Palestine Polytechnic University

College of Information Technology and Computer Engineering

Department of Computer Engineering

HaresNet: A Smart Linux-Based Wi-Fi Router for Secure Home and Small-Office Networks

Ismael Rjoub

To fulfill the requirements for a bachelor's degree in the field of Computer Systems Engineering

February 2026

Abstract

This project presents HaresNet, a comprehensive software-defined router platform designed to transform a standard Linux computer into a fully manageable WiFi router with enterprise-grade features for home and small-office environments. HaresNet addresses the limitations of conventional consumer routers by integrating essential Linux networking services—including hostapd for wireless access, dnsmasq for DHCP/DNS, and nftables for traffic control—into a unified system. The platform leverages an Ethernet connection for WAN connectivity and a USB WiFi adapter for LAN access, providing a clean separation between upstream internet and local network traffic. By combining these standard tools with a custom Python backend and a responsive React-based frontend (”Guardian Angel”), HaresNet offers a robust solution for enhancing network visibility, security, and management without requiring proprietary hardware.

The system delivers a rich set of features including real-time device discovery, per-device access control, time-based scheduling, and dynamic service blocking. It also provides comprehensive traffic monitoring with historical analysis and configurable bandwidth limits. Security is prioritized through a two-factor authentication mechanism for administrative access. This report documents the complete development lifecycle of HaresNet, detailing the theoretical foundations, architectural design, implementation methodology, and extensive testing results which demonstrate the platform’s effectiveness as a flexible and secure alternative to traditional home routers.

الملخص

يقدم هذا المشروع منصة **HaresNet**، وهي منصة راوتر برمجية شاملة مصممة لتحويل جهاز كمبيوتر يعمل بنظام Linux قياسي إلى راوتر WiFi مُدار بالكامل مع ميزات على مستوى المؤسسات للبيئات المنزلية والمكتبية الصغيرة. يعالج HaresNet قيود الراوترات التقليدية من خلال دمج خدمات شبكات Linux الأساسية---بما في ذلك hostapd للوصول اللاسلكي، وdnsmasq لـ DHCP/DNS، وnftables للتحكم في حركة المرور---في نظام موحد. تستفيد المنصة من اتصال Ethernet لشبكة WAN ومحول WiFi USB لشبكة LAN، مما يوفر فصلاً نظيفاً بين الإنترنت وحركة المرور المحلية. من خلال الجمع بين هذه الأدوات القياسية مع backend مخصص بلغة Python وواجهة أمامية سريعة الاستجابة ("Guardian Angel") يقدم HaresNet حلاً قوياً لتعزيز رؤية الشبكة والأمان والإدارة دون الحاجة إلى أجهزة خاصة.

يوفر النظام مجموعة غنية من الميزات بما في ذلك اكتشاف الأجهزة في الوقت الفعلي، والتحكم في الوصول لكل جهاز، والجدولة الزمنية، وحظر الخدمات الديناميكي. كما يوفر مراقبة شاملة لحركة المرور مع تحليل تاريخي وحدود قابلة للتكوين لعرض النطاق الترددي. يتم إعطاء الأولوية للأمان من خلال آلية المصادقة الثنائية للوصول الإداري. يوثق هذا التقرير دورة حياة التطوير الكاملة لـ HaresNet، مفصلاً الأسس النظرية، والتصميم المعماري، ومنهجية التنفيذ، ونتائج الاختبارات الموسعة التي تثبت فعالية المنصة كبديل مرن وآمن للراوترات المنزلية التقليدية.

Contents

List of Figures

List of Tables

List of Abbreviations

ABBREVIATION	Full Name / Description
2FA	Two-Factor Authentication
AP	Access Point
API	Application Programming Interface
ARP	Address Resolution Protocol
CDN	Content Delivery Network
CLI	Command Line Interface
CPU	Central Processing Unit
DB	Database
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DoH	DNS over HTTPS
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
JWT	JSON Web Token
LAN	Local Area Network
MAC	Media Access Control (Address)
NAT	Network Address Translation
OS	Operating System
OTP	One-Time Password
QoS	Quality of Service
REST	Representational State Transfer
SNI	Server Name Indication
SQL	Structured Query Language
SSID	Service Set Identifier
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
URL	Uniform Resource Locator

ABBREVIATION	Full Name / Description
USB	Universal Serial Bus
WAN	Wide Area Network
WEP	Wired Equivalent Privacy (legacy)
WiFi	Wireless Fidelity
WPA2	Wi-Fi Protected Access 2
WPA3	Wi-Fi Protected Access 3

Chapter 1

Introduction

1.1 Preface

In this chapter, we introduce the HaresNet platform and explain the motivation behind developing a software-defined router solution that operates on commodity hardware. We begin by presenting the problem statement, which highlights the limitations of current consumer routers and strict network management needs. We then outline the core objectives of the project, representing the fundamental goals that must be achieved to deliver a functional and valuable solution. Following this, we detail both the functional and non-functional requirements to define precisely what capabilities the system must provide. We state the initial assumptions that establish the baseline conditions for deployment, provide a detailed system description to illustrate how HaresNet operates as an integrated platform, summarize the main design constraints in a structured format, present the project timeline that guided the development process, and finally provide a narrative outline of the remaining chapters in this report.

1.2 Problem Statement

The landscape of home and small-office networking has evolved significantly, yet the tools available to manage these networks have largely stagnated. Modern environments host a diverse array of connected devices—from laptops and smartphones to IoT sensors and smart appliances—creating complex management and security challenges. Standard consumer routers typically function as ”black boxes,” offering basic connectivity but lacking the granular visibility and control required by today’s administrators. Users often cannot easily identify which devices are consuming bandwidth, enforce strict access schedules for children or employees, or block specific distracting or malicious services without technical expertise.

Furthermore, the rigid firmware of commercial routers often precludes the addition of advanced features like real-time traffic analysis or custom firewall rules. This gap forces a choice

between affordable but under-powered consumer gear and expensive, complex enterprise equipment. HaresNet aims to bridge this divide by providing a software-defined solution that brings transparency, flexibility, and enterprise-grade control to standard hardware, empowering non-technical users to secure and manage their networks effectively.

1.3 Aims and Objectives

In this project, we propose HaresNet as a comprehensive solution that aims to achieve the following objectives:

- **Build a Practical Software-Defined Router:** Transform standard Linux computers into fully functional WiFi routers using a USB WiFi adapter and standard networking tools, sharing internet from an Ethernet WAN to a wireless LAN.
- **Provide Network Control for Non-Technical Users:** Design an intuitive, responsive web-based dashboard that presents device status and controls without requiring command-line interaction.
- **Enforce Per-Device Policies:** Implement robust controls including time-based scheduling, domain/service blocking via firewall rules, and configurable traffic limits.
- **Deliver Monitoring and Observability:** Provide real-time traffic monitoring per device, historical data storage, and push notifications for key network events.

1.4 Requirements

This section presents the detailed requirements that HaresNet must satisfy to achieve the objectives outlined above.

1.4.1 Functional Requirements

The functional requirements define the specific capabilities of the HaresNet platform:

- **Access Point Management:** Configure and manage a secure wireless access point supporting WPA2/WPA3 encryption, ensuring reliable connectivity on the 2.4GHz band.
- **Device Discovery & Management:** Automatically detect connected devices via DHCP/ARP, display their details (MAC, IP, hostname), and allow administrators to block or unblock internet access instantly.
- **Time-Based Scheduling:** Enable administrators to create flexible access schedules that automatically allow or block specific devices during defined time windows.

- **Service & URL Blocking:** Support blocking of specific websites and services by resolving domains to IP addresses and enforcing firewall rules, with automatic IP refreshing.
- **Traffic Monitoring:** Monitor real-time upload/download speeds for each device and store historical usage data for analysis and visualization.

1.4.2 Non-Functional Requirements

Non-functional requirements define the quality attributes of the system:

- **Usability:** Administrative actions must be accessible via the web dashboard without CLI knowledge, providing clear feedback.
- **Reliability:** Core services (AP, DHCP, Firewall) must auto-start and recover from failures; policies must persist across reboots.
- **Performance:** Dashboard should load quickly, and traffic monitoring must introduce minimal overhead.
- **Security:** Passwords must be hashed, sessions secured with signed tokens, and firewall policies should be robust.
- **Scalability:** The system must support at least 50 concurrent client devices without significant degradation.

1.5 System Description

HaresNet is implemented as a modular software platform running on a Linux host, consisting of a responsive frontend, a Python-based API, and a low-level service layer. The user interacts with ”Guardian Angel,” a modern web dashboard that provides real-time visualizations and control forms. This frontend communicates via REST API and WebSockets with a Flask backend, which acts as the system’s brain, handling authentication, data persistence in SQLite, and orchestrating the underlying Linux services.

The service layer translates high-level user intents into concrete system configurations. It manages `hostapd` for the WiFi access point and `dnsmasq` for DHCP/DNS services. Crucially, it interfaces with the Linux kernel’s `nftables` framework to enforce firewall policies, relying on packet filtering for device blocking and traffic counting. this layered architecture ensures that robust, kernel-level networking performance is accessible through a user-friendly interface.

1.6 Design Constraints

Building a software router on general-purpose hardware introduces specific constraints:

Table 1.1: Design Constraints and their Impact

Constraint	**Description**
Hardware Dependency	Wireless capability depends on the USB WiFi adapter's chipset and driver support for AP mode.
Service Blocking	Blocking large services requires complex, continuous IP resolution due to CDNs.
Encrypted DNS	DoH/DoT evolution challenges traditional DNS-based filtering, requiring firewall-level mitigations.
Privilege Requirements	Core networking requires root privileges, necessitating careful security design for the web backend.
Single Point of Failure	As the central gateway, host system failure disconnects all devices, requiring robust recovery.

1.7 Implementation Timeline

The development of HaresNet followed a structured timeline that balanced feature implementation with testing and refinement. Table 1.2 presents the project schedule.

Table 1.2: Project Gantt Chart

Task	First Semester (Weeks)					Second Semester (Weeks)				
	1–2	3–5	6–8	9–11	12–16	1–2	3–5	6–8	9–11	12–16
Idea Selection and Proposal	■	■								
Literature Review and Background		■	■							
Requirements and High-Level Design				■	■					
Networking Stack Implementation				■	■	■				
Backend and Dashboard Implementation					■	■	■			
Testing and Evaluation				■	■	■	■			
Documentation and Final Report		■	■	■	■	■	■			

1.8 Report Outline

This report documents the development of HaresNet across six chapters: Chapter 1 introduces the project context and requirements; Chapter 2 reviews theoretical foundations and related work; Chapter 3 details the system design and architecture; Chapter 4 describes the implementation methodology; Chapter 5 presents testing and evaluation results; and Chapter 6 concludes with future work and final remarks.

Chapter 2

Background

2.1 Preface

This chapter presents the core concepts and technologies that enable **HaresNet** to operate as an Ubuntu-based smart router with a web dashboard. We focus on the practical foundations behind routing and NAT, Wi-Fi access point management, DHCP/DNS services, and firewall policy enforcement using `nftables`. We then summarize the web application architecture used to manage the platform and discuss the security mechanisms that protect the administrator interface and system configuration. In addition, we clarify how dashboard updates can be handled in a standard web (legacy) interaction model without relying on persistent real-time channels.

After the theoretical background, we review existing router platforms and network filtering tools to clarify the gap HaresNet addresses. This chapter provides the necessary background for the design and implementation details presented in the next chapter.

2.2 Linux Routing Fundamentals

A router forwards traffic between different networks. In HaresNet, this mainly means forwarding packets between the **LAN** (Wi-Fi clients) and the **WAN** (Internet uplink). Linux can perform routing efficiently in the kernel, but it must be configured correctly to behave as a gateway device.

2.2.1 IP Forwarding

IP forwarding enables the Linux kernel to route packets across interfaces. It is typically disabled by default and must be enabled through `sysctl` (for example via `net.ipv4.ip_forward=1`) to allow LAN clients to reach the WAN. Once enabled, the kernel consults its routing table to decide the outgoing interface and forwards traffic without requiring a user-space forwarding process.

2.2.2 Network Address Translation (NAT)

NAT allows multiple LAN devices to share a single public IP address. HaresNet relies on source NAT (*masquerading*) so that outbound packets from private LAN addresses appear to originate from the WAN address. Connection tracking maintains the mapping for return traffic, enabling correct bidirectional communication. This approach is practical for home/SOHO networks and aligns with standard Linux-based gateway deployments.

2.3 Wireless Access Point Management

HaresNet provides Wi-Fi connectivity using `hostapd`, which is widely used to implement IEEE 802.11 access points on Linux systems [**?**, **?**]. The access point layer is responsible for advertising the network, accepting client associations, and enforcing the selected Wi-Fi security mode.

2.3.1 `hostapd` Role and Configuration

`hostapd` runs as a daemon and controls the Wi-Fi interface in access point mode. It is configured using parameters such as SSID, channel, country code, and authentication/encryption settings. In HaresNet, the dashboard selections are translated into a validated `hostapd` configuration file to reduce misconfiguration risks and syntax errors compared to manual editing.

2.3.2 Supported Security Modes

HaresNet supports multiple Wi-Fi security standards to accommodate different deployment scenarios:

- **WPA2-PSK:** The recommended standard for most deployments. Widely supported across all modern devices and provides strong security with properly configured passphrases.
- **WPA3-SAE:** The latest security standard offering improved protection against offline dictionary attacks and forward secrecy. Requires compatible client devices and wireless adapters.
- **WPA2/WPA3 Mixed Mode:** Transition mode supporting both WPA2 and WPA3 clients simultaneously, useful during migration periods.
- **OPEN:** Unencrypted network mode. Available for specific use cases but not recommended for general deployments.
- **WEP:** Legacy encryption standard included only for compatibility with extremely old devices. WEP suffers from critical cryptographic vulnerabilities and should not be used in practice.

For production deployments, HaresNet recommends using WPA2-PSK as the minimum security standard, with WPA3-SAE preferred when client compatibility allows.

2.3.3 Applying Changes Safely

Certain changes (such as SSID or password updates) require restarting `hostapd`, which disconnects clients briefly. HaresNet applies configuration updates in a controlled way by regenerating the config and restarting services only when necessary, minimizing downtime and avoiding partial inconsistent states.

2.4 DHCP and DNS Services

Client connectivity depends on automatic address assignment and name resolution. HaresNet uses **`dnsmasq`** because it is lightweight and well-suited for small networks [**?**].

2.4.1 DHCP Address Assignment

When a device joins the Wi-Fi network, DHCP assigns it an IP address and essential parameters such as default gateway and DNS settings. `dnsmasq` maintains a lease database mapping MAC addresses to assigned IPs and lease times. This lease information is also useful for tracking active clients over time.

2.4.2 DNS Forwarding and Local Resolution

`dnsmasq` forwards DNS queries from LAN clients to upstream resolvers, and it can cache responses to improve performance. DNS can also support basic domain-based filtering, but this is not sufficient alone when clients bypass router DNS (for example using encrypted DNS). For this reason, HaresNet's policy enforcement is designed to rely primarily on firewall-level control rather than DNS-only blocking.

2.4.3 Integration with Device Visibility

HaresNet periodically reads DHCP leases and network tables (such as ARP mappings) to maintain an up-to-date device list. This supports device labeling and policy application based on stable identifiers (especially MAC addresses), improving accuracy even when IP addresses change.

2.5 Firewall Policy Enforcement with `nftables`

Policy enforcement in HaresNet is implemented using **`nftables`**, the modern Linux packet filtering framework [**?**]. It allows consistent handling of filtering and NAT rules within a unified ruleset.

2.5.1 Ruleset Structure

`nftables` organizes policy using **tables**, **chains**, and **rules**. Chains correspond to points in packet processing (input, forward, output, and NAT hooks). Rules match packet attributes (protocols, addresses, interfaces, MAC addresses) and apply actions such as accept or drop.

2.5.2 Device Blocking

Device blocking is implemented by adding rules that match the device MAC address and drop forwarded traffic. Matching by MAC is important because IP addresses can change due to DHCP, while MAC addresses remain stable. This makes device-level policy enforcement more reliable for home networks.

2.5.3 Service Blocking and Updates

Blocking online services often requires tracking changing infrastructure (multiple IPs and frequent updates). HaresNet supports service blocking by maintaining sets of destination IPs and applying drop rules for selected devices. The backend can refresh these sets periodically to keep blocking effective as service endpoints change.

2.5.4 Traffic Measurement

`nftables` supports counters for packets and bytes. By reading counters over time and computing differences between intervals, HaresNet can estimate device usage and traffic rate for dashboard visibility and policy decisions (such as limits and alerts).

2.6 Web Application Architecture

HaresNet is managed through a web interface designed to simplify configuration and provide visibility into connected devices and policies.

2.6.1 Backend API with Flask

The backend uses **Flask** to provide HTTP APIs for configuration management and device control [**?**]. Endpoints are organized into logical modules (for example: authentication, device management, policy enforcement, and monitoring). Flask is suitable here because the system benefits from a clean, modular API without unnecessary complexity.

2.6.2 Database Layer with SQLAlchemy and SQLite

Persistent data is handled using **SQLAlchemy** [**?**] with **SQLite** as the database engine [**?**]. This design fits HaresNet's deployment model because SQLite is file-based and lightweight, while SQLAlchemy provides a consistent ORM approach for device records, labels, schedules, services, and configuration state.

2.6.3 Dashboard Update Model (Legacy Web Interaction)

In the current HaresNet version, the dashboard follows a standard web interaction style. Administrative actions (such as blocking/unblocking a device, applying a schedule, or changing Wi-Fi settings) are performed through normal HTTP requests to the backend API. Monitoring views can be kept up-to-date either by user-triggered refresh actions or by lightweight periodic polling from the browser to fetch the latest device list and traffic counters.

This approach improves compatibility and reduces deployment complexity because it does not require persistent connections or special real-time infrastructure. It also keeps the system behavior predictable for small networks, where clarity and reliability are often more important than continuous streaming updates.

2.6.4 Authentication and Session Security

HaresNet protects administrative actions using JWT-based authentication via **Flask-JWT-Extended** [**?**]. Passwords are stored using **bcrypt** hashing [**?**], aligned with recommended password storage practices [**?**]. JWT expiration and access control ensure that sensitive operations remain protected even if a session token is exposed.

2.7 Related Work and Comparative Analysis

Existing solutions in home/SOHO networking generally fall into two groups: full router/firewall platforms with broad features (often complex), and DNS-based filtering tools with strong usability but limited enforcement scope. HaresNet is positioned as an integrated, device-centric

platform that runs on standard hardware and focuses on practical policies, visibility, and usability.

Table 2.1: Comparative analysis of related network management solutions

Solution	**Primary Strengths**	**Typical Limitations**	**HaresNet Advantage**
OpenWrt [**?**]	Powerful features, wide hardware support, extensible packages	Requires firmware flashing, complex configuration, technical learning curve	Runs on standard PC hardware with a simpler device-focused dashboard
pfSense / OPNsense [**?**, **?**]	Enterprise-grade firewall features, strong networking controls	Oriented toward advanced admins, large feature surface, often dedicated hardware	Focuses on home/SOHO needs with simpler workflows and device policies
Pi-hole [**?**]	Excellent DNS-based blocking, clear web UI, easy deployment	DNS-layer only, not a full router, limited enforcement if DNS is bypassed	Firewall-enforced policies + routing + device scheduling in one platform
AdGuard Home [**?**]	Modern DNS filtering, per-client features, supports encrypted DNS options	Still DNS-focused, not a complete gateway/router, limited traffic control	Integrated routing/NAT, device control, monitoring, and service blocking

The comparison in Table 2.1 highlights a repeated limitation in current tools. Router platforms such as OpenWrt and pfSense/OPNsense provide strong capabilities, but they typically assume technical expertise and may require firmware changes or dedicated appliances. In contrast, DNS-based tools such as Pi-hole and AdGuard Home offer usability and visibility, but their enforcement can be weakened when clients bypass router DNS or use encrypted DNS mechanisms.

HaresNet addresses these gaps by combining **complete gateway functionality** (routing and NAT), **Wi-Fi access point management**, **firewall-enforced device/service policies**, and **traffic monitoring** into a single platform designed for small environments. Instead of relying only on DNS filtering, HaresNet enforces key controls at the firewall layer, improving robustness. At the same time, it prioritizes a device-centric dashboard and practical workflows to reduce configuration errors and make common actions (block/unblock, schedule policies, monitor usage) easy to apply and reverse.

2.8 Security and Privacy Considerations

Consumer networking devices frequently suffer from weak defaults and inconsistent update practices, which increases exposure to attacks and misconfiguration risks [**?**, **?**, **?**]. HaresNet is designed with a security-first mindset to protect both the administrator interface and the managed network.

Key protections include secure password storage using bcrypt [**?**, **?**], JWT-based authentication with expiration and role-based access control [**?**], deny-by-default policy design in the firewall, and controlled configuration application (avoiding partial states). Privacy is also considered by limiting what is stored persistently and keeping monitoring focused on network-level measurements needed for management and enforcement.

2.9 Chapter Summary

This chapter introduced the theoretical background required to understand HaresNet. We reviewed Linux routing essentials (IP forwarding and NAT), Wi-Fi access point management through `hostapd`, DHCP/DNS services using `dnsmasq`, and firewall policy enforcement with `nftables`. We also summarized the web architecture that enables a practical dashboard, including Flask APIs and SQLAlchemy/SQLite persistence, and clarified how dashboard state can be updated using a standard web interaction model.

Finally, we compared HaresNet with common router platforms and DNS filtering tools. The review shows that existing solutions are either powerful but complex, or user-friendly but limited in enforcement scope. HaresNet bridges this gap by delivering an integrated, device-centric solution that runs on standard hardware and enforces policies at the firewall level while maintaining usability for home and small-office environments.

Chapter 3

System Design

3.1 Preface

In this chapter, we present the system analysis and design of **HaresNet**, a platform built to transform a standard Linux device into a reliable, software-defined router with a dedicated management dashboard. The purpose of this chapter is to clarify how the system is structured, how its components are organized, and how the overall design supports stable routing, policy enforcement, and centralized administration.

We begin by describing the architectural blueprint of HaresNet, focusing on the high-level system layers and the relationships between them. We then outline the hardware and software requirements needed to operate the platform correctly. After that, we explain the end-to-end network flow to show how traffic moves through the router and where policy enforcement takes place. Finally, we document the data model used for state management and conclude with the security principles applied to protect integrity and confidentiality across the system.

3.2 System Architecture

HaresNet is designed around a modular, layered architecture that emphasizes maintainability and scalability. Instead of implementing all router logic as a single unit, the system is separated into four clear layers, where each layer has a specific responsibility and communicates with the others through defined interfaces:

- **Presentation Layer (Guardian Angel):** A responsive React-based web interface that acts as the administrator's control center, supporting real-time monitoring and configuration actions.
- **API Layer (Flask Backend):** The main orchestration layer that exposes RESTful endpoints, manages authentication, and coordinates the interaction between the dashboard and the underlying router services.

- **Service Layer:** A set of specialized modules, each responsible for a focused router function such as access point control, traffic monitoring, and firewall policy generation.
- **System Layer:** The foundation that relies on native Linux networking subsystems (`hostapd`, `dnsmasq`, and `nftables`) to perform packet forwarding, signal broadcasting, and rule enforcement.

Figure 3.1 illustrates how these layers interact as a unified system. In addition, Figure 3.2 presents a detailed block-level view of the internal components and their communication paths, while Figure 3.3 summarizes the primary administrator use cases and system responses.

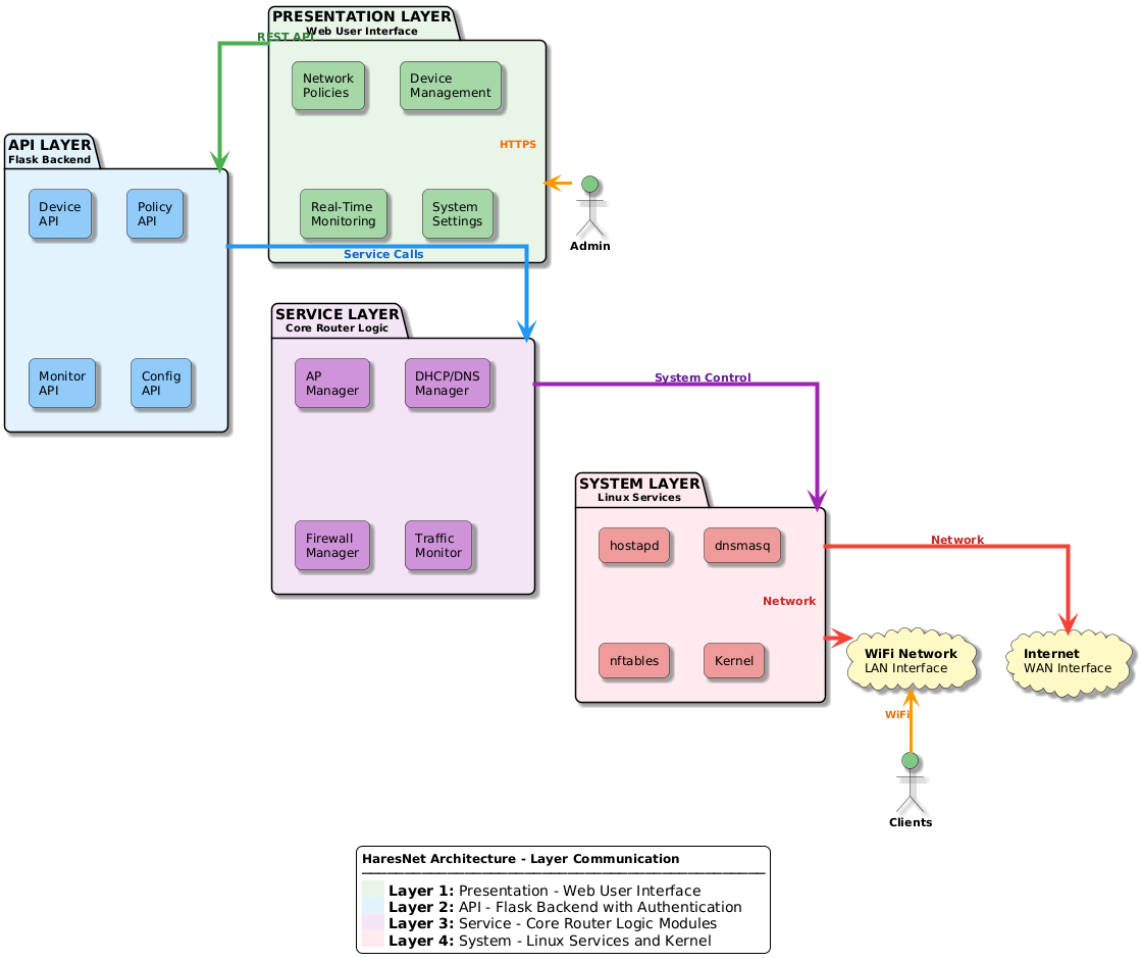

Figure 3.1: Architectural layers of HaresNet

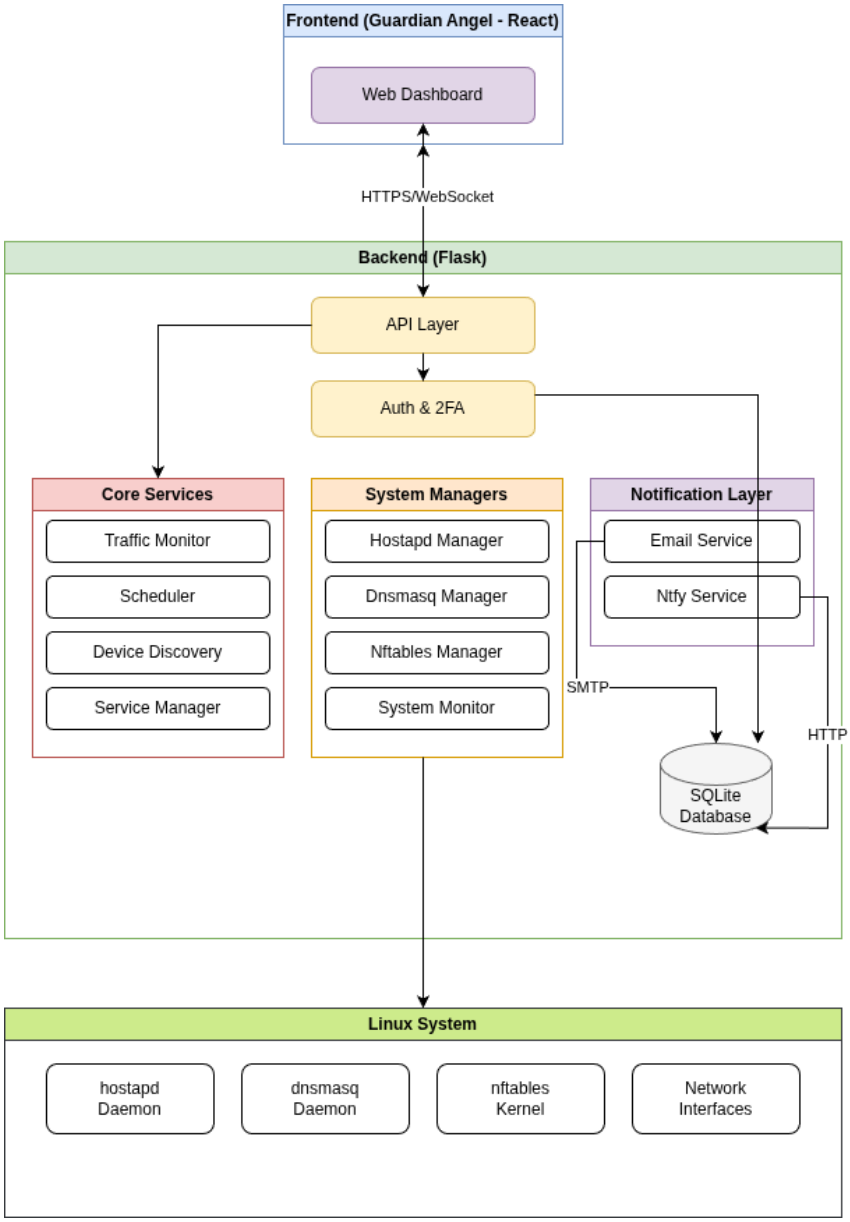

Figure 3.2: Detailed Block Diagram of System Components and Interactions

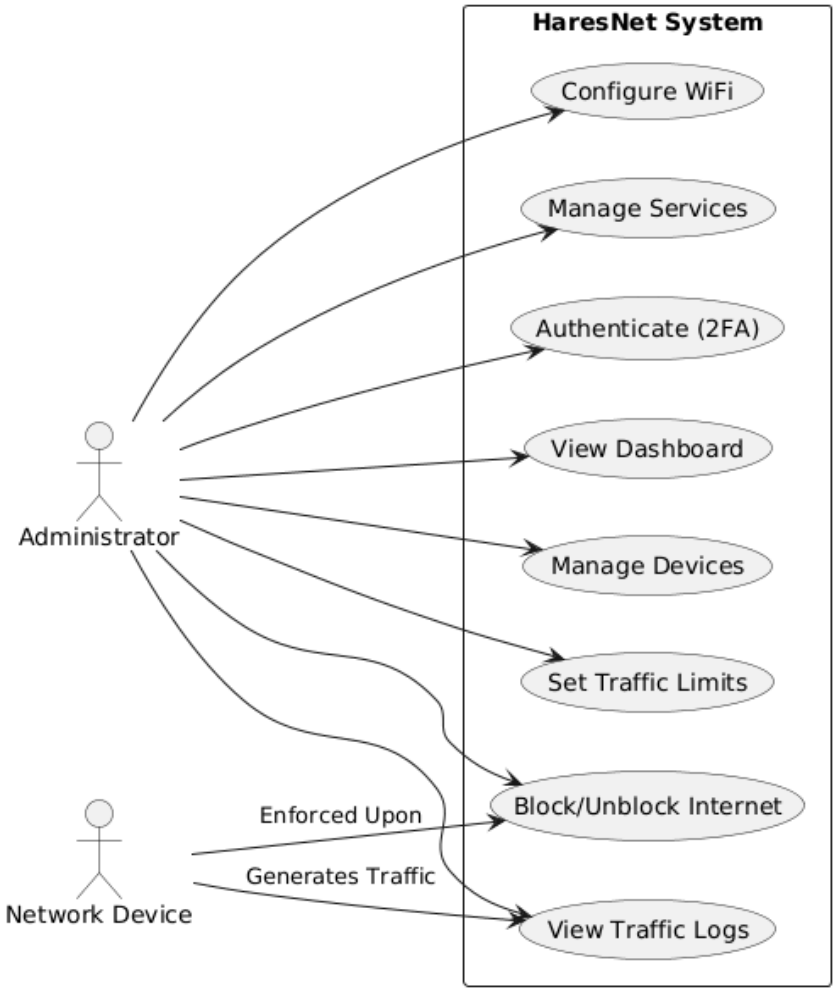

Figure 3.3: Use Case Diagram showing Administrator and System Interactions

3.3 Hardware Requirements

To provide routing and centralized management, HaresNet depends on specific hardware elements that form the physical layer for all network traffic. While the system remains flexible in deployment choices, it relies on a consistent baseline of components to ensure stable operation:

- **Compute Unit:** A Linux-capable device such as a Raspberry Pi (Model 4 or 5) or a Mini PC, responsible for hosting the operating system, running the HaresNet software stack, and supporting the Docker container runtime.
- **WAN Interface (Ethernet Adapter):** A dedicated Ethernet interface that connects the system to the upstream internet source (modem or gateway), carrying all inbound and outbound traffic between the local network and the internet.
- **LAN Interface (Wireless Adapter):** A USB or PCIe Wi-Fi adapter that supports **Access Point (AP)** or **Monitor** mode to broadcast the wireless network. For stronger coverage and more stable signal reach, high-gain antennas are recommended.

3.4 Deployment and Containerization

HaresNet adopts a container-based deployment model to ensure reproducibility and simplify installation across different devices. By relying on **Docker**, the system packages its services into portable containers, reducing dependency conflicts and ensuring consistent behavior across environments.

The deployment is managed using **Docker Compose**, where the backend API, the frontend web server, the database, and the message broker are defined as separate services. This approach supports reliable execution on different platforms (including Raspberry Pi and x86 systems) while keeping the host environment clean and predictable. In addition, containerization simplifies updates and rollbacks, aligning with practical DevOps workflows and maintenance requirements.

3.5 Network Topology and Traffic Flow

The network topology of HaresNet is structured to separate the trusted local Wi-Fi network from the untrusted public internet. As shown in Figure 3.4, client traffic follows a clear path through the router, ensuring that policy enforcement occurs before packets are permitted to reach the WAN interface:

1. Client devices connect to the HaresNet WiFi Access Point and obtain network settings through DHCP.
2. Traffic generated by clients enters through the LAN interface and is processed by the Linux kernel forwarding engine.
3. **nftables** enforces the firewall rules by filtering traffic according to device policies, service blocks, and time-based schedules, allowing only permitted packets to proceed.
4. Allowed traffic is translated using Network Address Translation (NAT/Masquerading) so multiple clients can share the WAN address, then forwarded to the internet.

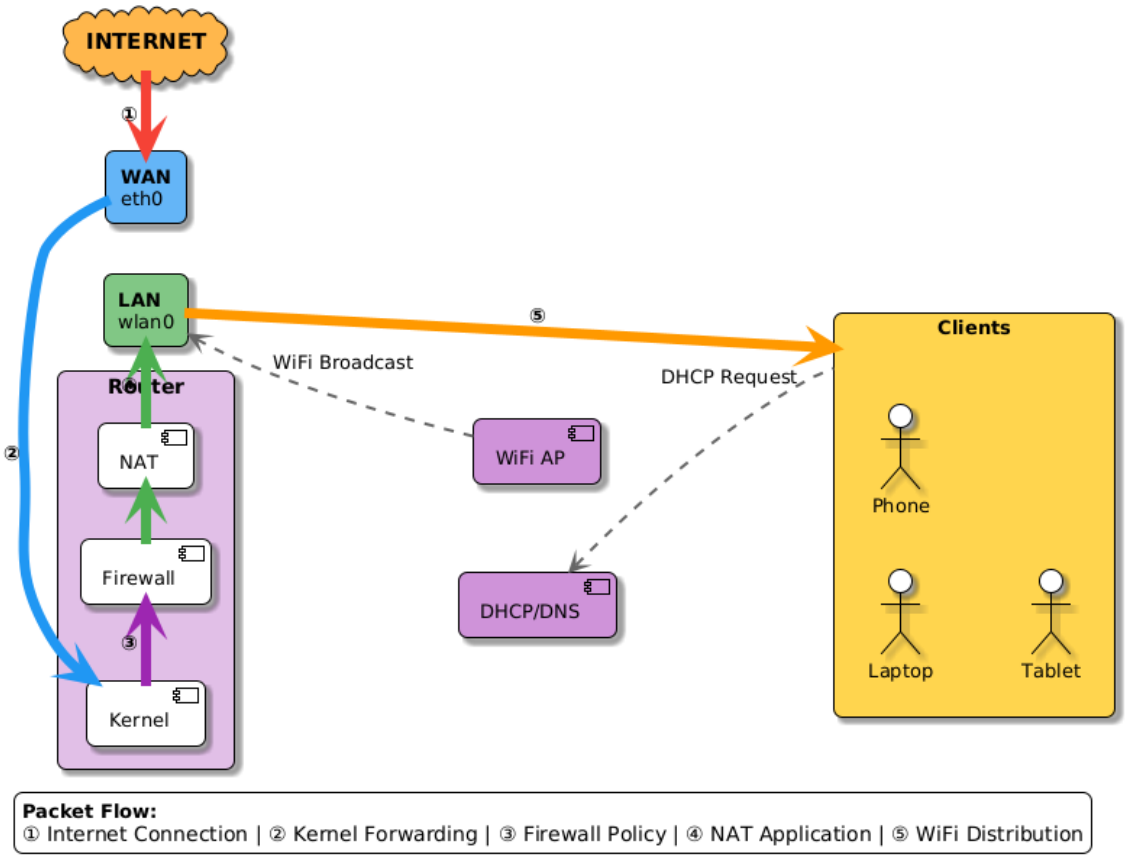

Figure 3.4: HaresNet network topology and traffic flow

3.6 Core Software Components

The backend is organized as a set of focused modules, where each module is designed with a clear responsibility to keep the system structured and maintainable. Table 3.1 summarizes the core components and explains their roles within the HaresNet ecosystem.

Table 3.1: Core backend components

Component	Responsibility
`hostapd_manager`	Manages the access point lifecycle by generating configuration files and applying secure restart procedures.
`dnsmasq_manager`	Controls DHCP and DNS services and maintains the lease database used for device identity tracking.
`nftables_manager`	Converts high-level policies into nftables rules and applies kernel-level enforcement for packet filtering.
`device_discovery`	Observes the network to detect new devices and maintain a real-time inventory of connected clients.
`traffic_monitor`	Collects traffic counters and usage data to support monitoring features and policy enforcement such as quotas.
`scheduler`	Enforces time-based access policies by automatically enabling or disabling rules within defined windows.
`service_manager`	Maintains structured service definitions to support intelligent blocking of platform categories.

3.7 Data Model

To maintain state consistently and support policy-driven management, HaresNet stores its system data in a relational database. The data model is structured to represent the relationships between administrators, devices, enforcement rules, and observed traffic activity. Figure 3.5 presents the class diagram of the schema and highlights how entities such as `Devices`, `Rules`, and `TrafficLogs` connect to support enforcement and monitoring.

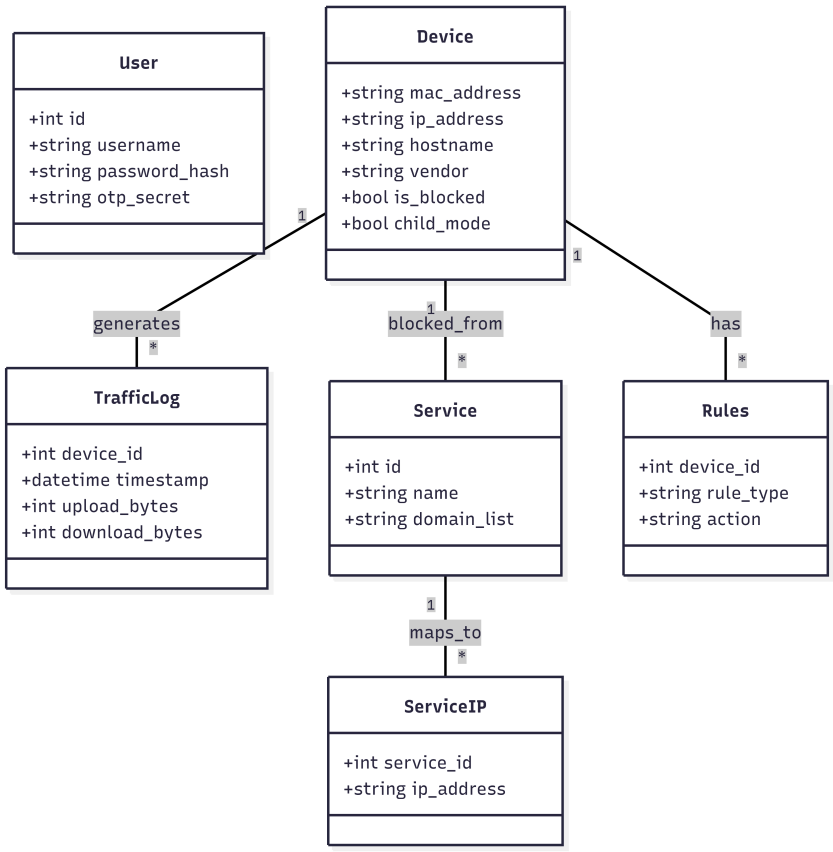

Figure 3.5: Entity-Relationship Class Diagram of the HaresNet Database

3.8 Security Design

Security is a foundational requirement in HaresNet because the platform controls network access and applies enforcement policies that affect all connected clients. To ensure integrity and confidentiality, the system follows a security-first design that relies on layered protection and strict access control:

- **Strong Authentication:** The administration console is protected using bcrypt-hashed passwords combined with Time-based One-Time Passwords (TOTP) to provide two-factor verification.
- **Least Privilege Principle:** The API enforces authorization checks on all sensitive endpoints, ensuring only authenticated sessions can modify configuration and system state.
- **Defense in Depth:** Policy enforcement is applied at the kernel level using `nftables`, ensuring restrictions remain effective even if the management interface is unavailable or compromised.

3.9 Chapter Summary

This chapter presented the full design perspective of HaresNet, beginning with its layered architecture and continuing through hardware requirements, containerized deployment, network traffic flow, backend module structure, database modeling, and security foundations. By defining these elements clearly, we establish how the system operates as a controlled gateway and how its design choices support stable enforcement and maintainability. In the next chapter, we move from design to implementation by detailing the specific workflows, configurations, and code structures used to realize this architecture in practice.

Chapter 4

Implementation and Methodology

4.1 Preface

This chapter translates the HaresNet design into its practical implementation. We present the development environment, explain the project structure, and clarify the responsibilities of the core modules. We then describe how the router is initialized in practice, including access point setup, DHCP/DNS configuration, and firewall activation. After that, we cover the policy features (device control, service/URL blocking, and scheduling) and the monitoring stack (traffic history, limits, notifications, and system metrics). The chapter ends with deployment and configuration notes that help future developers reproduce the same setup reliably.

4.2 Development Environment

HaresNet was developed and validated on Ubuntu-based systems using the following technologies:

- **Python + Flask** for the backend [**?**].
- **SQLite + SQLAlchemy** for configuration and persistent state [**?**, **?**].
- **Flask-SocketIO** for real-time dashboard updates [**?**].
- **hostapd** and **dnsmasq** for WiFi AP and DHCP/DNS services [**?**, **?**].
- **nftables** for NAT, forwarding policy, and device/service enforcement [**?**].
- Optional **InfluxDB** integration for time-series traffic storage [**?**].

4.3 Project Structure

The system implementation is separated into a backend and a frontend. Figure 4.1 presents the main dashboard interface used for administration.

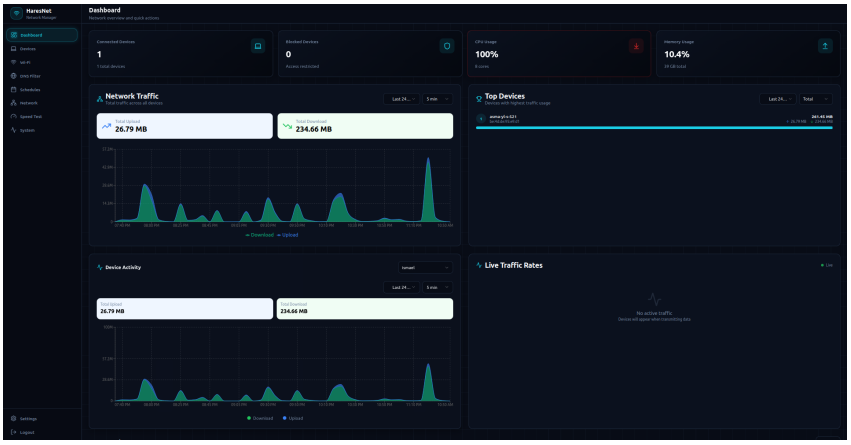

Figure 4.1: Dashboard overview.

The backend is implemented in Python using Flask as the web framework, with SQLAlchemy and SQLite for data persistence. It exposes REST API endpoints that cover authentication, device management, network policy enforcement, service blocking, WiFi configuration, and system monitoring. To keep the dashboard synchronized with the current router state, real-time updates are pushed through Flask-SocketIO WebSocket connections.

To interact with Linux networking services in a clean and controlled manner, the implementation uses dedicated manager modules. The `hostapd_manager` is responsible for controlling the access point daemon, the `dnsmasq_manager` manages DHCP and DNS configuration, and the `nftables_manager` applies firewall rules for NAT, forwarding, and policy enforcement. In parallel, background services run continuously to monitor traffic counters, discover connected devices, enforce scheduled policies, and parse DNS logs.

The frontend is a React single-page application (**Guardian Angel**) that provides the administrative dashboard. The interface displays device status, traffic statistics, and security settings in a responsive layout. All administrative operations are protected using JWT-based authentication with two-factor verification through time-based one-time passwords (TOTP).

4.4 Backend Initialization

The Flask backend is constructed using the application factory pattern, which improves maintainability and supports testing. The factory function (`create_app()`) creates the Flask instance, initializes extensions (SQLAlchemy, Flask-JWT-Extended, Flask-SocketIO), and registers API blueprints to keep routing modular and organized.

Configuration is loaded through environment variables alongside default settings. This enables the same codebase to run across development, testing, and production without requiring code changes. The key parameters include the WAN and LAN interface names, subnet ranges, security secrets (Flask secret key and JWT secret), and notification credentials.

During startup, the backend connects to the SQLite database, creates tables if they are missing, and launches background worker threads for traffic monitoring, device discovery, schedule enforcement, and DNS log parsing. This automated startup process ensures that essential services are running and ready before the dashboard begins sending requests.

4.5 Authentication and Administrator Flow

Administrator access is implemented in `app/api/auth.py`. The flow is:

1. **Credential verification:** the user provides username and password.
2. **OTP generation:** the backend generates a short-lived one-time code.
3. **Code delivery:** the code is delivered through the configured channel (e.g., email or push notification).
4. **OTP verification:** the user submits the code to obtain the authenticated session token.

Passwords are stored as bcrypt hashes [**?**], and session tokens are handled using Flask-JWT-Extended [**?**]. After successful login, the platform also provides an endpoint to update the administrator profile, including username, password, and email.

4.6 WiFi Access Point Implementation

WiFi access point configuration is managed through `hostapd_manager.py`. When the administrator updates the SSID, password, security mode, or channel, HaresNet regenerates the hostapd configuration accordingly and restarts the AP service to apply the new settings.

Supported security modes include WPA2-PSK, WPA3-SAE, WPA2/WPA3 mixed mode, OPEN, and WEP. For production use, the platform recommends WPA2-PSK as the minimum acceptable standard, and prefers WPA3-SAE when device compatibility allows. OPEN and WEP are included only for legacy compatibility requirements and are strongly discouraged due to known security weaknesses.

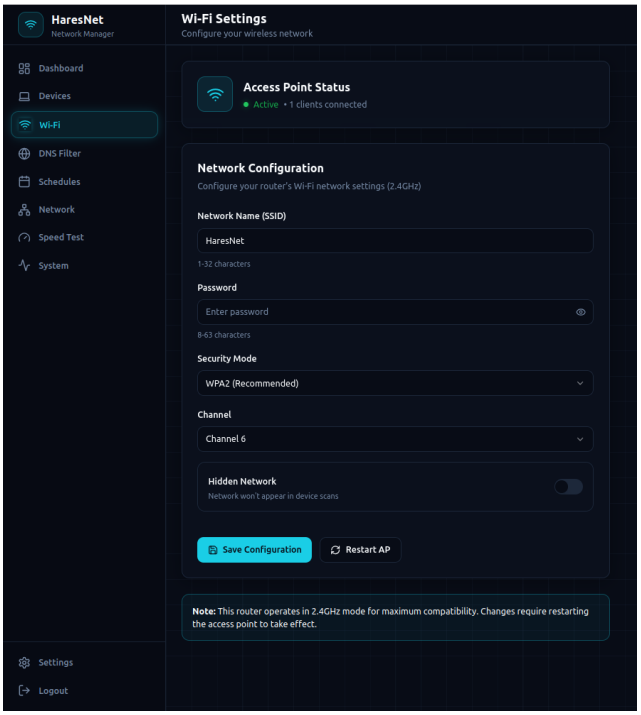

Figure 4.2: WiFi settings screen.

4.7 DHCP/DNS Service Implementation

DHCP and DNS services are configured through `dnsmasq_manager.py`. This layer is responsible for defining the essential network parameters required by clients, including:

- IP lease range and lease time.
- DNS servers for upstream resolution.
- Logging options that can be enabled for troubleshooting and analysis.

4.8 Firewall and Policy Enforcement

Policy enforcement is concentrated in `nftables_manager.py`. This manager generates and applies a consistent nftables ruleset that covers the main router behaviors as well as the policy controls required by the dashboard. The ruleset includes:

- **NAT/masquerading** from LAN to WAN.
- **Forwarding rules** to allow established connections and controlled new connections.
- **Device blocking rules** based on MAC identity.
- **Service blocking rules** using destination IP sets per service.

- **Counter rules** used to measure traffic per device.

By relying on nftables, HaresNet enforces its policies at the kernel level, which keeps restrictions active even when the web UI is not open.

4.8.1 Device Blocking

Device blocking is provided through `app/api/devices.py`. When an administrator blocks a device, HaresNet applies a rule that drops forwarded packets for that device based on its MAC address, while still keeping the device visible in the monitoring views. Figure 4.3 presents the device control interface.

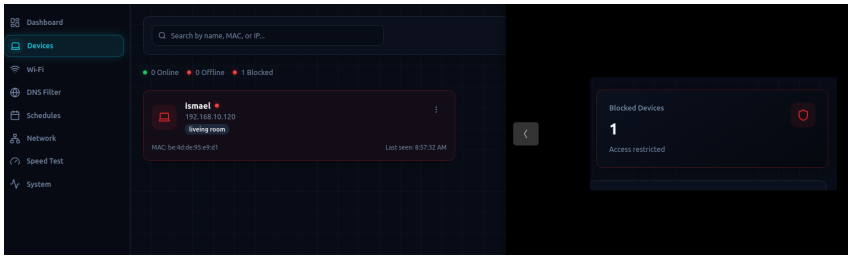

Figure 4.3: Device control actions.

4.8.2 Service and URL Blocking

Service blocking is implemented through `app/api/services.py` and `service_manager.py`. A service can be created using a domain or a full URL entered by the administrator. The backend normalizes this input by extracting the domain and removing protocol and path components. It then resolves the domain into a set of IP addresses and stores them as `ServiceIP` entries.

The firewall layer uses these destination IP sets to block traffic for selected devices. To reduce issues caused by DNS updates and CDN rotation, the system can refresh service IPs over time. Figure 4.4 shows the interface used to configure service and URL blocking.

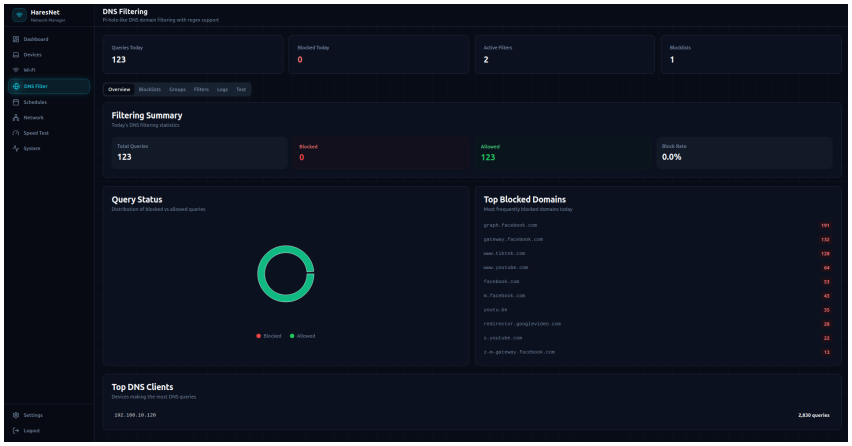

Figure 4.4: Service and URL blocking interface.

4.9 Scheduling

Time-based control is implemented using the scheduler service (`scheduler.py`) with schedules stored persistently. Each schedule specifies:

- A device target.
- A set of days of the week.
- Start and end time.
- Action (allow or block).

The scheduler runs periodic checks to identify active schedules and applies the corresponding policy through the firewall layer. Figure 4.5 presents the scheduling configuration screen.

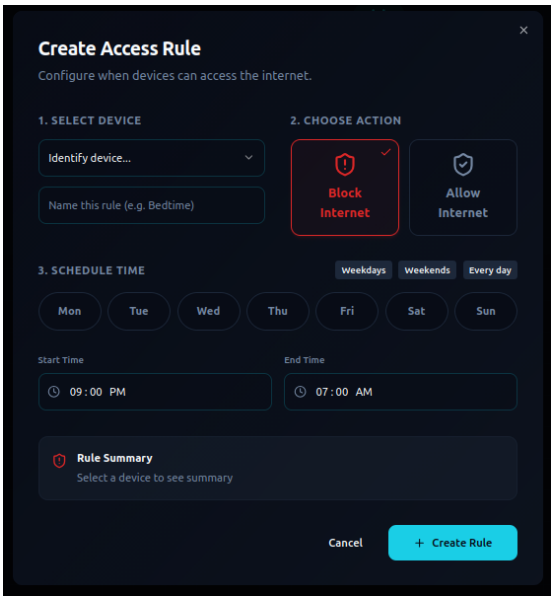

Figure 4.5: Scheduling screen.

4.10 Traffic Monitoring and Limits

Traffic monitoring is implemented by combining nftables counters with periodic collection. The `traffic_monitor.py` service reads per-device counters, calculates deltas, and stores traffic records in the database. The traffic endpoints allow querying by time range and support aggregation (such as 5-minute buckets or hourly summaries), enabling dashboard charts for "last hour", "last day", and "last week" views.

HaresNet supports **daily** and **hourly** traffic limits per device. If a device exceeds its configured limit, the system triggers a notification and records the alert timestamp to prevent repeated alerts over short intervals.

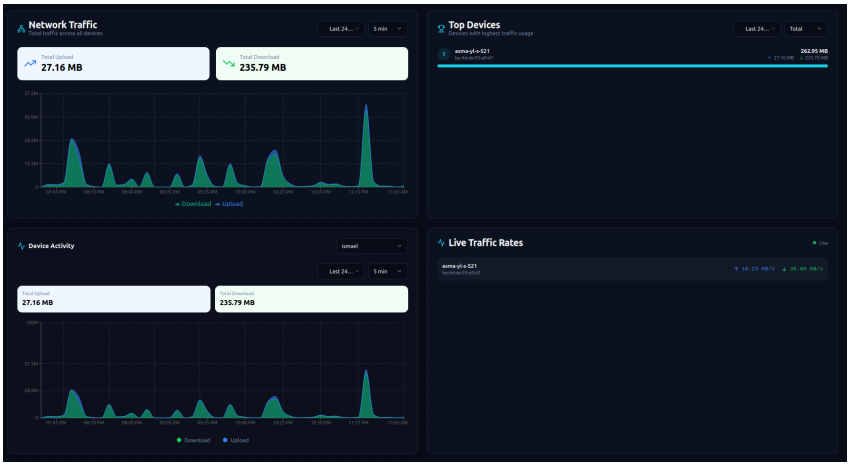

Figure 4.6: Traffic monitoring charts.

4.11 Notifications (Multiple Channels)

HaresNet supports alerts through multiple configurable notification channels so administrators can receive important events via email or mobile push. The supported channels include:

- **NTFY:** Push notifications via the NTFY service (https://ntfy.sh).
- **Email (SMTP):** Email alerts sent through configured SMTP server.
- **Blynk:** Push notifications via Blynk IoT platform (used for 2FA OTP delivery).

Typical notification events include:

- New device connection detected.
- Daily/hourly traffic limit exceeded.
- Large traffic spike exceeding a configurable threshold.
- Two-factor authentication OTP codes.

Figure 4.7 illustrates an example of NTFY push notifications received on a mobile device.

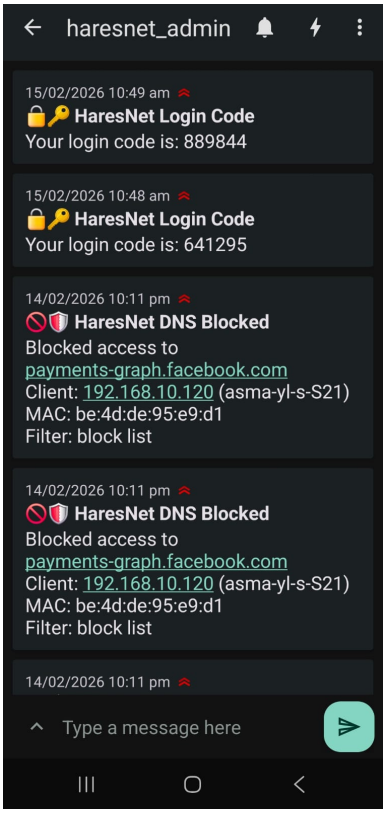

Figure 4.7: NTFY push notification example.

4.12 System Monitoring and Speed Test

System-level monitoring is handled by `system_monitor.py`, which reports CPU usage, memory usage, uptime, and interface statistics. The platform also integrates an internet speed test feature through a CLI tool, allowing administrators to validate WAN performance directly from the dashboard [**?**].

4.13 Deployment and Configuration

HaresNet supports both bare-metal and container-based deployment. When running in container mode, the services must be started with the required network capabilities and privileges so the platform can configure interfaces and apply firewall rules correctly [**?**, **?**].

Configuration Parameters

Deployment is configured through environment variables that define:

- WAN and LAN interface names.

- LAN subnet, DHCP range, and DNS servers.
- Security secrets (Flask secret key, JWT secret key).
- Notification topic and system timezone.

4.14 Chapter Summary

This chapter explained how HaresNet is implemented and how it applies policies through Linux services and nftables. The next chapter evaluates the platform through reproducible experiments, including device control, scheduling, service blocking, monitoring accuracy, and authentication behavior.

Chapter 5

Testing and Results

5.1 Preface

This chapter documents the practical testing performed to validate the security and robustness of **HaresNet** under realistic usage and attack-like scenarios. The focus is not on theoretical discussion, but on confirming that the implemented controls behave as intended: Wi-Fi security modes resist weak-password abuse, traffic confidentiality remains protected against sniffing attempts, and the router services stay responsive during flooding conditions. Each test case includes the goal, test setup, observed result, and a short conclusion that links the outcome to the system design choices described in the previous chapters.

5.2 Testing Environment and Tools

The tests were executed in a controlled lab environment using:

- **HaresNet Router**: Ubuntu-based router device running the HaresNet services (dashboard, firewall policy, and supporting components).
- **Client device**: A regular Wi-Fi client connected to the access point for normal browsing and service validation.
- **Attacker workstation**: Kali Linux was used to run security testing tools in order to simulate a hostile network participant.
- **Monitoring and analysis**: Wireshark was used to observe network behavior and confirm whether traffic contents are readable or encrypted.

5.3 Wi-Fi Security Validation (WPA2 vs WPA3 SAE)

5.3.1 Goal

This test validates that Wi-Fi security mode selection has a direct impact on resistance to offline password guessing, and that using strong credentials significantly increases the difficulty of unauthorized access.

5.3.2 WPA2 with a weak password

Setup: The access point was configured to use WPA2 with a deliberately weak password. The test objective was to verify that weak passwords are practically crackable and therefore unsafe for deployment.

Observed result: The handshake was captured successfully, and the weak password was recovered.

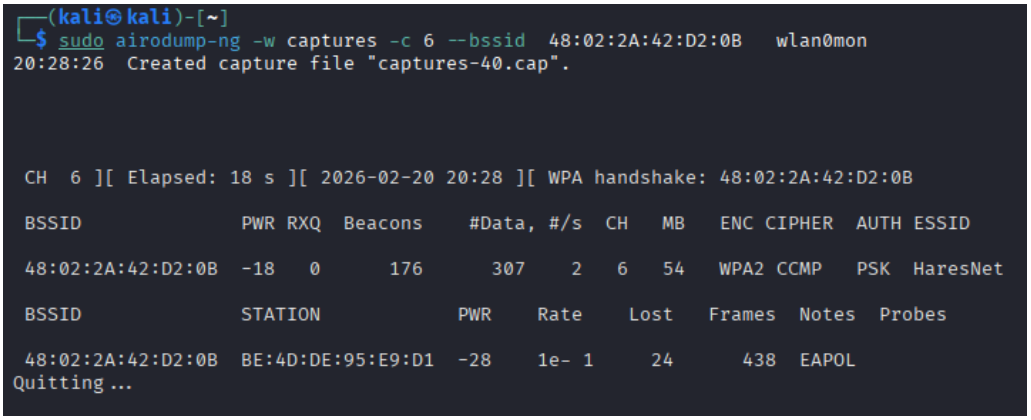

Figure 5.1: Handshake capture attempt during WPA2 testing.

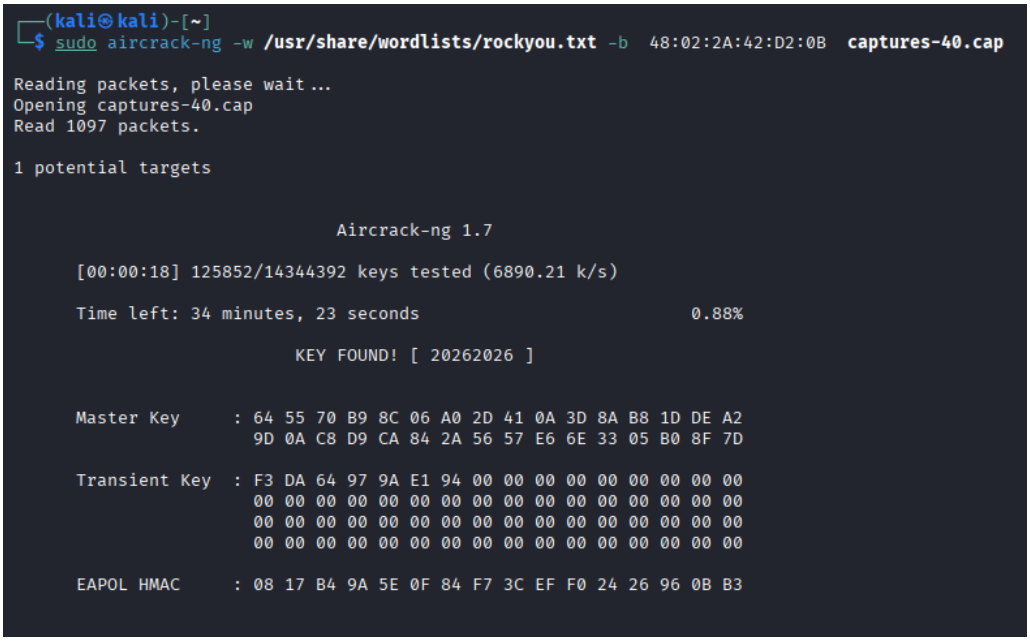

Figure 5.2: Successful cracking result for a weak WPA2 password.

Conclusion: WPA2 itself is widely used, but weak passwords remain a major risk. This confirms the necessity of enforcing strong password policy for WPA2 deployments.

5.3.3 WPA2 with a strong password

Setup: WPA2 was kept enabled, but the password was changed to a strong format (example used during testing: `HARESnet_5@ppu%Srong`).

Observed result: The cracking attempt did not succeed within a long execution time window, indicating that password strength meaningfully increases resistance.

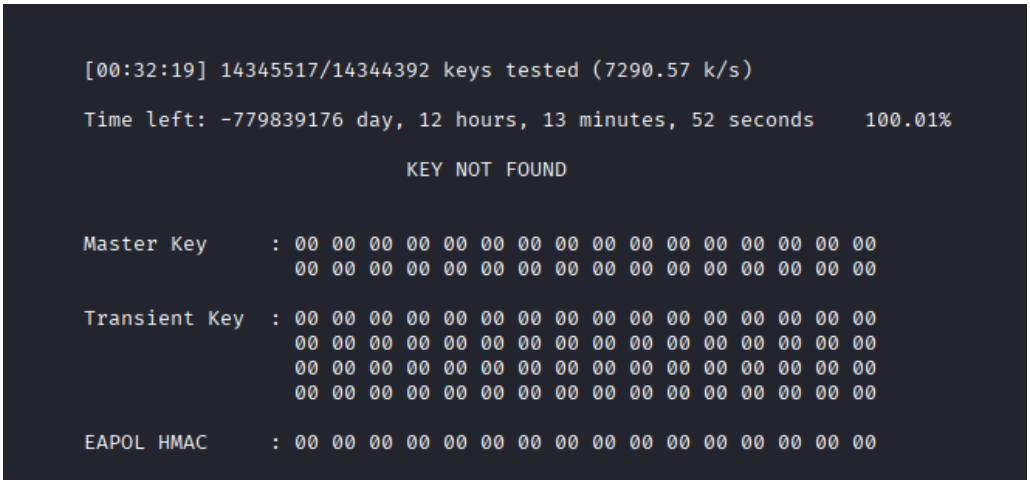

Figure 5.3: WPA2 cracking attempt failing against a strong password.

Conclusion: While WPA2 remains susceptible to password guessing in principle, strong

passwords make the attack impractical in a realistic time frame.

5.3.4 WPA3 (SAE) testing

Setup: The router was configured using WPA3 (SAE). The same general cracking workflow was attempted.

Observed result: The cracking attempt failed as expected under WPA3 SAE conditions.

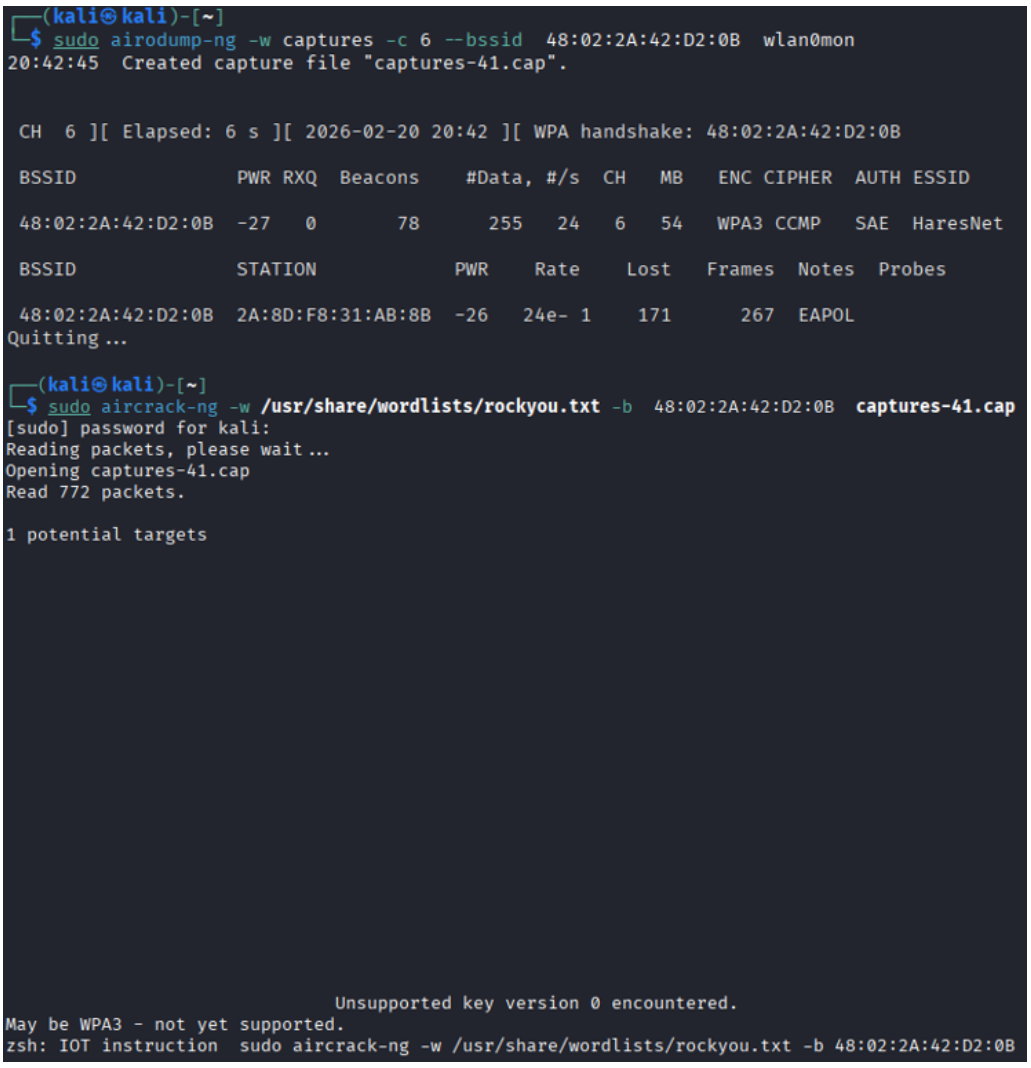

Figure 5.4: WPA3 SAE cracking attempt result (failed).

Conclusion: WPA3 SAE provides improved protection compared to WPA2, and is the recommended mode when supported by client devices.

5.4 Flooding and Denial-of-Service Resilience

5.4.1 Goal

These tests evaluate whether HaresNet stays functional and responsive while receiving high volumes of traffic, and whether normal browsing remains possible during the stress condition.

5.4.2 ICMP ping flood

Setup: ICMP flooding traffic was generated from the attacker workstation towards the router.

Observed result: The router remained responsive. A large portion of packets were dropped/lost, while the rest did not cause visible service disruption.

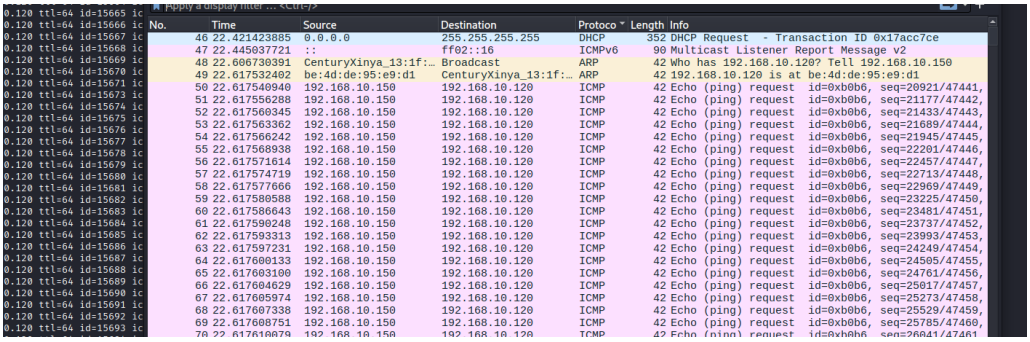

Figure 5.5: Ping flood behavior showing packet loss and continued responsiveness.

Conclusion: The system continued operating normally, indicating reasonable resilience against basic ICMP flooding in the tested environment.

5.4.3 Spoofed source IP attempt

Setup: A spoofed source IP was used to generate traffic, aiming to simulate malformed or inconsistent network identity behavior.

Observed result: The behavior aligned with expectations: the router should recognize valid assigned identities, and spoofed sources did not yield a meaningful advantage in the observed setup.

```
┌──(kali㉿kali)-[~]
└─$ sudo hping3 -1 -a  192.168.10.10 -c 1 192.168.10.120
HPING 192.168.10.120 (wlan0 192.168.10.120): icmp mode set, 28 headers + 0 data bytes

── 192.168.10.120 hping statistic ──
1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms

┌──(kali㉿kali)-[~]
└─$
```

Figure 5.6: Spoofed source attempt during flooding test.

5.4.4 HTTP service flooding (port 80)

Setup: A high-rate traffic stream targeted the router web service on port 80 while the client attempted to access the dashboard page.

Observed result: Wireshark showed heavy traffic, yet the page continued to load without noticeable interruption.

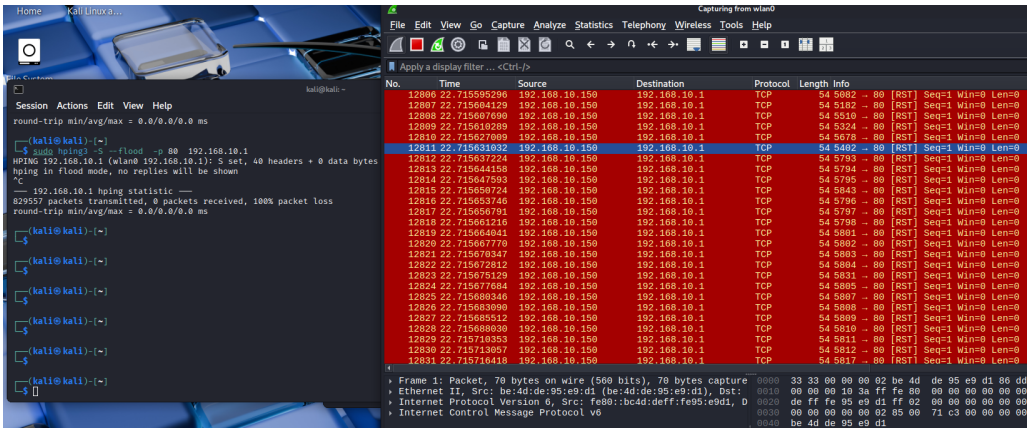

Figure 5.7: Captured traffic during HTTP flooding test.

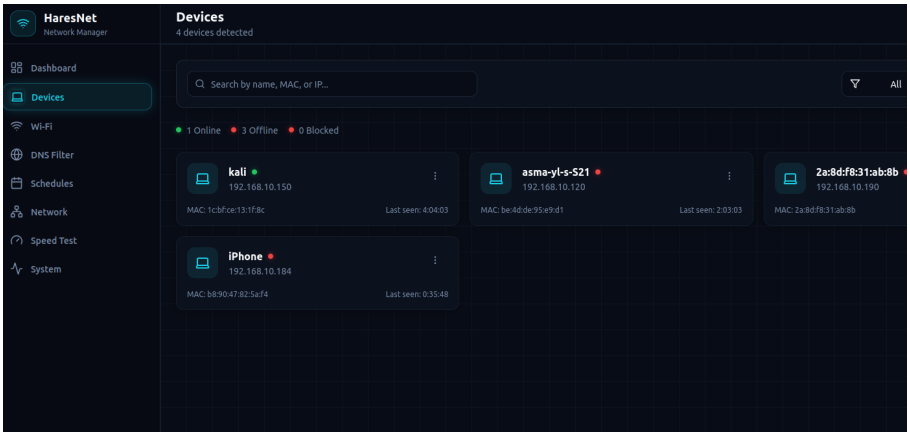

Figure 5.8: Dashboard remains accessible during the flooding condition.

Conclusion: Under the tested load, the service remained accessible, which suggests the stack can tolerate moderate flooding without immediate failure.

5.4.5 MAC flood

Setup: A MAC flooding scenario was generated to stress layer-2 learning behavior.

Observed result: The dashboard page remained accessible, indicating that the system continued normal operation during the test.

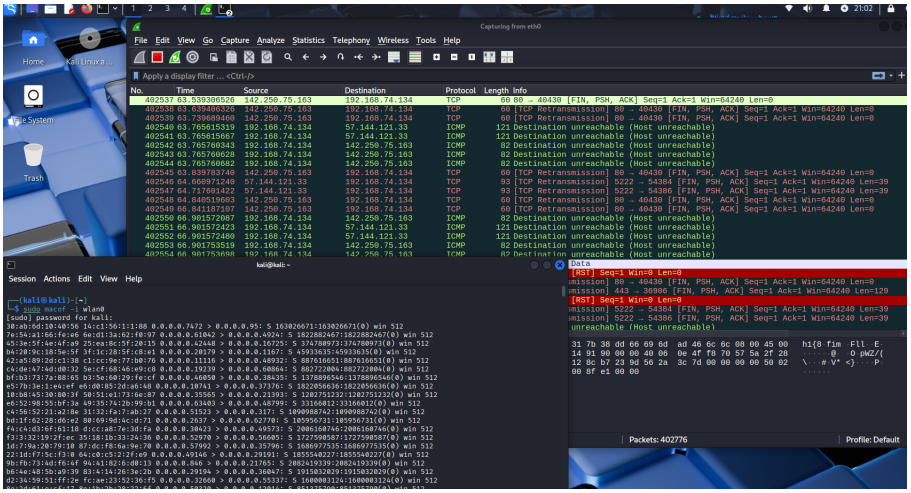

Figure 5.9: Web interface accessibility during MAC flood test.

5.5 Sniffing Attempt and Traffic Confidentiality

5.5.1 Goal

This test confirms that even if an attacker attempts to intercept traffic through ARP spoofing and sniffing, the captured content should not reveal sensitive information in clear text.

5.5.2 ARP spoofing and sniffing

Setup: Network scanning was performed to identify the target device, then an ARP spoofing-based sniffing attempt was executed using Ettercap.

Observed result: When analyzing traffic in Wireshark, the payload content was not readable in plain text. Following the captured streams showed encrypted data.

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -sn 192.168.10.1/24
Starting Nmap 7.98 ( https://nmap.org ) at 2026-02-21 19:07 -0500
Nmap scan report for 192.168.10.1
Host is up (0.0037s latency).
MAC Address: 48:02:2A:42:D2:0B (B-Link Electronic Limited)
Nmap scan report for asma-yl-s-S21.haresnet.local (192.168.10.120)
Host is up (0.031s latency).
MAC Address: BE:4D:DE:95:E9:D1 (Unknown)
Nmap scan report for kali.haresnet.local (192.168.10.150)
Host is up.
Nmap done: 256 IP addresses (3 hosts up) scanned in 2.44 seconds
```

Figure 5.10: Target discovery prior to sniffing attempt.

```
┌──(kali㉿kali)-[~]
└─$ sudo ettercap -T -S -i wlan0 -M arp:remote /192.168.10.1// /192.168.10.120//

ettercap 0.8.4 copyright 2001-2026 Ettercap Development Team

Listening on:
 wlan0 → 1C:BF:CE:13:1F:8C
         192.168.10.150/255.255.255.0
         fe80::3e3a:8970:a636:8de/64

This product includes GeoLite2 Data created by MaxMind, available from https://www.maxmind.com/.
Privileges dropped to EUID 65534 EGID 65534...

  34 plugins
  42 protocol dissectors
  56 ports monitored
38702 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services
Lua: no scripts were specified, not starting up!

Scanning for merged targets (2 hosts)...

* |=========================================================>| 100.00 %

1 hosts added to the hosts list...

ARP poisoning victims:

 GROUP 1 : 192.168.10.1 48:02:2A:42:D2:0B

Starting Unified sniffing...


Text only Interface activated...
Hit 'h' for inline help



Sat Feb 21 19:15:19 2026 [868037]
  :::0 ⟶ ff02::1:ff95:e9d1:0 | SFRC (0)


Sat Feb 21 19:15:19 2026 [875472]
  :::0 ⟶ ff02::16:0 |  (0)


Sat Feb 21 19:15:20 2026 [112235]
  :::0 ⟶ ff02::16:0 |  (0)
```

Figure 5.11: Ettercap ARP spoofing session used for sniffing attempt.

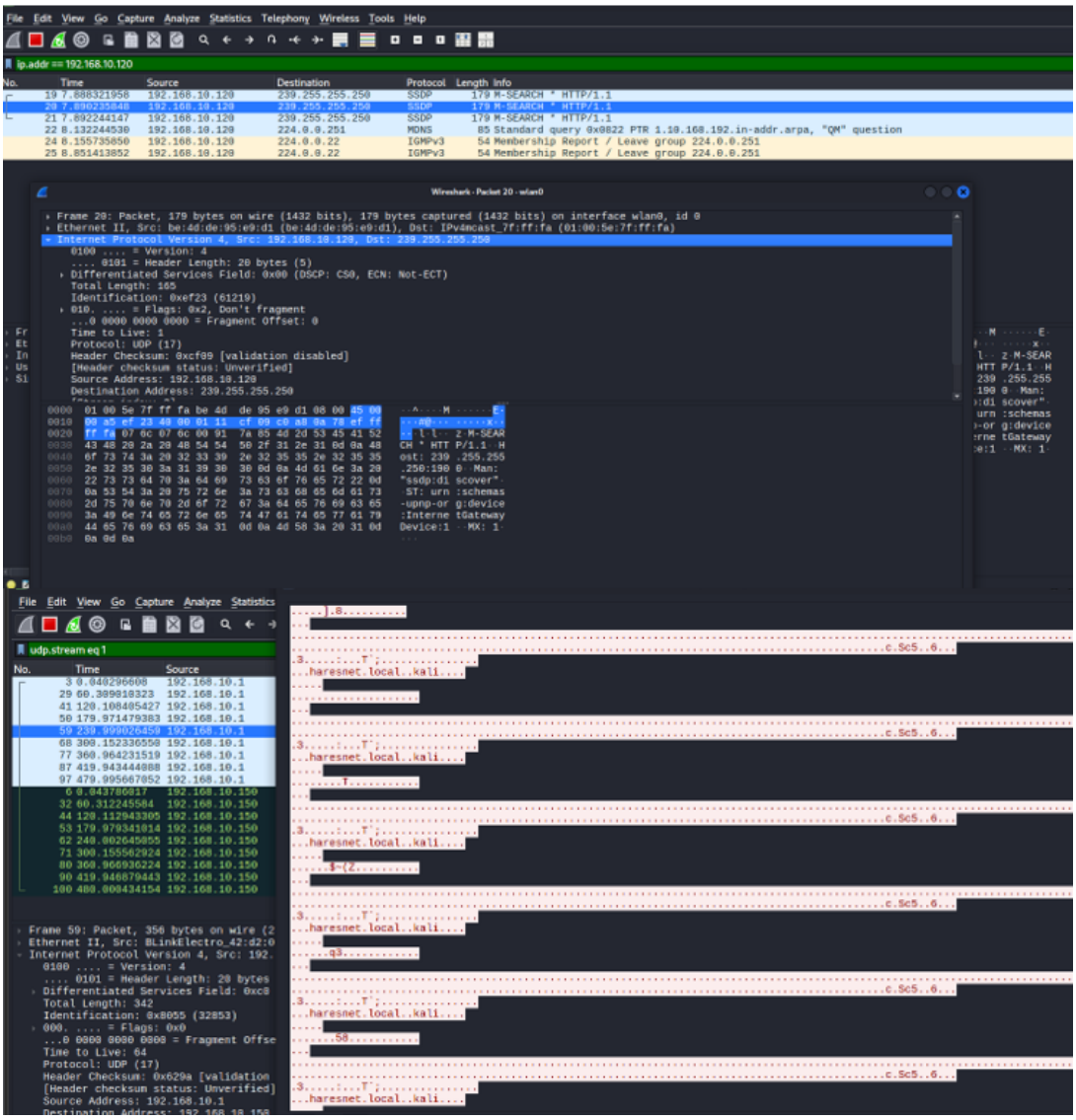

Figure 5.12: Follow Stream output showing encrypted content.

Conclusion: The test indicates that traffic confidentiality is preserved, and the attacker cannot directly read sensitive content from captured packets.

5.6 Security Policies and Administrator Controls

5.6.1 Goal

This section validates user-facing security policies that reduce common router misconfiguration risks and keep the administrator aware of important events.

5.6.2 Credential policy and MFA

Observed behavior: HaresNet avoids common default router credentials and supports changing the administrator username/password. It also supports multi-factor verification via email or notifications, with time-limited codes.

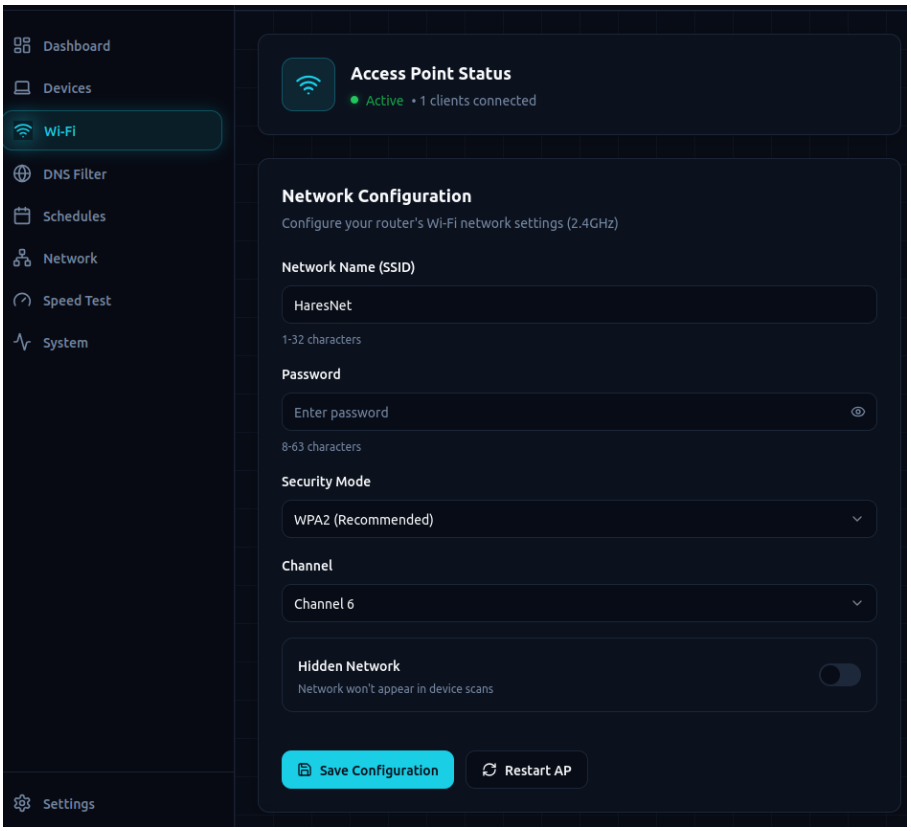

Figure 5.13: Wi-Fi network configuration page in the HaresNet dashboard.

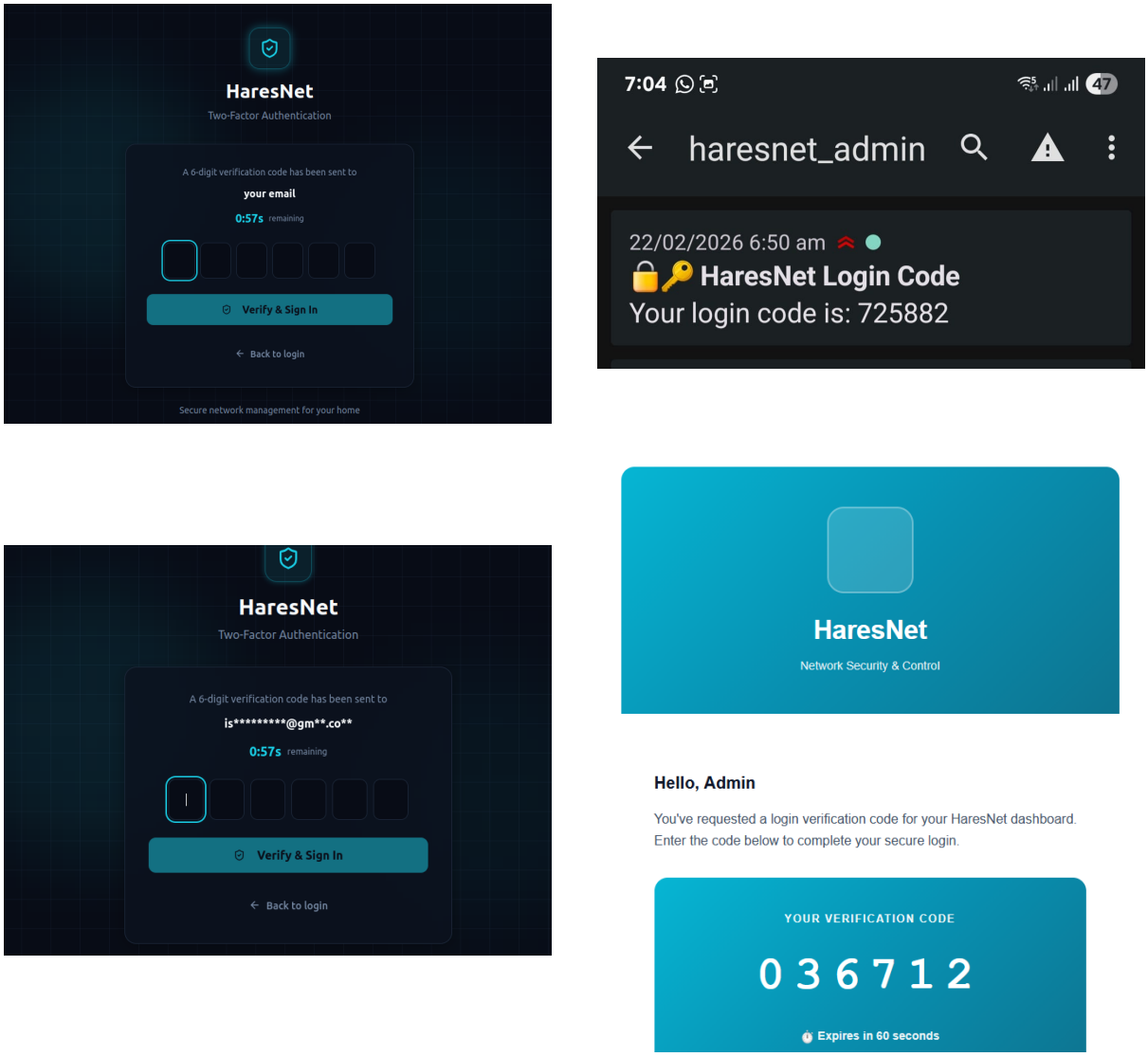

Figure 5.14: MFA verification code delivery example.

5.6.3 Notification system

Observed behavior: The router provides administrator notifications for events such as accessing a blocked website, connecting a new device, or exceeding usage limits.

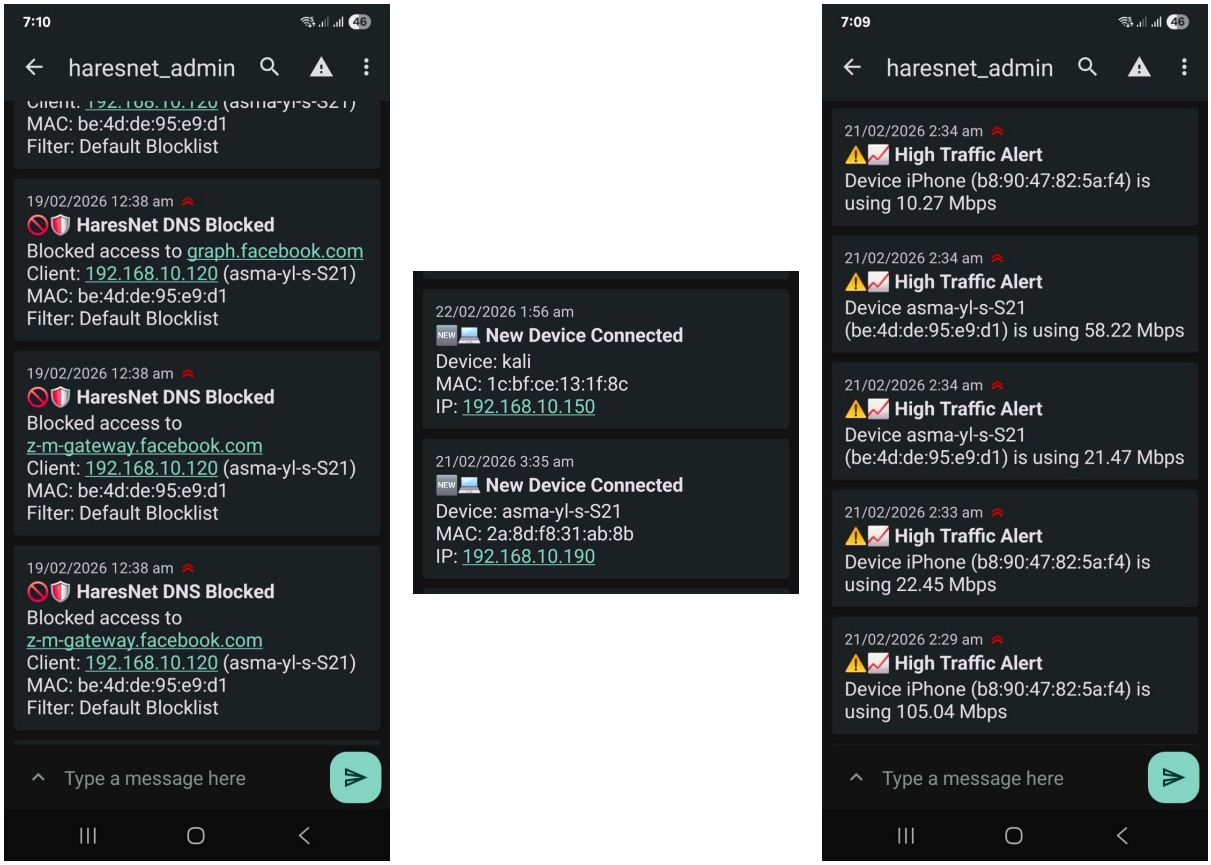

Figure 5.15: Notification examples: blocked website access, new device connection, and usage-limit exceeded.

5.6.4 Parental control mode

Observed behavior: Child mode / parental control policies can be enabled to restrict inappropriate domains and reduce exposure to harmful content.

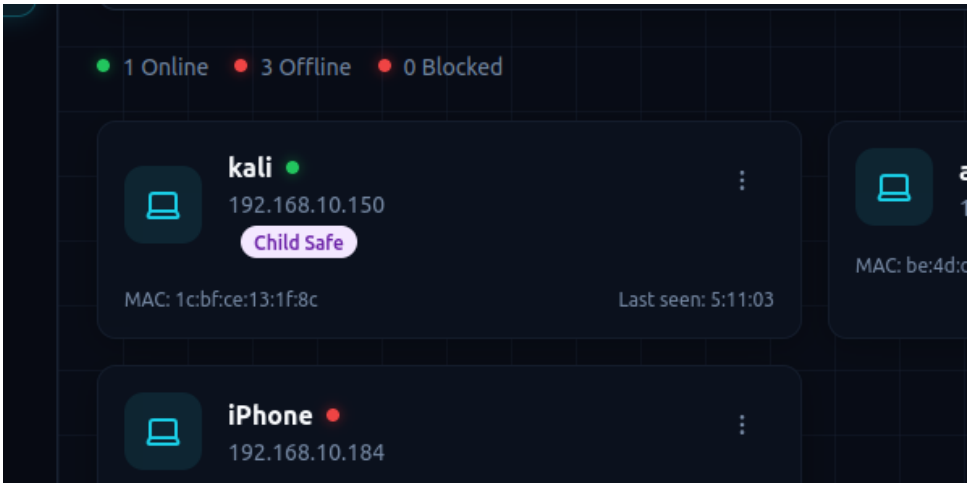

Figure 5.16: Enabling child mode / parental control policies.

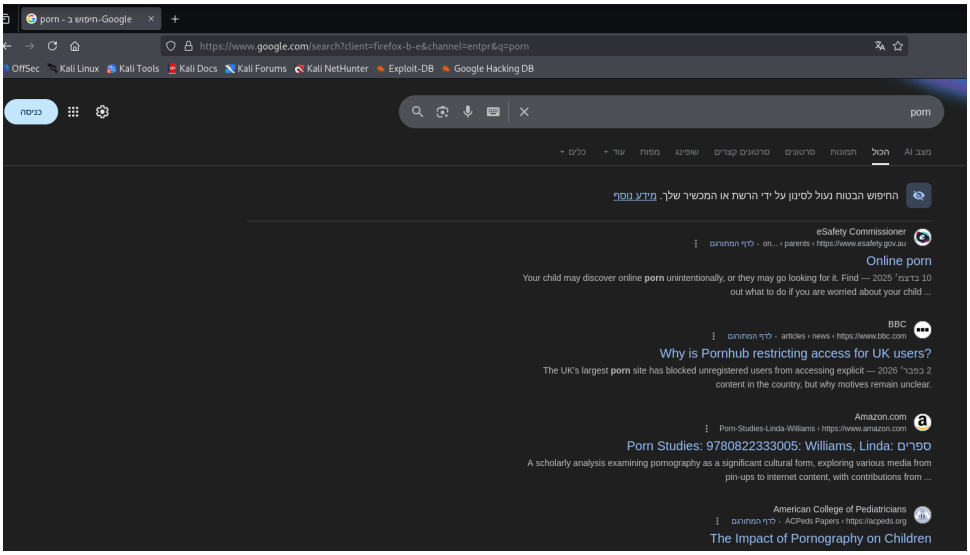

Figure 5.17: Example of blocked content under child mode.

5.7 Summary

The performed tests provide practical evidence that HaresNet security configurations and policies work as intended under the tested conditions. In particular, WPA3 improves resistance against password-guessing attempts, encrypted traffic prevents readable sniffing outputs, and the system remains usable during basic flooding scenarios. In addition to these test cases, HaresNet continuously monitors connected devices and triggers administrator notifications for important events (e.g., blocked access attempts, new device connections, and limit violations), which helps maintain ongoing visibility and operational assurance during real use. These outcomes support the design direction of using strong access control, secure defaults, and administrator visibility as core principles of the platform.

Chapter 6

Conclusion and Future Work

6.1 Preface

This chapter concludes the HaresNet project by summarizing its overall contributions and reflecting on the objectives achieved. It also highlights the main limitations observed during implementation and practical testing. Finally, it presents a set of future improvements that can expand HaresNet into a more complete, capable, and resilient router management platform.

6.2 Project Summary

HaresNet demonstrates that a standard Linux computer can be converted into a manageable WiFi router platform that provides advanced control and visibility. The system brings together access point management, DHCP/DNS services, firewall-based policy enforcement, device-focused rules, and monitoring features into a single dashboard-driven solution. By emphasizing usability and reproducibility, HaresNet can support non-technical administrators in daily operation while also offering a stable foundation for developers who want to extend the platform with new router capabilities.

6.3 Key Contributions

The main contributions of this project are:

- **Software-defined routing on commodity hardware:** WAN over Ethernet and LAN over a USB WiFi access point.
- **Device-centric control:** real-time device discovery with one-click blocking/unblocking and time-based schedules.

- **Service/URL blocking layer:** per-device blocking using firewall IP sets with refresh capability.
- **Monitoring and limits:** real-time traffic, historical views, and configurable daily/hourly limits with notifications.
- **Security-first administration:** OTP-based 2FA, bcrypt password hashing, and token-protected administration endpoints.

6.4 Limitations

Although the system achieved its objectives, the following limitations should be considered:

- **Service identification complexity:** domain-to-IP blocking works well in many scenarios, but large platforms may rotate IP addresses or rely on shared CDN infrastructure, which can reduce blocking precision.
- **Encrypted DNS evolution:** DNS-over-HTTPS and similar approaches can bypass simple DNS filtering. Firewall-based mitigation can help, but the ecosystem continues to evolve.
- **Hardware variability:** WiFi adapters differ in AP stability and driver behavior, which means deployment quality may vary across chipsets.
- **Limited reporting export:** the current design focuses on dashboard visualization; exporting structured reports can be expanded.

6.5 Future Work

The current implementation provides a solid foundation for software-defined router management. Several enhancements can extend HaresNet into a broader platform for network security, policy control, and traffic analysis:

6.5.1 Network Security Enhancements

- **Malware detection and prevention:** Integrate signature-based and behavior-based detection by analyzing network traffic patterns. This may include real-time scanning of HTTP/HTTPS payloads, DNS query analysis for known malicious domains, and integration with threat intelligence feeds to block connections to command-and-control servers.
- **Intrusion Detection System (IDS):** Implement network-based intrusion detection using pattern matching and anomaly detection. Monitor for suspicious activities such as port scanning, brute-force attempts, unusual traffic volumes, and protocol violations.

- **Deep Packet Inspection (DPI):** Add protocol-aware inspection to recognize applications and services beyond basic IP/port matching. This can improve service blocking accuracy and support better bandwidth management based on real application behavior rather than destination addresses alone.

6.5.2 Traffic Analysis and Monitoring

- **Real-time packet analyzer:** Develop an integrated packet capture and analysis interface similar to Wireshark, allowing administrators to inspect live traffic, filter by protocol/source/destination, and export captures for offline analysis. This supports troubleshooting connectivity issues and understanding network behavior.
- **DNS analytics and per-domain visibility:** Parse DNS logs and correlate queries with devices to produce "attempted access" reports. Track requested domains per device, raise alerts for suspicious patterns, and provide detailed DNS query statistics.
- **Application-level traffic classification:** Apply machine learning or heuristic techniques to recognize applications (e.g., video streaming, social media, gaming) even under encryption, enabling more intelligent QoS policies and richer usage reporting.

6.5.3 Policy and Management Features

- **Policy templates and profiles:** Offer ready-made templates (Kids profile, Guest profile, Work profile) and allow applying them to multiple devices with one click, reducing configuration effort and limiting mistakes.
- **Weekly and monthly quotas:** Extend the database model and enforcement logic to support weekly/monthly traffic budgets, time-of-day bandwidth limits, and minute-level burst controls.
- **Geo-blocking capabilities:** Enable traffic allow/deny decisions based on geographic origin using GeoIP databases, which can help restrict access to services in specific regions or reduce exposure to high-risk sources.

6.5.4 Advanced Blocking and Filtering

- **More robust service blocking:** Combine DNS-layer filtering with SNI-based detection for encrypted HTTPS traffic, and optionally support transparent proxy modes to improve accuracy when services use shared infrastructure or encrypted DNS.
- **Content filtering integration:** Integrate web content classification databases to support category-based blocking (adult content, gambling, social media) without requiring manual maintenance of service lists.

6.5.5 Reporting and Administration

- **Exportable audit reports:** Support exporting logs, traffic summaries, and policy snapshots as PDF/CSV for compliance audits, parental reporting, or long-term documentation.
- **Role-based administration:** Add multiple administrator accounts with distinct permission levels (owner vs. operator) and maintain audit logs for administrative actions.
- **Mobile application:** Develop native mobile applications (iOS/Android) to simplify on-the-go monitoring and policy management through push notifications and quick-action controls.

6.6 Closing Remarks

HaresNet establishes a strong base for router and network management on Linux. Its modular design and reproducible evaluation approach make it suitable for both academic use and practical deployment. With the future enhancements outlined in this chapter, the platform can evolve into a more comprehensive and scalable solution for securing and managing home and small-office networks.

References

[1] Jouni Malinen, "hostapd: Ieee 802.11 ap and ieee 802.1x/wpa authenticator." Project documentation, 2026.

[2] M. H. Iyad, *Joint Crypto-Compression Based on Selective Encryption for WMSNs*. PhD thesis, Thesis submitted in partial fulfillment of the requirements of the degree ….

[3] Simon Kelley, "dnsmasq documentation." Official documentation, 2026.

[4] netfilter project, "nftables wiki." Project documentation, 2026.

[5] Pallets Projects, "Flask documentation." Official documentation, 2026.

[6] SQLAlchemy Authors, "Sqlalchemy documentation." Official documentation, 2026.

[7] SQLite Project, "Sqlite documentation." Official documentation, 2026.

[8] Vimalloc, "Flask-jwt-extended documentation." Project documentation, 2026.

[9] PyCA, "bcrypt (python package)." PyPI documentation, 2026.

[10] OWASP Foundation, "Password storage cheat sheet." OWASP Cheat Sheet Series, 2026.

[11] OpenWrt Project, "Openwrt documentation." Official documentation, 2026.

[12] Netgate, "pfsense documentation." Official documentation, 2026.

[13] Deciso B.V., "Opnsense documentation." Official documentation, 2026.

[14] Pi-hole Project, "Pi-hole documentation." Official documentation, 2026.

[15] AdGuard, "Adguard home documentation." Official documentation, 2026.

[16] P. Weidenbach and J. vom Dorp, "Home router security report 2020," tech. rep., Fraunhofer FKIE, 6 2020.

[17] J. vom Dorp and R. Helmke, "Home router security report 2022," tech. rep., Fraunhofer FKIE, 11 2022.

[18] J. Ye, X. de Carné de Carnavalet, L. Zhao, M. Zhang, L. Wu, and W. Zhang, "Exposed by default: A security analysis of home router default settings and beyond," *IEEE Internet of Things Journal*, vol. 12, no. 2, pp. 1182–1199, 2025.

[19] Miguel Grinberg, “Flask-socketio documentation.” Project documentation, 2026.

[20] InfluxData, “Influxdb 2 documentation.” Official documentation, 2026.

[21] speedtest-cli Contributors, “speedtest-cli (python package).” PyPI documentation, 2026.

[22] Docker, Inc., “Docker documentation.” Official documentation, 2026.

[23] Docker, Inc., “Docker compose documentation.” Official documentation, 2026.