Zrób to sam

Mapa zasięgu Wi-Fi na ESP32.

- DIY: mapa zasięgu Wi-Fi w magazynie na ESP32

DIY: mapa zasięgu Wi-Fi w magazynie na ESP32

Poziom: średnio

Dotyczy: ESP32 (płytka z Wi-Fi 2,4 GHz), ESPHome 2026.9, Python 3 z numpy i matplotlib, hala w Swarzędzu, Zabbix 7.0 na mon01

Czas: ~120 min

Budujemy narzędzie, które mierzy siłę sygnału Wi-Fi w kolejnych punktach hali i rysuje z niej mapę z zaznaczonym progiem -65 dBm. Moduł ESP32 z ESPHome udostępnia sygnał przez swój serwer WWW, skrypt Python na laptopie zapisuje pomiary do pliku CSV, a drugi skrypt rysuje mapę za pomocą matplotlib. Mapa z tej strony pochodzi z danych sztucznych, bo w hali nie robiliśmy pomiarów.

ZAŁOŻENIA SCENARIUSZA

hala w Swarzędzu jest w sieci centrali: moduł ESP32 łączy się z siecią `Zielona-Magazyn` (VLAN 30), dostaje adres z DHCP na `fw01` i wysyła pomiary do `mon01` w VLAN 10 przez `fw01`, bez tunelu. Wariant z DHCP na `dc01` i `dc02`: DHCP: adresy dla biura i magazynu z przełączaniem awaryjnym. Wariant z osobną lokalizacją: IPsec site-to-site: Pamiątkowo i Swarzędz między FortiGate a routerem innej firmy.

Skróty użyte na tej stronie (19)

Skrót	Co znaczy
API	Application Programming Interface: sposób, w jaki programy rozmawiają ze sobą
BSSID	Basic Service Set Identifier: adres MAC konkretnego radia punktu dostępowego, po którym odróżniamy dwa punkty rozgłaszające ten sam SSID
CSV	w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami
DHCP	Dynamic Host Configuration Protocol: usługa, która sama rozdaje komputerom adresy IP
ESP32	Tani mikrokontroler z Wi-Fi i Bluetooth firmy Espressif: malutki komputer, z którego budujemy własne czujniki sieciowe

Skrót	Co znaczy
ESPHome	Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć
HTTP	Hypertext Transfer Protocol: protokół stron WWW
IDW	Inverse Distance Weighting: metoda interpolacji, w której wartość w punkcie jest średnią ważoną pomiarów z sąsiedztwa, a bliższe pomiary liczą się bardziej; użyta do narysowania mapy zasięgu Wi-Fi
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
JSONPath	Zapis ścieżki, która wskazuje jedną wartość w dokumencie JSON, np. $.value to pole value na samym wierzchu
MAC	Media Access Control: unikalny adres fizyczny karty sieciowej, przypisany fabrycznie
OTA	Over The Air: wgrywanie nowego oprogramowania do urządzenia przez sieć, bez kabla
REST	sposób budowania API przez zwykłe żądania HTTP (GET, POST) zamiast osobnego protokołu
UniFi	linia sprzętu Ubiquiti do sieci firmowych i domowych: punkty dostępowe Wi-Fi, przełączniki i bramy, zarządzana z aplikacji UniFi Network
USB	Universal Serial Bus: standard portów i urządzeń podłączanych do komputera, tu pendrive'y i inne nośniki wymienne
VLAN	Virtual LAN: wydzielona logicznie sieć na tym samym sprzęcie
Wi-Fi	Sieć bezprzewodowa: wygodna w biurze, ale w serwerowni dokłada kolejną możliwą przyczynę awarii
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS
YAML	YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach

Fiszka

TYLKO ODCZYT

Sprawdzenie konfiguracji ESPHome: esphome config

```
esphome config esp-mapa-wifi.yaml
```

Polecenie sprawdza plik YAML bez kompilacji i bez płytki. Konfigurację z tej strony sprawdziliśmy na wersji 2026.9.1 i jej wynik brzmiał „Configuration is valid!”. Kompilacji ani pracy na prawdziwej płytce nie sprawdzaliśmy.

ESPHome = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć · **YAML** = YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach · **esphome** = ESPHome: buduje i wgrywa oprogramowanie na ESP32 z pliku YAML, pokazuje logi urządzenia

TYLKO ODCZYT

Wgranie programu na płytkę: esphome run

```
esphome run esp-mapa-wifi.yaml
```

Pierwsze wgranie idzie przez USB, kolejne przez sieć (OTA) z hasłem z pliku `secrets.yaml`.

USB = Universal Serial Bus: standard portów i urządzeń podłączanych do komputera, tu pendrive'y i inne nośniki wymienne · **OTA** = Over The Air: wgrywanie nowego oprogramowania do urządzenia przez sieć, bez kabla · **esphome** = ESPHome: buduje i wgrywa oprogramowanie na ESP32 z pliku YAML, pokazuje logi urządzenia

TYLKO ODCZYT

Odczyt sygnału z modułu: curl

```
curl -s -u "NazwaUsera:HasloWebServera" http://AdresESP/sensor/sygnal_wifi
```

> (gdzie `AdresESP` to adres IP modułu, który widzimy w logu po starcie albo w panelu DHCP na `fw01`, a `NazwaUsera` i `HasloWebServera` to dane z `secrets.yaml`)

Przykład (poglądowy; format według dokumentacji ESPHome, wartości wymyślone):

```
{"id":"sensor-sygnal_wifi","value":-61,"state":"-61 dBm"}
```

Nazwy ścieżek `/sensor/sygnal_wifi` i `/text_sensor/bssid_punktu` wynikają z nazw czujników w pliku YAML i przed pomiarem sprawdzamy je w przeglądarce, wchodząc na adres modułu.

IP = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) · **DHCP** = Dynamic Host Configuration Protocol: usługa, która sama rozdaje komputerom adresy IP · **ESPHome** = Oprogramowanie, które zamienia ESP32 w gotowy czujnik sieciowy bez pisania kodu od zera, a odczyty udostępnia przez sieć · **YAML** = YAML Ain't Markup Language: format plików konfiguracyjnych oparty na wcięciach · **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

□□ ZMIENIA SYSTEM

Przygotowanie środowiska Python: venv i pip

```
python3 -m venv ~/mapa-wifi
~/mapa-wifi/bin/pip install numpy matplotlib
```

□□ **python3** = uruchom interpreter Pythona w wersji 3 albo wskazany skrypt

□□ ZMIENIA SYSTEM

Rejestracja pomiarów i rysowanie mapy: loguj_rssi.py i mapa_wifi.py

```
export ESP_UZYTKOWNIK="NazwaUsera"
export ESP_HASLO="HasloWebServera"
~/mapa-wifi/bin/python loguj_rssi.py --zrodlo esp --adres AdresESP --plik pomiary.csv
~/mapa-wifi/bin/python mapa_wifi.py pomiary.csv mapa-wifi.png
```

Przykład (poglądowy; wynik skryptu mapy dla danych sztucznych z tej strony):

```
punktów: 60, słabszych niż -65 dBm: 17
```

□□ **export** = ustaw zmienną środowiskową widoczną dla uruchamianych programów

Z życia wzięte

□□ Z ŻYCIA WZIĘTE

Piątek, 14:00. Marek mówi, że w trzecim rzędzie regałów skanery tracą sieć, a wykonawca punktów dostępowych odpowiada, że „pokrycie jest wystarczające”, bo tak wyszło z programu na laptopie. Tomek nie ma licencji na żaden program do pomiarów, a ma w szufladzie płytkę ESP32 i powerbank. Pierwszego wieczoru robi 60 punktów pomiarowych co kilka metrów w przejściach, a następnego dnia rysuje mapę i pokazuje Markowi i wykonawcy dokładnie, gdzie sygnał spada poniżej -65 dBm.

Projekt DIY krok po kroku

Co budujemy

Moduł ESP32 trzyma połączenie z siecią Wi-Fi hali i co kilka sekund mierzy siłę sygnału, a razem z nią zapamiętuje adres MAC punktu dostępowego (BSSID), z którym jest połączony. Laptop, który niesiemy razem z modułem, odpytuje serwer WWW płytki, bierze medianę z kilku próbek i zapisuje jeden wiersz CSV na punkt pomiarowy. Na końcu skrypt rysuje mapę.



Rysunek 1. Przepływ pomiaru: ESP32 mierzy sygnał, laptop czyta go przez REST, zapisuje do pliku CSV, a skrypt matplotlib rysuje mapę zasięgu z progiem -65 dBm. Warto zauważyć, że laptop i moduł muszą być w tej samej sieci, bo laptop odpytuje płytkę po jej adresie IP. (kliknięcie otwiera pełny rozmiar)

Lista części

Część	Ilość	Uwagi
Płytka ESP32 z Wi-Fi 2,4 GHz (na przykład ESP32 DevKit)	1	wersja klasyczna widzi tylko pasmo 2,4 GHz
Powerbank z wyjściem USB i kabel	1	zasilanie podczas chodzenia po hali
Laptop z Pythonem 3	1	do rejestratora i mapy
Taśma miernicza albo dalmierz laserowy	1	do wyznaczania współrzędnych punktów
Plan hali z regałami i punktami dostępowymi	1	wymiary z planu albo z pomiaru
Pudełko plastikowe i uchwyt na tyczce	1	płytka na wysokości trzymania skanera

Schemat połączeń

Płytka nie ma żadnych elementów dodatkowych: zasila ją USB z powerbanku, a komunikacja idzie przez Wi-Fi. Połączeń jest tyle, ile pokazuje rysunek 1: ESP32 do punktu dostępowego, punkt dostępowy do laptopa, laptop do pliku.

Konfiguracja ESPHome

Plik `secrets.yaml` trzyma hasła poza plikiem konfiguracji. Wartości poniżej to placeholdery, a klucz API generujemy poleceniem `openssl rand -base64 32`, jak opisuje strona Mapa cieplna na ESP32: ΔT szafy i różnica ciśnień między korytarzami (DIY).

```
# secrets.yaml, obok pliku esp-mapa-wifi.yaml
wifi_ssid: "Zielona-Magazyn"
wifi_haslo: "HasloSieciMagazyn"
api_klucz: "WynikPolecenia_openssl_rand_base64_32"
ota_haslo: "HasloOta"
web_uzytkownik: "NazwaUsera"
web_haslo: "HasloWebServera"
```

Plik konfiguracyjny:

```
esphome:
  name: esp-mapa-wifi
  friendly_name: Pomiar Wi-Fi magazyn

esp32:
  board: esp32dev
  framework:
    type: esp-idf

logger:

wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_haslo
  power_save_mode: none

api:
  encryption:
    key: !secret api_klucz

ota:
  - platform: esphome
    password: !secret ota_haslo

web_server:
  port: 80
```

```
  version: 3
  auth:
    username: !secret web_uzytkownik
    password: !secret web_haslo

sensor:
  - platform: wifi_signal
    name: "Sygnal Wifi"
    id: sygnal_wifi
    update_interval: 2s

text_sensor:
  - platform: wifi_info
    bssid:
      name: "BSSID Punktu"
      id: bssid_punktu
    ssid:
      name: "SSID"
    ip_address:
      name: "Adres IP"
```

Czujniki `wifi_signal` i `wifi_info` oraz serwer WWW z REST opisuje dokumentacja ESPHome [1][2]. Moduł łączy się z siecią `Zielona-Magazyn`, czyli propozycją sieci dla skanerów z VLAN 30 ze strony Wi-Fi od strony fizycznej: pasma, kanały i rozmieszczenie punktów. Laptop musi być w tej samej sieci, żeby sięgnąć adresu płytki, a zapora `fw01` nie przepuści takiego ruchu z VLAN 20. Wyłączone oszczędzanie energii (`power_save_mode: none`) skraca opóźnienia odczytów.

Rejestrator pomiarów

Skrypt pyta o współrzędne punktu, zbiera kilka próbek, wybiera medianę i najczęstszy BSSID, po czym dopisuje wiersz do pliku CSV. Mediana odrzuca pojedyncze odchyłki, a BSSID mówi, z którym punktem dostępowym płytka była wtedy połączona. Parametr `--poprawka` służy do kalibracji.

```
#!/usr/bin/env python3
"""Rejestrator RSSI: dla każdego punktu pomiaru zapisuje medianę z kilku próbek do pliku
CSV."""
import argparse
import base64
import csv
import json
```

```
import os
import platform
import re
import statistics
import subprocess
import sys
import time
import urllib.request
from collections import Counter

NAGLOWEK = ["punkt", "x_m", "y_m", "bssid", "rssi_dbm", "probki"]


def odczyt_esp(adres, uzytkownik, haslo):
    """Czyta sygnał i BSSID z serwera WWW modułu ESP32 (ESPHome, REST)."""
    def pobierz(sciezka):
        zadanie = urllib.request.Request(f"http://{adres}{sciezka}")
        if uzytkownik:
            dane = base64.b64encode(f"{uzytkownik}:{haslo}".encode()).decode()
            zadanie.add_header("Authorization", f"Basic {dane}")
        with urllib.request.urlopen(zadanie, timeout=5) as odp:
            return json.load(odp)
    sygnal = pobierz("/sensor/sygnal_wifi")
    bssid = pobierz("/text_sensor/bssid_punktu")
    return float(sygnal["value"]), str(bssid.get("value", bssid.get("state", ""))).upper()


def parsuj_iw(tekst):
    """Wynik polecenia: iw dev NazwaKarty link."""
    s = re.search(r"signal:\s*(-?\d+)\s*dBm", tekst)
    b = re.search(r"Connected to\s+((?:[0-9a-f]{2}:){5}[0-9a-f]{2})", tekst, re.I)
    if not s or not b:
        raise ValueError("brak połączenia lub nieznany format iw")
    return float(s.group(1)), b.group(1).upper()


def parsuj_netsh(tekst):
    """Wynik polecenia: netsh wlan show interfaces (angielski i polski Windows)."""
    s = re.search(r"(?:Signal|Sygnał)\s*:\s*(\d+)\s*%", tekst)
    b = re.search(r"BSSID\s*:\s*((?:[0-9a-f]{2}:){5}[0-9a-f]{2})", tekst, re.I)
```

```
    if not s or not b:
        raise ValueError("brak połączenia lub nieznany format netsh")
    return int(s.group(1)) / 2 - 100, b.group(1).upper()


def odczyt_laptop(karta):
    if platform.system() == "Windows":
        wynik = subprocess.run(["netsh", "wlan", "show", "interfaces"],
                               capture_output=True, text=True, check=True)
        return parsuj_netsh(wynik.stdout)
    wynik = subprocess.run(["iw", "dev", karta, "link"],
                           capture_output=True, text=True, check=True)
    return parsuj_iw(wynik.stdout)


def zmierz(odczyt, probki, odstep):
    """Zbiera próbki i zwraca medianę RSSI oraz najczęstszy BSSID."""
    wartosci, punkty = [], []
    for _ in range(probki):
        try:
            rssi, bssid = odczyt()
            wartosci.append(rssi)
            punkty.append(bssid)
        except Exception as blad:
            print("  próbka pominięta:", blad)
        time.sleep(odstep)
    if not wartosci:
        raise RuntimeError("brak poprawnych próbek")
    return statistics.median(wartosci), Counter(punkty).most_common(1)[0][0], len(wartosci)


def main():
    p = argparse.ArgumentParser(description=__doc__)
    p.add_argument("--zrodlo", choices=["esp", "laptop"], default="esp")
    p.add_argument("--adres", help="adres IP lub nazwa modułu ESP32")
    p.add_argument("--karta", default="wlan0", help="karta Wi-Fi laptopa (Linux)")
    p.add_argument("--plik", default="pomiary.csv")
    p.add_argument("--probki", type=int, default=7)
    p.add_argument("--odstep", type=float, default=1.0)
    p.add_argument("--poprawka", type=float, default=0.0,
```

```
                    help="dB dodawane do wyniku (kalibracja względem skanera)")
    a = p.parse_args()

    if a.zrodlo == "esp":
        if not a.adres:
            sys.exit("Brak parametru --adres modułu ESP32")
        uzytkownik = os.environ.get("ESP_UZYTKOWNIK", "")
        haslo = os.environ.get("ESP_HASLO", "")
        odczyt = lambda: odczyt_esp(a.adres, uzytkownik, haslo)
    else:
        odczyt = lambda: odczyt_laptop(a.karta)

    nowy = not os.path.exists(a.plik)
    with open(a.plik, "a", newline="", encoding="utf-8") as f:
        zapis = csv.writer(f)
        if nowy:
            zapis.writerow(NAGLOWEK)
        numer = 1
        while True:
            try:
                wpis = input("Współrzędne punktu 'x y' w metrach (pusty wiersz kończy):
").strip()
            except EOFError:
                break
            if not wpis:
                break
            try:
                x, y = (float(v.replace(",", ".")) for v in wpis.split())
            except ValueError:
                print("  oczekuję dwóch liczb, na przykład: 12,5 7")
                continue
            rssi, bssid, ile = zmierz(odczyt, a.probki, a.odstep)
            rssi += a.poprawka
            zapis.writerow([f"P{numer:03d}", x, y, bssid, f"{rssi:.1f}", ile])
            f.flush()
            print(f"  P{numer:03d}: {rssi:.1f} dBm, punkt dostępowy {bssid}, próbek {ile}")
            numer += 1


if __name__ == "__main__":
```

```
    main()
```

Skrypt sprawdziliśmy na przykładowych tekstach z poleceń `iw` i `netsh` oraz na sztucznym serwerze HTTP, który odpowiada jak ESPHome. Z prawdziwą płytką go nie próbowaliśmy.

Mapa zasięgu

Skrypt wczytuje CSV i oblicza wartość w każdej komórce siatki co pół metra metodą odwrotnych odległości (IDW), czyli średnią ważoną pomiarów, w której bliższe punkty liczą się bardziej [3]. Na mapie zaznaczamy regały, punkty dostępowe, punkty pomiarowe i linię progu -65 dBm.

```
#!/usr/bin/env python3
"""Rysuje mapę zasięgu Wi-Fi hali na podstawie pomiary.csv (interpolacja IDW)."""
import csv
import sys

import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.patches import Rectangle

DLUGOSC, SZEROKOSC = 60.0, 30.0                      # wymiary hali w metrach (przykład)
REGALY = [(5, y, 50, 1.2) for y in (5, 10, 15, 20, 25)]   # x, y, długość, grubość
PUNKTY_DOSTEPOWE = {"ap05": (12, 7.5), "ap06": (48, 7.5), "ap07": (30, 17.5), "ap08": (12,
27.5)}
PROG = -65.0                                         # dBm, cel zasięgu skanerów


def wczytaj(plik):
    x, y, v = [], [], []
    with open(plik, newline="", encoding="utf-8") as f:
        for w in csv.DictReader(f):
            x.append(float(w["x_m"]))
            y.append(float(w["y_m"]))
            v.append(float(w["rssi_dbm"]))
    return np.array(x), np.array(y), np.array(v)


def idw(x, y, v, gx, gy, p=2.0):
```

```
    """Interpolacja odwrotnych odległości; nie zna regałów, więc potrzebuje gęstych
punktów."""
    d = np.hypot(gx[..., None] - x, gy[..., None] - y)
    d = np.maximum(d, 1e-6)
    w = 1.0 / d ** p
    return (w * v).sum(axis=-1) / w.sum(axis=-1)


def main(plik="pomiary.csv", wynik="mapa-wifi.png"):
    x, y, v = wczytaj(plik)
    gx, gy = np.meshgrid(np.arange(0, DLUGOSC + 0.01, 0.5), np.arange(0, SZEROKOSC + 0.01,
0.5))
    z = idw(x, y, v, gx, gy)

    fig, os_ = plt.subplots(figsize=(11, 6))
    obraz = os_.imshow(z, extent=(0, DLUGOSC, 0, SZEROKOSC), origin="lower",
                       cmap="RdYlGn", vmin=-90, vmax=-40, alpha=0.85)
    os_.contour(gx, gy, z, levels=[PROG], colors="black", linestyles="--", linewidths=1.4)
    for rx, ry, dl, gr in REGALY:
        os_.add_patch(Rectangle((rx, ry), dl, gr, color="#444444", alpha=0.9))
    os_.scatter(x, y, c=v, cmap="RdYlGn", vmin=-90, vmax=-40, edgecolors="black", s=40,
zorder=3)
    for nazwa, (ax, ay) in PUNKTY_DOSTEPOWE.items():
        os_.plot(ax, ay, marker="^", color="navy", markersize=13, zorder=4)
        os_.annotate(nazwa, (ax, ay), textcoords="offset points", xytext=(6, 8), fontsize=9,
color="navy")
    os_.set_xlabel("x [m]")
    os_.set_ylabel("y [m]")
    os_.set_title(f"Mapa zasięgu Wi-Fi hali: linia przerywana to {PROG:.0f} dBm")
    os_.set_aspect("equal")
    fig.colorbar(obraz, ax=os_, label="RSSI [dBm]", shrink=0.8)
    slabe = int((v < PROG).sum())
    print(f"punktów: {len(v)}, słabszych niż {PROG:.0f} dBm: {slabe}")
    fig.tight_layout()
    fig.savefig(wynik, dpi=130)


if __name__ == "__main__":
    main(*sys.argv[1:3])
```

Metoda IDW nie zna regałów, więc między punktami pomiarowymi mapa jest przybliżeniem. Dlatego punkty gęsto wyznaczamy w przejściach, a nie wewnątrz rzędów, i traktujemy mapę jako wskazówkę, gdzie mierzyć dokładniej.

Dane przykładowe

Do próby skryptu mapy wygenerowaliśmy sztuczne dane: 60 punktów w hali 60 na 30 metrów z pięcioma rzędami regałów i czterema punktami dostępowymi, w modelu z tłumieniem 9 dB na każdy regał. Skrypt jest tu tylko po to, żeby każdy mógł powtórzyć rysunek bez hali.

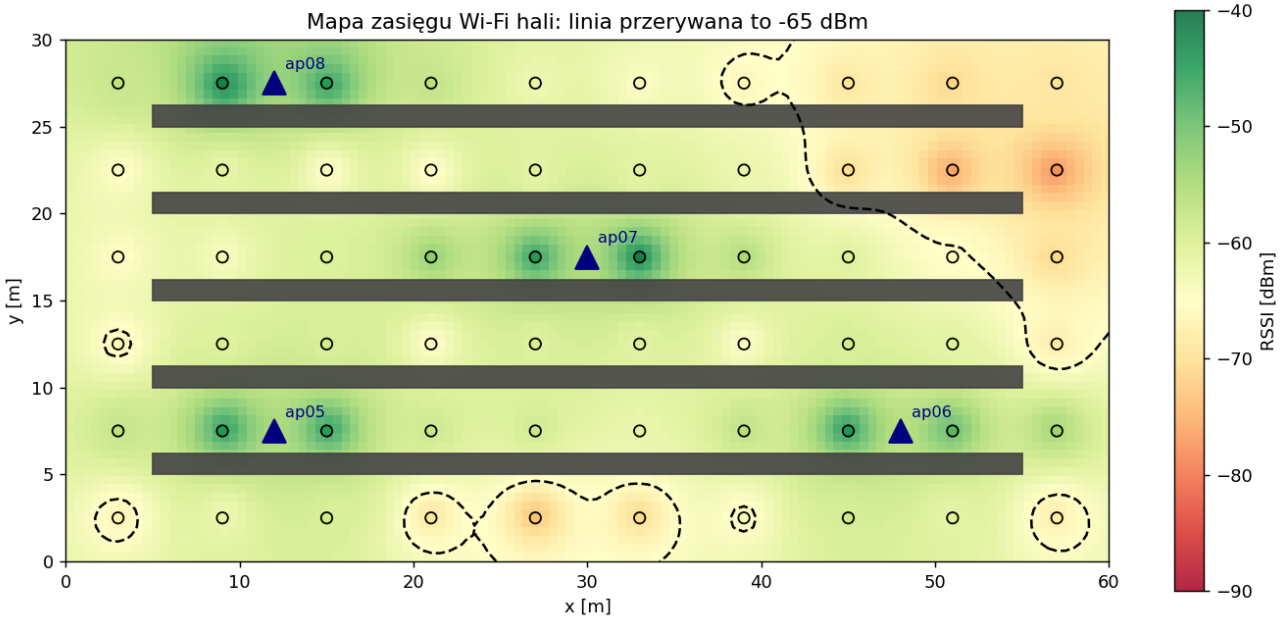
```
#!/usr/bin/env python3
"""Generuje PRZYKŁADOWE pomiary hali (dane sztuczne, tylko do próby skryptu mapy)."""
import csv
import math
import random

random.seed(7)
REGALY = [(5, y, 50, 1.2) for y in (5, 10, 15, 20, 25)]          # x, y, długość, grubość
AP = {"AA:BB:CC:00:00:05": (12, 7.5), "AA:BB:CC:00:00:06": (48, 7.5),
      "AA:BB:CC:00:00:07": (30, 17.5), "AA:BB:CC:00:00:08": (12, 27.5)}


def regaly_na_drodze(x1, y1, x2, y2):
    ile = 0
    for rx, ry, dl, gr in REGALY:
        trafiony = any(rx <= x1 + (x2 - x1) * t / 50 <= rx + dl and ry <= y1 + (y2 - y1) * t /
50 <= ry + gr
                       for t in range(51))
        ile += trafiony
    return ile


with open("pomiary.csv", "w", newline="", encoding="utf-8") as f:
    w = csv.writer(f)
    w.writerow(["punkt", "x_m", "y_m", "bssid", "rssi_dbm", "probki"])
    n = 1
    for y in (2.5, 7.5, 12.5, 17.5, 22.5, 27.5):
        for x in range(3, 58, 6):
            najlepszy = None
            for bssid, (ax, ay) in AP.items():
```

```
                d = max(math.hypot(x - ax, y - ay), 1.0)
                rssi = -35 - 22 * math.log10(d) - 9 * regaly_na_drodze(x, y, ax, ay) +
random.gauss(0, 1.5)
                if najlepszy is None or rssi > najlepszy[1]:
                    najlepszy = (bssid, rssi)
            w.writerow([f"P{n:03d}", x, y, najlepszy[0], f"{najlepszy[1]:.1f}", 7])
            n += 1
```



Rysunek 2. Mapa zasięgu na danych sztucznych: hala 60 na 30 metrów, pięć rzędów regałów, cztery punkty dostępowe i linia -65 dBm. Warto zauważyć, że w prawym górnym rogu sygnał spada poniżej progu, mimo że sąsiednie przejścia są w porządku. (kliknięcie otwiera pełny rozmiar)

Kalibracja

Antena ESP32 odbiera inaczej niż skaner, więc przed pomiarem porównujemy oba urządzenia w pięciu punktach hali i obliczamy medianę różnicy. Tę różnicę wpisujemy w parametr `--poprawka`, na przykład `--poprawka -3`, jeśli skaner pokazuje stale o 3 dB mniej niż płytka. Bez takiej kalibracji próg -65 dBm z mapy odnosi się do płytki, a nie do skanerów.

Obudowa i sposób pracy

Płytkę mocujemy w plastikowym pudełku na tyczce na wysokości, na której pracownik trzyma skaner, około 1,2 metra, do potwierdzenia z Markiem. Metalowa obudowa albo ręka zasłaniająca antenę zmieniają wynik, więc antena pozostaje wolna, a tyczkę trzymamy zawsze tak samo. Pomiary robimy przy typowym wypełnieniu regałów, poza zmianą o 6:00 i poza inwentaryzacją, tak by nie przeszkadzać w pracy.

Drugi wariant: sam laptop

Skrypt `loguj_rssi.py` ma tryb `--zrodlo laptop`, który czyta siłę sygnału z samego laptopa. W Linuksie używa polecenia `iw dev NazwaKarty link`, a w Windowsie `netsh wlan show interfaces`, w którym sygnał podawany jest w procentach i przeliczany na dBm wzorem: procent podzielony przez 2, minus 100. Przeliczenie jest przybliżone, bo producenci kart różnie ustalają skalę.

Wariant laptopowy jest najprostszy, ale mierzy to, co widzi karta laptopa, a nie skaner, więc kalibrację robimy tak samo. Pełną listę widocznych punktów dostępowych w danym miejscu daje polecenie `nmcli -f SSID,BSSID,CHAN,SIGNAL device wifi list` z Linuksa, jak na stronie o Wi-Fi.

Płytka ESP32-C5 ma radio 5 GHz i jest obsługiwana przez ESPHome. Konfiguracja z płytką `esp32-c5-devkitc-1`, wariantem `esp32c5` i opcją `band_mode: 5GHZ` w sekcji `wifi` przeszła polecenie `esphome config`. Kompilacji ani pracy na płytce nie sprawdzaliśmy, więc przed użyciem próbujemy na jednej sztuce.

Integracja z monitoringiem

Jeśli moduł zostanie na stałe w hali, Zabbix na `mon01` może co minutę pobierać siłę sygnału i pokazywać jej przebieg w ciągu dnia, tak jak pobiera dane z modułów `esp01` i `esp02`. Dzięki temu widać, że sygnał spada, gdy regały są pełne, albo że po zmianie ustawień punktu dostępowego coś się pogorszyło.

Pole elementu danych w Zabbix	Wartość
Typ	agent HTTP
Adres URL	`http://AdresESP/sensor/sygnal_wifi`
Uwierzytelnianie	podstawowe, dane z `secrets.yaml`
Przetwarzanie wstępne	JSONPath `$.value`
Typ danych	liczba zmiennoprzecinkowa, jednostka dBm
Interwał	60 s
Wyzwalacz	wartość poniżej -75 dBm przez 10 minut

◫ ZRZUT EKRANU DO UZUPEŁNIENIA

Zrzut do uzupełnienia: formularz elementu danych typu agent HTTP w Zabbix z adresem `/sensor/sygnal_wifi` i przetwarzaniem JSONPath.

Strona Mapa cieplna na ESP32: ΔT szafy i różnica ciśnień między korytarzami (DIY) pokazuje ten sam wzorzec: odczyt przez `curl` i pobranie pola `value` przez Zabbix. Pola elementu danych opisuje dokumentacja Zabbix [4].

Bezpieczeństwo i ograniczenia

Moduł z włączonym serwerem WWW działa w sieci skanerów, więc zabezpieczamy go loginem i hasłem z `secrets.yaml`, a hasło sieci Wi-Fi nie trafia do pliku YAML ani do repozytorium. Serwer HTTP nie szyfruje ruchu, więc po pomiarach moduł wyłączamy albo zostawiamy go w osobnej sieci testowej. Skrypt czyta hasło ze zmiennej środowiskowej, żeby nie wisiało w historii poleceń.

ESP32 klasyczny mierzy tylko pasmo 2,4 GHz, więc mapa dotyczy tego pasma. Skanery w paśmie 5 GHz tłumią się mocniej, więc dla nich pomiar robimy płytką z radiem 5 GHz albo laptopem. Mapa z IDW jest przybliżeniem, a jedna próbka nie jest pomiarem: dlatego mediana z kilku próbek i gęsta siatka punktów.

W hali pracują wózki widłowe i ludzie na drabinach, więc pomiary robimy przy wyłączonym ruchu w danym przejściu i z drugą osobą pilnującą trasy.

> Więcej możliwości albo gotowe rozwiązanie od ręki (out of the box): programy do pomiaru pokrycia (site survey), na przykład Ekahau czy NetSpot, rysują mapę na planie budynku, mierzą wiele punktów dostępowych naraz i tworzą raport, a kontrolery punktów dostępowych mają często wbudowane narzędzia do pomiaru zasięgu. W dużych centrach danych i centrach logistycznych stosuje się zwykle stałe sondy Wi-Fi albo punkty dostępowe z trybem czujnika, które mierzą jakość sieci nieustannie, a nie tylko podczas jednego przejścia z laptopem. Dokładne możliwości zależą od producenta kontrolera i sprawdzamy je w jego dokumentacji. W Zielonej punkty `ap01` do `ap09` zarządza UniFi Network, więc jego narzędzia do pomiaru zasięgu sprawdzamy w dokumentacji naszej wersji, jak opisuje strona UISP i UniFi: dwa systemy zarządzania sprzętem Ubiquiti i czym się różnią.

Częste błędy

Objaw	Przyczyna	Co robimy

Skrypt zgłasza błąd 401	zła nazwa lub hasło serwera WWW	sprawdzamy zmienne `ESP_UZYTKOWNIK` i `ESP_HASLO`
Skrypt zgłasza błąd 404	ścieżka czujnika inna niż w skrypcie	sprawdzamy adres w przeglądarce i poprawiamy nazwy w YAML i skrypcie
Brak odpowiedzi modułu	laptop w innej sieci VLAN niż płytka	łączymy laptop z tą samą siecią Wi-Fi co płytka
Wyniki skaczą o kilkanaście dB	pojedyncze próbki, ręka na antenie	zwiększamy liczbę próbek, mocujemy płytkę na tyczce
W jednym miejscu inny BSSID niż oczekiwany	roaming płytki do innego punktu	zapisujemy BSSID i bierzemy go pod uwagę przy ocenie
Mapa nie pokazuje słabych miejsc za regałem	IDW nie zna regałów	dokładamy punkty pomiarowe w przejściach i na końcach rzędów

□□ Ćwiczenia

1. Dlaczego skrypt zapisuje medianę z kilku próbek, a nie jedną wartość?

Odpowiedź

Pojedyncza próbka może być zakłócona chwilową przeszkodą, ręką albo ruchem w hali. Mediana z kilku próbek odrzuca skrajne wartości i daje stabilny wynik.

1. Laptop z Windowsem pokazuje sygnał 70 procent. Ile to w przybliżeniu dBm?

Odpowiedź

70 podzielone przez 2 to 35, minus 100 daje około -65 dBm, czyli dokładnie próg z mapy. Przeliczenie jest przybliżone.

1. Czemu mapa nie powinna być jedyną podstawą decyzji o dołożeniu punktu dostępowego?

Odpowiedź

Interpolacja IDW nie zna regałów ani ścian i wygładza wartości między punktami. Przed zakupem sprawdzamy pomiarem dodatkowe punkty w miejscu, które wygląda na słabe.

Powiązane: Wi-Fi od strony fizycznej: pasma, kanały i rozmieszczenie punktów, PoE: klasy mocy, budżet przełącznika i długość kabla, Mapa cieplna na ESP32: ΔT szafy i różnica ciśnień między

korytarzami (DIY), UISP i UniFi: dwa systemy zarządzania sprzętem Ubiquiti i czym się różnią

Źródła:

1. ESPHome, „WiFi Signal Sensor” i „WiFi Info Text Sensor”: esphome.io
2. ESPHome, „Web Server Component”, REST API: esphome.io
3. Wikipedia, „Inverse distance weighting”: en.wikipedia.org/wiki/Inverse_distance_weighting
4. Zabbix, „HTTP agent”: zabbix.com