WiFi Hotspot Detector

Android Application for Network Security Monitoring

Made By: AZZOUZ M'hamed Alaa Eddine

Key Features

wifi
Real-Time Scanning
Continuous network detection

exclamation-triangle
Threat Detection
Identify rogue hotspots

map-marker-alt
Location Tracking
Pinpoint signal source

mobile-alt
Haptic Feedback
Vibration-guided navigation

Technical Documentation
Version 1.0

Designed for authorized security monitoring in controlled environments

December 8, 2025

Contents

1 Executive Summary

This document presents a comprehensive technical overview of the WiFi Hotspot Detector, an Android application designed to identify and locate unauthorized WiFi networks in controlled environments such as classrooms. The application leverages Android's WiFi scanning capabilities, signal strength measurement (RSSI), and haptic feedback to enable users to physically locate rogue access points and mobile hotspots that may be used for unauthorized communication.

1.1 Key Features

- Real-time WiFi network scanning with automatic refresh
- Suspicious hotspot detection based on naming patterns and signal strength
- Visual signal strength representation through color-coded circles
- Distance estimation based on RSSI measurements
- Continuous haptic feedback proportional to signal proximity
- User-friendly interface with custom list views and tracking display

2 Purpose and Use Case

2.1 Primary Objective

The WiFi Hotspot Detector addresses a specific security concern in educational and professional environments: the detection of unauthorized wireless networks that may be used to circumvent communication restrictions or facilitate cheating during examinations [1].

2.2 Target Environment

Classroom Monitoring Scenario

During examinations, students may attempt to use personal mobile devices configured as WiFi hotspots to share information. Traditional visual inspection is insufficient, as devices can be concealed. This application enables proctors to:

1. Detect hidden and visible WiFi networks in real-time
2. Identify suspicious networks based on signal strength and naming patterns
3. Physically locate the source through signal triangulation using haptic feedback

2.3 Legal and Ethical Considerations

Important Notice: This application is intended for authorized monitoring in controlled environments. Users must:

- Obtain proper authorization before deployment
- Comply with local privacy and surveillance regulations
- Use the application only in environments where monitoring is legally permitted
- Respect individual privacy rights and data protection laws

3 Technical Architecture

3.1 System Components

The application consists of three primary components organized in a Model-View-Controller (MVC) architecture pattern:

1. **MainActivity**: Network scanning and list display
2. **TrackerActivity**: Signal tracking and proximity detection
3. **WifiAdapter**: Custom list adapter for network visualization

3.2 Technology Stack

Component	Technology
Platform	Android 5.0+ (API Level 21+)
Language	Java
UI Framework	Android XML Layouts
Permissions	Location, WiFi State, Vibration
Hardware	WiFi radio, Vibrator (haptic engine)

Table 1: Technology Stack Overview

4 Android Permissions and Requirements

4.1 Required Permissions

The application requires several Android permissions to function properly [2]:

```
<!-- WiFi + network state -->
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
    />

<!-- Location (required for WiFi scans on Android < 13) -->
<uses-permission android:name="android.permission.
   ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
    />
<uses-permission android:name="android.permission.
   ACCESS_BACKGROUND_LOCATION" />
```

```
10
11 <!-- Nearby WiFi devices (Android 13+) -->
12 <uses-permission
13     android:name="android.permission.NEARBY_WIFI_DEVICES"
14     android:usesPermissionFlags="neverForLocation" />
15
16 <!-- Vibration -->
17 <uses-permission android:name="android.permission.VIBRATE" />
```

Listing 1: AndroidManifest.xml Permissions

4.2 Permission Rationale

Location Permissions: Android requires location permissions to access WiFi scan results due to the potential for location inference from nearby network BSSIDs [1]. The application does not collect, store, or transmit location data.

WiFi State Permissions: Required to enable WiFi scanning and retrieve network information including SSID, BSSID, and signal strength (RSSI).

Vibration Permission: Enables haptic feedback for proximity indication.

5 Component Design and Implementation

5.1 MainActivity: Network Scanner

The MainActivity serves as the primary interface for network discovery and selection.

5.1.1 Core Functionality

```
1  public class MainActivity extends AppCompatActivity {
2      private WifiManager wifiManager;
3      private WifiAdapter adapter;
4      private List<ScanResult> lastScanResults;
5      private List<ScanResult> suspiciousNetworks;
6
7      @Override
8      protected void onCreate(Bundle savedInstanceState) {
9          super.onCreate(savedInstanceState);
10         setContentView(R.layout.activity_main);
11
12         // Initialize WiFi manager
13         wifiManager = (WifiManager) getApplicationContext()
14                 .getSystemService(Context.WIFI_SERVICE);
15
16         // Setup auto-refresh mechanism
17         autoRefreshHandler.post(autoRefreshRunnable);
18
19         // Request necessary permissions
20         checkAndRequestPermissions();
21     }
22
23     private void performScan() {
24         boolean started = wifiManager.startScan();
25         if (started) {
```

```
            scanCount++;
            Log.i(TAG, "Scan initiated successfully");
        } else {
            Log.w(TAG, "Scan throttled, using cached results");
        }
    }
}
```

Listing 2: MainActivity Key Methods

5.1.2 WiFi Scan Throttling

Android implements scan throttling to preserve battery life [3]. The application handles this through:

- **Foreground Limit:** 4 scans per 2-minute window
- **Background Limit:** 1 scan per 30 minutes
- **Mitigation Strategy:** Continuous reading of cached scan results updated by system background scans

```
private void performScan() {
    long now = System.currentTimeMillis();

    if (now - lastScanTime > SCAN_WINDOW_MS) {
        scanCount = 0;
    }

    if (scanCount >= 4) {
        long waitTime = (SCAN_WINDOW_MS - (now - lastScanTime)) / 1000;
        Log.w(TAG, "Throttled. Wait " + waitTime + " seconds");
        showScanResults(); // Use cached results
        return;
    }

    boolean started = wifiManager.startScan();
    if (started) {
        scanCount++;
    }
}
```

Listing 3: Scan Throttling Management

5.1.3 Suspicious Network Detection

The application employs heuristic analysis to identify potentially unauthorized networks:

```
private void identifySuspiciousNetworks() {
    for (ScanResult sr : lastScanResults) {
        boolean isSuspicious = false;
        String ssid = (sr.SSID == null || sr.SSID.isEmpty())
                ? "<hidden>" : sr.SSID;

        // Strong signal indicates close proximity (< 5 meters)
        if (sr.level > -50) {
```

```
            isSuspicious = true;
        }

        // Pattern matching for mobile hotspot names
        String ssidLower = ssid.toLowerCase();
        if (ssidLower.contains("android") ||
            ssidLower.contains("iphone") ||
            ssidLower.contains("hotspot") ||
            ssidLower.contains("mobile") ||
            // Manufacturer-specific patterns
            ssidLower.contains("sm-") ||      // Samsung
            ssidLower.contains("pixel") ||     // Google
            ssidLower.contains("xiaomi") ||
            ssidLower.contains("oneplus") ||
            (ssid.equals("<hidden>") && sr.level > -60)) {
            isSuspicious = true;
        }

        if (isSuspicious) {
            suspiciousNetworks.add(sr);
        }
    }
}
```

Listing 4: Suspicious Network Detection Algorithm

5.2 TrackerActivity: Signal Tracking and Localization

The TrackerActivity provides real-time signal monitoring and proximity feedback.

5.2.1 RSSI Measurement and Distance Estimation

Signal strength (RSSI) is measured in dBm and used to estimate distance through the free-space path loss formula [4]:

$$d = 10^{\frac{RSSI_{1m} - RSSI}{10 \cdot n}} \tag{1}$$

Where:

- d = estimated distance in meters
- $RSSI_{1m}$ = reference RSSI at 1 meter (typically -40 dBm)
- $RSSI$ = measured signal strength
- n = path loss exponent (2.5 for indoor environments)

```
private double calculateDistance(int rssi) {
    int rssiAt1m = -40;           // Reference RSSI at 1 meter
    double pathLossExponent = 2.5; // Indoor environment

    double distance = Math.pow(10,
        (rssiAt1m - rssi) / (10.0 * pathLossExponent));

    return distance;
}
```

Listing 5: Distance Calculation Implementation

5.2.2 Continuous Haptic Feedback

The application uses Android's VibrationEffect API to provide continuous vibration with intensity proportional to signal strength:

```
private void applyVibration(int rssi) {
    if (vibrator == null || !vibrator.hasVibrator()) {
        return;
    }

    int minRssi = -90;  // Very weak signal
    int maxRssi = -40;  // Very strong signal
    int clamped = Math.max(minRssi, Math.min(maxRssi, rssi));

    // Normalize RSSI to 0-1 range
    float normalized = (float) (clamped - minRssi) /
                       (float) (maxRssi - minRssi);

    // Map to vibration amplitude (0-255)
    int amplitude = (int) (normalized * 255);

    if (amplitude <= 10) {
        vibrator.cancel();
        return;
    }

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        // Create continuous waveform
        VibrationEffect effect = VibrationEffect.createWaveform(
            new long[]{0, 10000},       // Immediate start, 10s duration
            new int[]{0, amplitude},    // Off, then continuous
            1                           // Repeat from index 1
        );
        vibrator.vibrate(effect);
    } else {
        vibrator.vibrate(10000);
    }
}
```

Listing 6: Continuous Vibration Implementation

5.2.3 Visual Feedback System

The tracker interface uses concentric circles with color gradients to represent signal quality:

RSSI Range	Quality	Color
$>$ -50 dBm	Excellent ($<$ 2m)	Green (#4CAF50)
-60 to -50 dBm	Good (2-5m)	Light Green (#8BC34A)
-70 to -60 dBm	Fair (5-10m)	Orange (#FF9800)
-80 to -70 dBm	Weak (10-20m)	Deep Orange (#FF5722)
$<$ -80 dBm	Very Weak ($>$ 20m)	Red (#F44336)

Table 2: Signal Quality Color Coding

```
private void updateCircleColor(int rssi) {
    GradientDrawable drawable =
        (GradientDrawable) circleInner.getBackground();

    String fillColor;
    String strokeColor;

    if (rssi > -50) {
        fillColor = "#4CAF50";
        strokeColor = "#2E7D32";
    } else if (rssi > -60) {
        fillColor = "#8BC34A";
        strokeColor = "#558B2F";
    } else if (rssi > -70) {
        fillColor = "#FF9800";
        strokeColor = "#E65100";
    } else if (rssi > -80) {
        fillColor = "#FF5722";
        strokeColor = "#BF360C";
    } else {
        fillColor = "#F44336";
        strokeColor = "#B71C1C";
    }

    drawable.setColor(Color.parseColor(fillColor));
    drawable.setStroke(6, Color.parseColor(strokeColor));
}
```

Listing 7: Dynamic Circle Color Update

5.3 WifiAdapter: Custom List Display

The WifiAdapter extends ArrayAdapter to provide custom visualization of network data:

```
public class WifiAdapter extends ArrayAdapter<ScanResult> {
    @NonNull
    @Override
    public View getView(int position, View convertView, ViewGroup
   parent) {
        if (convertView == null) {
            convertView = LayoutInflater.from(context)
                .inflate(R.layout.wifi_list_item, parent, false);
        }

        ScanResult sr = networks.get(position);

        TextView txtSsid = convertView.findViewById(R.id.txtSsid);
        TextView txtBssid = convertView.findViewById(R.id.txtBssid);
        TextView txtSignalStrength =
            convertView.findViewById(R.id.txtSignalStrength);

        String ssid = (sr.SSID == null || sr.SSID.isEmpty())
            ? "<Hidden Network>" : sr.SSID;
        txtSsid.setText(ssid);
        txtBssid.setText(sr.BSSID);
        txtSignalStrength.setText(String.valueOf(sr.level));

        // Color code signal strength
```

```
24          if (sr.level > -50) {
25              txtSignalStrength.setTextColor(
26                  Color.parseColor("#4CAF50"));
27          } else if (sr.level > -60) {
28              txtSignalStrength.setTextColor(
29                  Color.parseColor("#8BC34A"));
30          } else if (sr.level > -70) {
31              txtSignalStrength.setTextColor(
32                  Color.parseColor("#FF9800"));
33          } else {
34              txtSignalStrength.setTextColor(
35                  Color.parseColor("#F44336"));
36          }
37
38          // Mark suspicious networks
39          boolean isSuspicious = false;
40          for (ScanResult susp : suspiciousNetworks) {
41              if (susp.BSSID.equals(sr.BSSID)) {
42                  isSuspicious = true;
43                  break;
44              }
45          }
46
47          if (isSuspicious) {
48              txtWarningIcon.setVisibility(View.VISIBLE);
49              txtSuspiciousLabel.setVisibility(View.VISIBLE);
50          }
51
52          return convertView;
53      }
54  }
```

Listing 8: WifiAdapter Custom List Item

6 User Interface Design

6.1 Material Design Principles

The application follows Google's Material Design guidelines for Android applications, featuring:

- **Color Scheme:** Primary blue (#2196F3), accent green (#4CAF50)
- **Typography:** Roboto font family with hierarchical sizing
- **Elevation:** Shadow effects for visual depth
- **Touch Targets:** Minimum 48dp for interactive elements

6.2 Layout Hierarchy

6.2.1 MainActivity Layout

```
<LinearLayout>
    <!-- Header with title and network count -->
    <LinearLayout android:background="#2196F3">
        <TextView text="WiFi Hotspot Detector" />
        <TextView android:id="@+id/txtNetworkCount" />
    </LinearLayout>

    <!-- Scan button -->
    <Button android:id="@+id/btnScan"
            android:background="#4CAF50" />

    <!-- Status text -->
    <TextView android:id="@+id/txtStatus" />

    <!-- Networks list -->
    <ListView android:id="@+id/listViewWifi" />

    <!-- Empty state message -->
    <TextView android:id="@+id/txtEmptyState"
              android:visibility="gone" />
</LinearLayout>
```

Listing 9: activity_main.xml Structure

6.2.2 TrackerActivity Layout

```
<LinearLayout>
    <!-- Header -->
    <LinearLayout android:background="#2196F3">
        <TextView android:id="@+id/txtTrackerTitle" />
        <TextView android:id="@+id/txtTrackerBssid" />
    </LinearLayout>

    <!-- Signal visualization -->
    <RelativeLayout>
        <View android:id="@+id/circleOuter" />
        <View android:id="@+id/circleMiddle" />
        <View android:id="@+id/circleInner" />

        <LinearLayout>
            <TextView android:id="@+id/txtRssiValue" />
            <TextView android:id="@+id/txtDistance" />
            <TextView android:id="@+id/txtSignalQuality" />
        </LinearLayout>
    </RelativeLayout>

    <!-- Controls -->
    <LinearLayout>
        <SwitchCompat android:id="@+id/switchVibration" />
        <Button android:id="@+id/btnRefresh" />
        <Button android:id="@+id/btnBack" />
    </LinearLayout>
</LinearLayout>
```

Listing 10: activity_tracker.xml Structure

7 Operational Workflow

7.1 Application Startup Sequence

1. Application launches and checks for required permissions
2. If permissions missing, displays explanation dialogs
3. Requests location permission (foreground)
4. Requests background location permission (optional)
5. Verifies WiFi and location services are enabled
6. Initializes WifiManager and registers broadcast receiver
7. Starts auto-refresh loop (5-second interval)
8. Displays cached scan results if available

7.2 Network Detection and Tracking Workflow

1. User taps "SCAN FOR NETWORKS" button
2. Application triggers WiFi scan via WifiManager
3. Scan results processed and filtered for suspicious patterns
4. Networks displayed in sorted list (strongest signal first)
5. User selects target network from list
6. TrackerActivity launches with BSSID parameter
7. Continuous RSSI monitoring begins (500ms interval)
8. Visual and haptic feedback updates in real-time
9. User follows vibration intensity to locate source
10. Distance estimation guides physical approach

8 Technical Challenges and Solutions

8.1 WiFi Scan Throttling

Challenge: Android's aggressive scan throttling limits foreground apps to 4 scans per 2 minutes [3].

Solution:

- Implement auto-refresh using cached scan results
- Leverage system background scans
- Provide user instructions for disabling throttling in Developer Options
- Graceful degradation when scans are blocked

8.2 Location Permission Requirements

Challenge: WiFi scanning requires location permissions on Android 6.0+, which users may perceive as privacy-invasive [2].

Solution:

- Display clear permission rationale dialogs
- Explain that location data is not collected
- Provide step-by-step permission instructions
- Handle permission denial gracefully

8.3 RSSI Measurement Accuracy

Challenge: WiFi signal strength varies due to multipath propagation, interference, and environmental factors [4].

Solution:

- Use conservative path loss exponent (2.5)
- Display approximate distance ranges
- Update measurements at 500ms intervals
- Combine RSSI with haptic feedback for improved localization

9 Testing and Validation

9.1 Test Environment

- **Device:** Android 10+ smartphones
- **Environment:** Classroom (30m $\times$ 20m)
- **Target:** Mobile hotspot (hidden SSID)
- **Interference:** 5-10 legitimate WiFi networks

9.2 Test Scenarios

1. **Network Detection:** Verify all nearby networks appear in scan results
2. **Suspicious Flagging:** Confirm hotspots identified correctly
3. **Signal Tracking:** Validate RSSI updates and distance estimates
4. **Haptic Feedback:** Test vibration intensity correlation with proximity
5. **Permission Handling:** Verify graceful behavior when permissions denied
6. **Throttling:** Confirm cached results used when scans blocked

Metric	Result
Scan Interval	5 seconds (cached)
RSSI Update Rate	500 ms
Detection Range	1-30 meters
Location Accuracy	±2-5 meters
Battery Impact	Moderate (10-15%/hour)

Table 3: Application Performance Metrics

9.3 Performance Metrics

10 Deployment and Configuration

10.1 Installation Requirements

- Android 5.0 (Lollipop) or higher
- WiFi hardware support
- Vibration motor (optional but recommended)
- 10 MB storage space

10.2 Initial Configuration

Step 1: Enable Developer Options

1. Settings → About Phone
2. Tap "Build Number" 7 times
3. Return to Settings → Developer Options

Step 2: Disable WiFi Scan Throttling

1. Developer Options → Networking
2. Toggle "WiFi scan throttling" OFF

Step 3: Grant Permissions

1. Install and launch application
2. Grant Location permission (Allow all the time)
3. Grant Nearby WiFi Devices permission (Android 13+)
4. Enable Location Services in device settings

11 Future Enhancements

11.1 Planned Features

1. **Multi-Point Triangulation:** Record RSSI from multiple positions to calculate precise location using trilateration algorithms
2. **Network Logging:** Store scan history with timestamps for audit trails
3. **Alerting System:** Automatic notifications when suspicious networks detected
4. **Heatmap Visualization:** Generate signal strength heatmaps of scanned areas
5. **Bluetooth Detection:** Expand to detect Bluetooth-based communication
6. **Network Analysis:** Display connected devices, bandwidth usage, encryption type

11.2 Technical Improvements

1. Implement Kalman filtering for RSSI smoothing
2. Add machine learning classification for hotspot detection
3. Optimize battery consumption through adaptive scan intervals
4. Support for 5GHz and 6GHz WiFi bands
5. Export scan data in CSV/JSON formats

12 Conclusion

The WiFi Hotspot Detector provides an effective solution for identifying and locating unauthorized wireless networks in controlled environments. By combining WiFi scanning, signal strength analysis, and haptic feedback, the application enables physical localization of rogue access points without requiring direct connection or network intrusion.

Key achievements include:

- Real-time detection of hidden and visible networks
- Intelligent suspicious network identification
- Intuitive proximity-based localization through continuous vibration
- User-friendly interface with clear visual feedback
- Robust permission handling and graceful degradation

The application demonstrates practical use of Android's networking APIs and serves as a valuable tool for maintaining communication security in examination and restricted environments. Future enhancements will focus on improved accuracy through multi-point triangulation and expanded detection capabilities.

13 Complete Source Code

The complete code is provided in the github repository:.
https://github.com/AladinAzz/WIFI-Finder

References

[1] Android Developers. *Wi-Fi Scanning Overview*. https://developer.android.com/develop/connectivity/wifi/wifi-scan, 2025.

[2] Android Developers. *Request Permission to Access Nearby Wi-Fi Devices*. https://developer.android.com/develop/connectivity/wifi/wifi-permissions, 2025.

[3] NetSpot. *How to Disable Wi-Fi Throttling on Android 10+*. https://www.netspotapp.com/help/how-to-disable-wi-fi-throttling-on-android-10/, 2025.

[4] Blues Wireless. *Comparing GPS, Cell, and WiFi Triangulation for Location Tracking*. https://blues.com/blog/beyond-gps-leveraging-cell-wifi-triangulation-for-precise-iot-location-tracking/, 2025.