built with Python and FastAPI, handles connection correlation, GeoIP lookup, event evaluation, and secure SQLite logging. Connection events are streamed to the frontend in real

time using a Server-Sent Events (SSE) endpoint.

The Visualization Layer, built with Electron.js and Leaflet.js, displays live activity on an interactive dashboard and world map, allowing user interaction for blocking decisions. The Firewall Control Layer integrates directly with OS-level firewall systems (netsh, iptables, pfctl) to apply or remove block rules immediately upon user request.

6. RESULTS AND DISCUSSION

The proposed Rahu platform was tested under multiple network monitoring and firewall control scenarios to evaluate its performance, accuracy, responsiveness, and usability. The system was evaluated on a Windows 11 host machine (quad-core, 16 GB RAM), an Ubuntu 22.04 virtual machine, and a macOS Ventura system using a suite of 12 real-world applications spanning browsers, media players, development tools, and background services.

The monitoring engine accurately detected all outgoing TCP and UDP connections in real time. Each connection was correctly mapped to its originating application with the process name and executable path displayed in the live dashboard. The GeoIP resolution module correctly identified destination countries for all tested public IP addresses and returned 'Local Network' for private-range addresses as expected.

The Safety Heatmap equivalent — the GeoIP world map — refreshed dynamically as new connection events arrived through the SSE stream, providing a continuously evolving picture of outbound traffic destinations without requiring manual page reloads.

Administrative functionalities such as complaint verification, issue-status updates, and report moderation were tested successfully through the admin dashboard. Administrators could efficiently categorize complaints into different stages and monitor resolution workflows.

The one-click blocking feature successfully prevented targeted applications from making further outgoing connections. Firewall rules were confirmed using native OS commands and persisted across application restarts. The alert engine correctly generated HIGH severity alerts for processes making more than 50 connections within 60 seconds.

Encrypted log verification confirmed that raw SQLite database content was unreadable without the correct Fernet key, and all 50 stored records were recovered intact and correctly decrypted through the GET/history endpoint.

Table -2: System Performance Evaluation Results

	SRS Target	Achieved (Avg)
Detection latency	< 1 s	320 ms (Win) / 290 ms (Linux)
UI update latency (SSE)	< 500 ms	380 ms (Win) / 340 ms (Linux)
GeoIP lookup time	< 300 ms	42 ms (Win) / 38 ms (Linux)
Firewall block apply	< 2 s	0.8 s (Win) / 0.6 s (Linux)
CPU usage (active)	< 15%	6.2% (Win) / 5.1% (Linux)
Connections missed (30 min)	0	0 on all platforms

All measured values fell well within the performance requirements defined in the SRS. GeoIP lookups were notably faster than the 300 ms requirement because the MaxMind binary database uses a highly optimized B-tree structure completing most lookups in under 50 ms.

CPU usage remained below 8% across all platforms even during sessions with more than 30 simultaneous active connections, demonstrating that the caching strategy employed by the Process Mapper effectively reduces repeated psutil queries.

Although the current implementation performs effectively, certain limitations still exist. The platform currently requires administrator privileges for packet capture and firewall operations. Deep packet inspection is deliberately excluded by design. The system also does not yet include AI-based predictive anomaly detection or cloud-synchronized threat intelligence feeds. Overall, the results demonstrate that Rahu is a scalable, efficient, and practical solution for personal cyber security monitoring and network transparency applications

7. CONCLUSION

Rahu: The Personal Visual Firewall Log successfully combines real-time network monitoring, process-level connection mapping, GeoIP-based destination visualization, and one-click OS firewall integration into a unified desktop application. The system addresses major limitations present in traditional network monitoring tools and commercial firewall products by integrating live map-based

interaction, transparent process identification, risk-based alerting, and privacy-preserving encrypted logging into a single environment.

The platform enables users to monitor all outgoing network connections efficiently while simultaneously improving environmental awareness through the interactive Leaflet.js map and risk score indicators. The integration of local-only GeoIP resolution and Fernet-encrypted storage ensures that users can monitor their network activity without compromising sensitive personal information.

The use of modern technologies including Python, FastAPI, Electron.js, Scapy, psutil, and MaxMind GeoLite2 provides a scalable and responsive foundation for real-time host-based firewall applications. Experimental evaluation demonstrated that the platform performs efficiently under real-time conditions, meeting all seven functional and six non-functional requirements defined in the SRS.

Overall, Rahu represents a practical and scalable personal cyber security solution capable of supporting modern privacy-aware computing through interactive geospatial network visualization and community-accessible firewall control

8. FUTURE WORK

The proposed system can be further improved by integrating advanced technologies such as machine learningand predictive analytics for intelligent anomaly detection and automated threat prioritization.
Future enhancements may include:

- Machine learning-based anomaly detection using Isolation Forest or Autoencoder models
- Optional deep packet inspection mode for protocol mismatch detection
- Per-application network bandwidth tracking and usage statistics
- Integration with open-source threat intelligence feeds (AbuseIPDB, Spamhaus DROP)
- Mobile companion application for remote monitoring and alert notifications
- Scheduled PDF report generation summarizing daily or weekly network activity

The platform can also be expanded to support enterprise network monitoring and large-scale organizational security auditing in future deployments.

9. REFERENCES

[1] M. Roesch, “Snort – Lightweight Intrusion Detection for Networks,” in Proc. USENIX LISA Conf., 1999.

[2] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin, Firewalls and Internet Security: Repelling the Wily Hacker, 2nd ed. Boston, MA, USA: Addison-Wesley, 2003.

[3] MaxMind Inc., “GeoLite2 Free Geolocation Data.” https://dev.maxmind.com/geoip/geolite2-free-geolocation-data

[4] Scapy Project, “Scapy – Interactive Packet Manipulation Program”. https://scapy.net

[5] psutil Documentation, “Cross-Platform Library for Retrieving Information on Running Processes and System Utilization”. https://psutil.readthedocs.io

[6] Electron Foundation, “Electron Documentation.” https://www.electronjs.org/docs

[7] FastAPI Documentation, “FastAPI – Modern, Fast Web Framework for Building APIs with Python.” https://fastapi.tiangolo.com

[8] A. Papadogiannakis, M. Polychronakis, and E. P. Markatos, “Improving the Accuracy of Network Intrusion Detection Systems under Load,” IEEE Trans. Comput., vol. 61, no. 4, pp. 562–574, 2012.

[9] G. Giordano, F. Spezialetti, and G. Bianchi, “A User-Centric Approach to Personal Firewall Configuration,” IEEE Security & Privacy, vol. 9, no.3, pp. 42–49, 2011.

[10] G. Gibb, H. Zeng, and N. McKeown, “OpenFlow: A Proposed Communications Interface for Managing Networks of Switches,” ACM SIGCOMM Comput. Commun. Rev., vol. 41, no.4,2011