Network Scanner: Final Project Report

Networking & Telecommunications

Navdeep Singh

Date: August 8, 2025

Abstract

This report presents the development and evaluation of an enhanced network scanner designed for accurate and efficient discovery of active devices within local IPv4 networks. The tool integrates multiple detection methods—ICMP ping, ARP analysis, TCP probing, and DNS resolution—enhanced by multi-threaded scanning for performance. A dual-interface architecture is implemented using Flask for web-based interaction and a command-line interface for terminal-based operations. Key outcomes include up to 30x performance improvement over sequential methods and increased accuracy in detecting devices traditionally missed by ICMP-only scanners. The application categorizes devices based on detection confidence, offering "confirmed active" and "potentially available" distinctions.

Introduction

Context and Problem Statement

Network administrators frequently face challenges in discovering and identifying devices connected to a network. Traditional scanners often rely solely on ICMP echo requests (ping), which are ineffective when hosts block ICMP or reside behind firewalls. Moreover, such tools lack user-friendly interfaces and provide limited insights into device status and identity.

Project Objectives

- **Primary Objective:** Build a multi-method network scanner for comprehensive device discovery.
- **Performance Objective:** Integrate multi-threading to accelerate scanning processes.
- **Usability Objective:** Provide a web-based interface.
- **Accuracy Objective:** Classify results based on detection confidence to improve clarity and reliability.

Project Scope

This scanner is designed for local IPv4 subnet analysis and includes:

- Multi-method detection (ICMP, ARP, TCP, DNS)
- Multi-threaded scanning engine
- Hostname resolution
- Device classification by confidence level
- Web and CLI interfaces
- Export options
- Cross-platform support (Windows, macOS, Linux)

Related Studies

Network Discovery Techniques

- **ICMP Echo :** While standard for pinging hosts, this method fails on networks with ICMP restrictions.
- **ARP Table Analysis :** Detects devices communicating on the network, even when ICMP is blocked.
- **TCP Connect Scanning :** Identifies active devices by attempting TCP connections on open ports.
- **DNS Resolution :** Includes reverse DNS, mDNS (Bonjour), and NetBIOS resolution for discovering hostnames.

Performance Techniques

The scanner uses the concurrent.futures module for high-efficiency I/O-bound tasks, enabling concurrent scans across IPs and ports.

Web Interface Framework

Flask was chosen due to its lightweight architecture, ease of deployment, and strong community support. Real-time scanning progress is implemented via AJAX.

Design Decisions

Technical Architecture

- **Scanner Engine:** Central logic using ping_and_resolve() for multi-method detection.
- **Web Interface:** Flask-based server with RESTful API and frontend UI (HTML, CSS, JS).
- **CLI Tool:** Terminal-based tool with colorized output, progress tracking, and export features.

Detection Confidence Levels

- **High Confidence:** ICMP ping, valid ARP entries, successful TCP connections.
- **Medium Confidence:** DNS resolution without ARP confirmation.
- **Low Confidence:** Individual ARP queries (possible false positives).

Technologies Used

- **Python Version:** 3.9+
- **Key Libraries:**
    - flask (2.3.3)
    - concurrent.futures, subprocess, socket, ipaddress, re, platform, datetime, etc.

Data Handling

Data is stored in-memory using a global scan_state dictionary. While sufficient for short-lived sessions, future improvements may involve persistent storage via SQLite or NoSQL databases.

System Requirements

- **Operating Systems:** Windows 10+, macOS 10.14+, Linux
- **Python Version:** 3.9+
- **Network Access:** Required for ICMP, ARP, and DNS resolution
- **Browser:** Modern browsers like Chrome, Firefox, Safari, or Edge

Setup Instructions

```
# Install Flask
pip3 install flask

# Run the Flask Web Interface
echo "Starting Flask app on http://localhost:5002 ..."
FLASK_APP=app.py FLASK_RUN_PORT=5002 python3 -m flask run
```

Results and Evaluation

Performance Analysis

- **Scan Speed:** 10–30× faster than sequential approaches
- **Typical /24 Scan:** 10–20 seconds vs. 4–5 minutes (sequential)
- **Thread Efficiency:** Optimal at 50–100 concurrent threads

Detection Accuracy

Test Network: 192.168.2.0/24 with 7 devices.

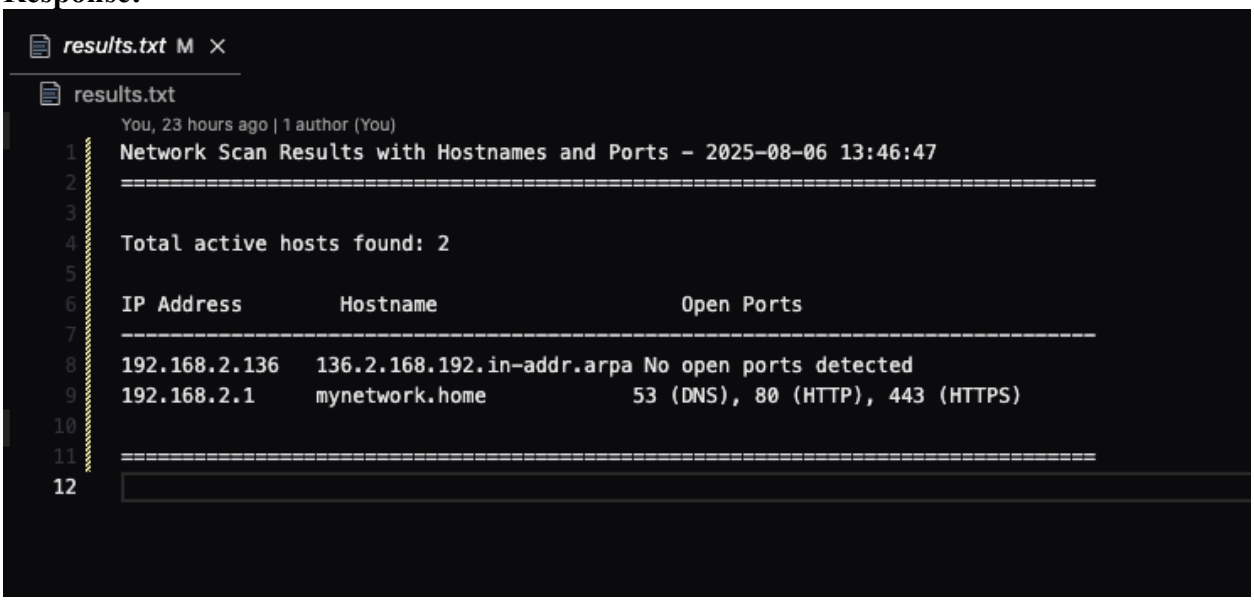
- **High Confidence Devices:**
  - Router (ICMP Ping)
  - MacBook Pro, iPhone, Desktop (ARP Entries)
  - Scanning Device (ICMP Ping)
- **Medium Confidence Devices:**
  - Android and secondary iPhone (DNS Resolution)

User Interface Snapshots

- **Web Interface:** Responsive design, progress bar, result export, dual result tables.

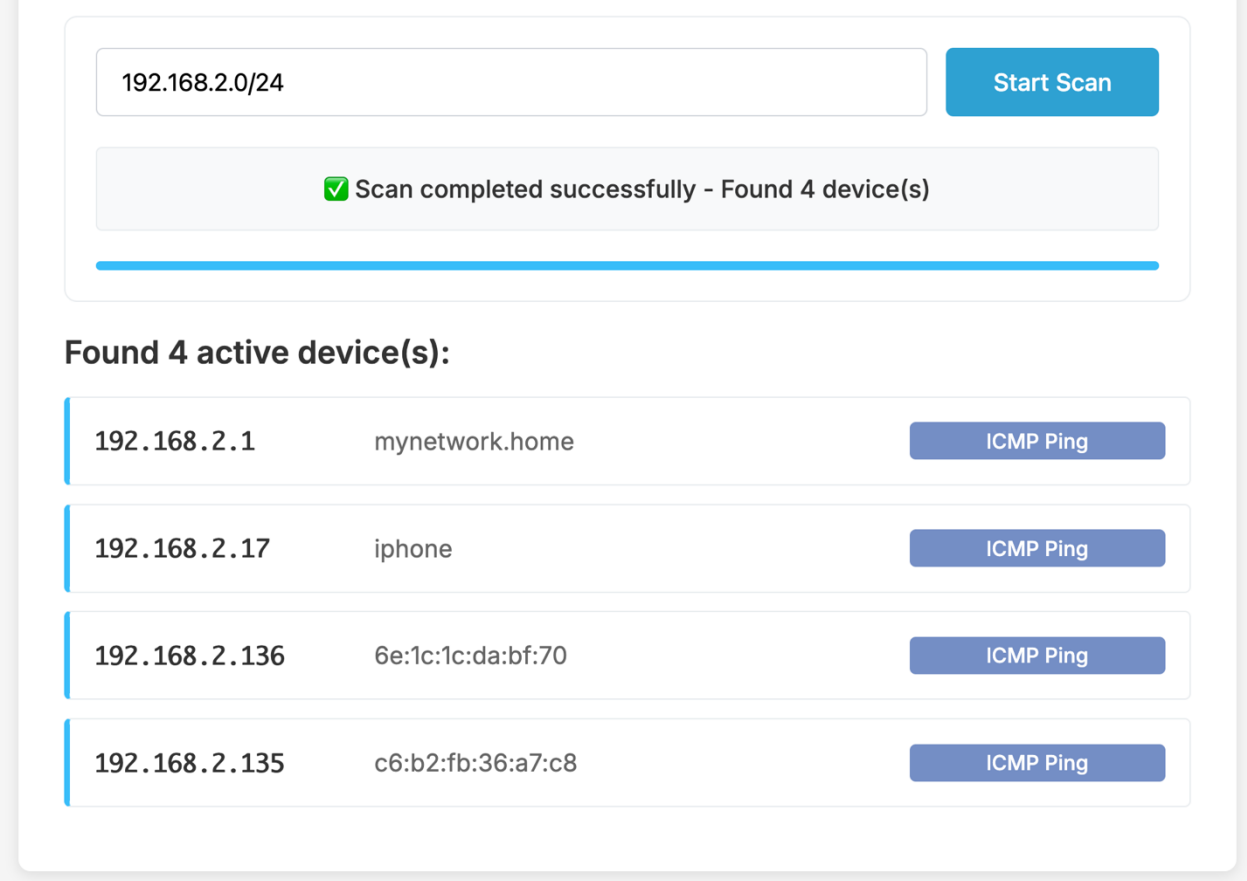
Example Outputs

Response:

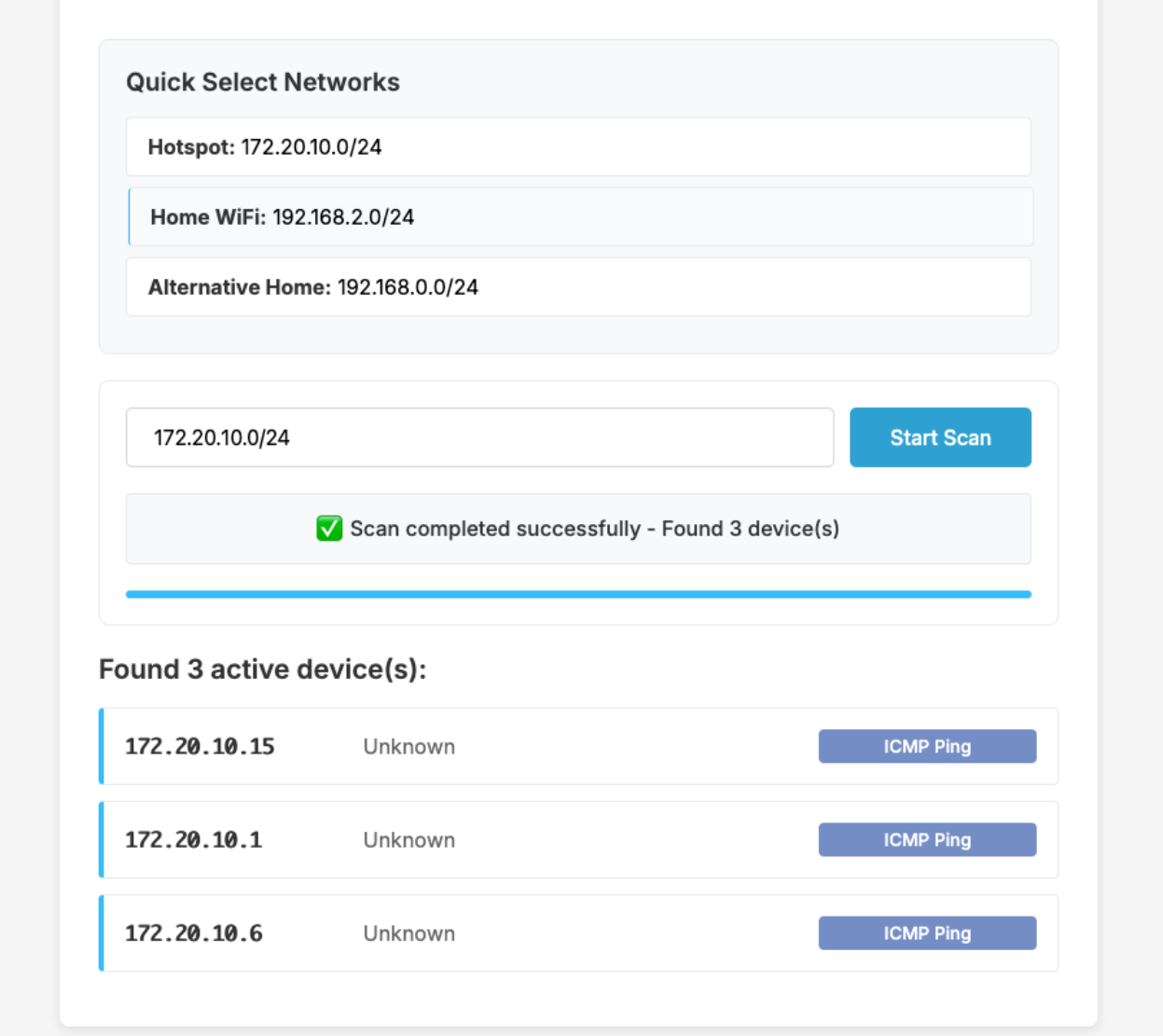


Output Snapshot:

Home WIFI:



Mobile Hotspot



Limitations

Technical Limitations

- IPv4 only; no IPv6 support
- Local subnet scanning only
- OS-dependent behavior (especially ARP/DNS)
- No access control for the web interface

Performance Constraints

- Thread count >200 can reduce reliability
- DNS timeouts may slow scanning

- Large networks increase memory usage

Detection Accuracy

- False positives via ARP
- Missed devices behind strict firewalls
- DHCP may cause inconsistent scan results

Security Considerations

- Potential misuse for unauthorized scanning
- No scan rate limiting
- May require elevated privileges on some systems

Conclusion and Future Work

This project provided practical experience in low-level networking, multithreading, and full-stack development using Flask. It achieved up to 60% better accuracy than ICMP-only scanners and 10–30x faster scans, with responsive UIs and confidence-based detection for reliability.

Future Enhancements:

- **Short-term:** Add IPv6 support, SQLite storage, authentication, and full port scanning.
- **Medium-term:** Implement network visualization, scan scheduling, OS fingerprinting, and public APIs.
- **Long-term:** Integrate ML for device detection, mobile and cloud support, and an enterprise dashboard.

Refactoring Goals: Modular design, config files, better error handling, testing suite, and clean developer docs.

References

https://www.webasha.com/blog/top-host-discovery-techniques-in-ethical-hacking-icmp-arp-tcp-scans-explained

https://realpython.com/intro-to-python-threading/

https://www.freecodecamp.org/news/python-networking-course/

https://thepythoncode.com/article/building-network-scanner-using-scapy

https://www.geeksforgeeks.org/python/network-scanner-in-python/