POLITECHNIKA GDAŃSKA

WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

Imię i nazwisko studenta: Tobiasz Dryjański
Nr albumu: 142987
Studia drugiego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Elektronika i telekomunikacja
Specjalność/profil: -

PRACA DYPLOMOWA MAGISTERSKA

Tytuł pracy w języku polskim: Urządzenie wbudowane do lokalizacji terminali mobilnych wewnątrz budynków w pasmie 2,4 GHz zintegrowane z anteną ESPAR

Tytuł pracy w języku angielskim: Embedded device for indoor positioning of mobile terminals in ISM 2.4 GHz frequency band integrated with ESPAR antenna.

Potwierdzenie przyjęcia pracy	
Opiekun pracy podpis	Kierownik Katedry/Zakładu (pozostawić właściwe) podpis
dr inż. Łukasz Kulas	

Data oddania pracy do dziekanatu:

POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

OŚWIADCZENIE dotyczące pracy dyplomowej zatytułowanej: Urządzenie wbudowane do lokalizacji terminali mobilnych wewnątrz budynków w pasmie 2,4 GHz zintegrowane z anteną ESPAR

Imię i nazwisko studenta: Tobiasz Dryjański
Data i miejsce urodzenia: 25.08.1993, Gdańsk
Nr albumu: 142987
Wydział: Wydział Elektroniki, Telekomunikacji i Informatyki
Kierunek: elektronika i telekomunikacja
Poziom kształcenia: drugi
Forma studiów: stacjonarne

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz.U. z 2017 poz. 880 z późn. zm.) i konsekwencji dyscyplinarnych określonych w ustawie Prawo o szkolnictwie wyższym (Dz.U. 2017 poz. 2183 z późn. zm.),[1] a także odpowiedzialności cywilnoprawnej oświadczam, że przedkładana praca dyplomowa została opracowana przeze mnie samodzielnie.

Niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadaniem tytułu zawodowego.

Wszystkie informacje umieszczone w ww. pracy dyplomowej, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami zgodnie z art. 34 ustawy o prawie autorskim i prawach pokrewnych.

Potwierdzam zgodność niniejszej wersji pracy dyplomowej z załączoną wersją elektroniczną.

Gdańsk, dnia ....
podpis studenta

[1] Ustawa z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym:
Art. 214 ustęp 4. W razie podejrzenia popełnienia przez studenta czynu podlegającego na przypisaniu sobie autorstwa istotnego fragmentu lub innych elementów cudzego utworu rektor niezwłocznie poleca przeprowadzenie postępowania wyjaśniającego.
Art. 214 ustęp 6. Jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdza popełnienie czynu, o którym mowa w ust. 4, rektor wstrzymuje postępowanie o nadanie tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz składa zawiadomienie o popełnieniu przestępstwa.

STRESZCZENIE

W dobie wielofunkcyjnych telefonów komórkowych lokalizacja bezprzewodowa jest jedną z najważniejszych gałęzi rozwoju telekomunikacji. Funkcjonalność ta jest możliwa dzięki globalnym systemom lokalizacji takim jak GPS, którego usługi są dostępne dla każdego przeciętnego użytkownika. Systemy globalne cierpią jednak z powodu swojej niskiej dokładności w zamkniętych środowiskach takich jak lasy czy wnętrza budynków. Popularnym obejściem tego ograniczenia jest wykorzystanie lokalnych systemów lokalizacji wewnątrz budynków. Wadą takich rozwiązań jest jednak wysoki koszt związany z infrastrukturą sprzętową.

Celem pracy było stworzenie urządzenia implementującego funkcjonalność lokalizacji terminali mobilnych opartej o sieć lokalną Wi-Fi i antenę inteligentną ESPAR. Punktem wyjściowym było wyjaśnienie teoretycznych zagadnień związanych z algorytmami lokalizacji oraz stworzenie projektu tworzonego systemu. Przedstawiono proces projektowania takiego urządzenia poprzez rozwiązywanie kolejnych problemów implementacyjnych. Wynikiem projektu jest system umożliwiający lokalizację terminala mobilnego podłączonego do sieci Wi-Fi. System został poddany testom dokładności, wykonanym w przestrzeni dwuwymiarowej. Podczas pomiarów zdefiniowano punkty referencyjne i testowe dla algorytmu lokalizacji k-NN. Sprawdzono dokładność lokalizacji terminala mobilnego w punktach testowych zależnie od parametrów użytego algorytmu. Ostateczny wynik testów wykazał zadowalającą dokładność lokalizacji.

Praca udowadnia, iż możliwe jest stworzenie lokalnego systemu lokalizacji bez znacznych zasobów sprzętowych. Wykorzystanie urządzenia lokalizującego z anteną ESPAR oraz *access pointów* Wi-Fi zamiast standardowej sieci sensorów jest wystarczające do stworzenia kompletnego systemu.

Słowa kluczowe: Lokalizacja bezprzewodowa, Komunikacja bezprzewodowa, k-najbliższych sąsiadów (k-NN), Antena ESPAR, Technologie mobilne, Monitorowanie sieci

Dziedzina nauki i techniki, zgodnie z wymogami OECD: Nauki inżynieryjne i techniczne, Elektrotechnika i elektronika

ABSTRACT

In the era of multifunctional mobile phones, wireless positioning is one of the most important branches of telecommunications development. This functionality is possible thanks to global positioning systems such as GPS, whose services are available to every average user. Global systems, however, suffer from their low accuracy in confined environments such as forests and building interiors. A popular workaround to this limitation is the use of local indoor positioning systems. The disadvantage of such solutions, however, is the high cost associated with the hardware infrastructure.

The goal of this thesis was to create a device implementing the functionality of the mobile terminal positioning based on a local Wi-Fi network and the ESPAR smart antenna. The thesis begins with an explanation of the theoretical issues related to positioning algorithms and with designing the developed system. The process of designing such a device is presented through solving subsequent implementation problems. The project resulted in creation of a system capable of positioning a mobile terminal connected to the Wi-Fi network. The system has been subjected to accuracy tests, carried out in a 2D space. During the measurements, reference and test points were defined for the k-NN positioning algorithm. The accuracy of the mobile terminal positioning in the test points was examined in dependence on the parameters of the algorithm. The tests proved the positioning accuracy to be satisfactory.

The thesis proves the possibility of creating a local positioning system without using a significant hardware resources. Using a locating device with an ESPAR antenna and Wi-Fi access points instead of a standard sensor network is enough to create a complete system.

Keywords: Wireless positioning, Wireless communication, k-Nearest Neigbours (k-NN), ESPAR antenna, Mobile technology, Network monitoring

SCOTT (www.scott-project.eu) has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737422. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, Norway.

SPIS TREŚCI

WYKAZ WAŻNIEJSZYCH OZNACZEŃ I SKRÓTÓW

GNSS – **(ang. Global Navigation Satelite System)** – globalny system nawigacji satelitarnej

AP – **(ang. Access Point)** – punkt dostępu sieci bezprzewodowej

ESPAR – **(ang. Electronically Switched Parasitic Array Radiator)** – typ anteny inteligentnej

API – **(ang. Application Programming Interface)** – interfejs programowania aplikacji

UART – **(ang. Universal Asynchronous Receiver Transmitter)** – interfejs szeregowy

MAC – **(ang. Media Access Control)** – unikalny fizyczny adres urządzenia sieciowego

TCP – **(ang. Transmission Control Protocol)** – protokół implementujący czwartą warstwę modelu TCP-IP

Wi-Fi – **(ang. Wireless Fidelity)** – implementacja IEEE 802.11

k-NN – **(ang. k-Nearest Neighbours)** – algorytm k-najbliższych sąsiadów

CDF – **(ang. Cumulative Distribution Function)** – funkcja dystrybucji prawdopodobieństwa

k – ilość sąsiadów dla algorytmu k-NN

K – ilość stacji bazowych dla algorytmu k-NN

AOA – **(ang. Angle of Arrival)** – metoda lokalizacji oparta o kąt, z którego przybył sygnał

TOA – **(ang. Time of Arrival)** – metoda lokalizacji oparta o czas przybycia sygnału

TDOA – **(ang. Time Difference of Arrival)** – metoda lokalizacji oparta o różnice czasów przybycia sygnałów

RSS – **(ang. Received Signal Strength)** – siła odebranego sygnału

RSSI – **(ang. Received Signal Strength Indicator)** – indykator siły sygnału, zapisywany na jednym bajcie

2D – **(ang. Two-Dimensional)** – dwuwymiarowy

TA – **(ang. Transceiver Address)** – adres nadajnika

HTTP – **(ang. HyperText Transfer Protocol)** – protokół transferu stron internetowych

JS – JavaScript

ACK – **(ang. Acknowledged)** – potwierdzenie

NACK – **(ang. Not Acknowledged)** – brak potwierdzenia

SMA – **(ang. SubMiniature version A)** – rodzaj złącza mikrofalowego

SPDT – **(ang. Single-Pole Double-Throw)** – klucz dwustanowy

PC – **(ang. Personal Computer)** – komputer osobisty

OSI – **(ang. Open Systems Interconnection Reference Model)** – standard opisujący strukturę komunikacji sieciowej

JSON – **(ang. JavaScript Object Notation)** – format tekstowego opisu obiektów JS

GUI – **(ang. Graphical User Interface)** – interfejs graficzny

1. WSTĘP I CEL PRACY

Technologie bezprzewodowe są symbolem postępu dzisiejszych czasów. W XXI w. stały się one niezbędne dla społeczeństwa. Ogromna liczba ludzi posiada telefony czy inne urządzenia zapewniające bezprzewodowy dostęp do internetu i innych usług. Jednym z tych udogodnień jest lokalizacja bezprzewodowa, umożliwiająca znalezienie położenia danego obiektu w przestrzeni. Szukanie obiektów odbywa się najczęściej za pomocą sygnałów elektromagnetycznych. Wynika to z faktu istnienia wielu systemów umożliwiających taką lokalizację, jak np. sieci bezprzewodowe [1].

Lokalizację o największym zasięgu zapewniają globalne systemy nawigacji satelitarnej GNSS (ang. *global navigation satelite system*). Mogą one być zarówno lokalne (BDS, *Galileo*) lub globalne (GPS, GLONASS). Systemy takie jak GPS były początkowo tworzone w celach wojskowych. Z początkiem XXI wieku zostały jednak przekazane na użytek publiczny.

Standardowo w sieci bezprzewodowej rozróżnia się dwa rodzaje urządzeń: stacje bazowe i terminale mobilne. Stacja bazowa w odniesieniu do lokalizacji jest punktem referencyjnym. Punkt taki oznacza stacjonarny terminal lokalizujący. Pozycja takiego terminala oznacza, iż może on służyć jako punkt odniesienia dla ruchomego terminala [2]. Im więcej punktów referencyjnych tym większa szansa na dokładne wyniki lokalizacji.

W przypadku GNSS, rolę punktów referencyjnych pełnią satelity. Systemy takie jak GPS działają jednostronnie, co oznacza, że nie zachodzi żadna wymiana danych z terminalem lokalizowanym. Korzystanie z jednostronnego systemu umożliwia wsparcie dla nieskończenie wielu urządzeń, które mogą być lokalizowane za pomocą tego systemu. Wprowadzenie systemu na użytek publiczny przyniosło wiele możliwości odnośnie lokalizowania wielu terminali w skali całego globu [2].

Systemy lokalizacji znajdują wiele zastosowań w praktyce. Popularnymi zastosowaniami GPS są nawigacja samochodowa czy wspomaganie orientacji w terenie. Do najważniejszych można jednak zaliczyć te związane z bezpieczeństwem. Monitorowanie osób starszych, może stanowić zabezpieczenie przed osłabnięciami czy innymi wypadkami które mogą się takim osobom zdarzyć. Innym zastosowaniem może być wyznaczanie pozycji osób dzwoniących na linie ratunkowe [1]. Szybkość działania, zasięg oraz dokładność lokalizacji w tych przypadkach jest kluczowa.

Problemem systemów satelitarnych jest jednak fakt, iż działają one poprawnie jedynie gdy obiekt lokalizowany znajduje się na wolnej przestrzeni. Terminale znajdujące się w budynkach, w gęstych lasach lub pod ziemią mają niską dokładność lokalizacji [3]. Problem ten może być trudny do rozwiązania.

Inną niedoskonałością (lub zaletą w kontekście bezpieczeństwa) jest fakt iż terminal musi pozwolić na bycie zlokalizowanym, tzn. lokalizacja musi być świadomym wyborem użytkownika. Wynika to z faktu, iż obliczenia lokalizacji dokonywane są na urządzeniu

lokalizowanym. Jest to problemem gdy osoba, którą chcemy odnaleźć jest nieprzytomna i nie jest w stanie pomóc w zlokalizowaniu się. Taka sytuacja może wystąpić podczas katastrof związanych z budynkami np. podczas pożaru czy zawalenia.

Systemy GNSS mogą być wspomagane poprzez inne podsystemy, w celu określenia dokładniejszej lokalizacji. Punkty referencyjne nie muszą być globalne, gdyż możliwe są też ich lokalne odpowiedniki. Systemy sieci komórkowej są w stanie lokalizować telefony za pomocą swoich stacji bazowych. Siła sygnału lub czas przyjścia danych z terminala do pobliskich stacji bazowych może posłużyć jako dane do lokalizacji. Popularną metodą lokalizacji w sieciach komórkowych jest triangulacja. Metoda ta opiera się o zebranie siły sygnału z trzech punktów referencyjnych w celu wyznaczenia pozycji. Każdy punkt referencyjny pozwala na lokalizację w dodatkowym wymiarze, więc trzy terminale referencyjne są tu minimum. Kolejne stacje bazowe mogą wpłynąć pozytywnie na dokładność lokalizacji. Taki system może więc dodać systemowi globalnemu dokładności w konkretnej lokalnej przestrzeni. Fuzja danych GNSS z danymi z wielu systemów sieci komórkowej mogłaby więc posłużyć poprawieniu dokładności GNSS.

Takie rozwiązania mogą wciąż nie być wystarczające gdy w grę wchodzą niejednorodności medium transmisyjnego. Fale elektromagnetyczne w niejednorodnych ośrodkach są poddawane są wszelakiej gamie efektów, takich jak odbicia i tłumienie. Takie niejednorodności pojawiają się gdy chcemy dotrzeć z falą do wnętrz budynków. Dodatkowo jest też możliwe, iż dany budynek zawiera elementy metalowe w swojej konstrukcji. Metal na drodze fali elektromagnetycznej zachowuje się jak idealny przewodnik, tj. całkowicie odbija falę padającą. Taki element może całkowicie zaburzyć proces lokalizacji.

Rozwiązaniem problemu dokładności lokalizacji wewnątrz budynków mogą być systemy lokalizacji rozmieszczone lokalnie w tychże budynkach. Dzięki takiemu rozwiązaniu możliwe było by lokalizowanie terminali z dużą dokładnością o ile wyeliminowało by się potencjalne problemy. Fale elektromagnetyczne propagują się wewnątrz budynków inaczej niż w wolnej przestrzeni. Propagacja odbywa się głównie za pomocą odbić, co skutkuje rozdzieleniem fali na wiele wiązek odbitych od różnych ścian. Powoduje to występowanie propagacji wielodrogowej, gdzie każda odbita fala dotrze do odbiornika w innym czasie i będzie charakteryzować się inną mocą odebraną. Wprowadza to pewną niejednoznaczność do pomiarów siły sygnału lub czasu jego przybycia.

System lokalizacji wewnątrzbudynkowej może działać na podobnych zasadach co poprzednie wymienione systemy. Przy pomocy stacji referencyjnych, możliwa jest triangulacja i wyznaczenie pozycji terminala ruchomego, za pomocą sygnałów całej sieci. Przedsięwzięcie stworzenia takiego systemu może być kosztowne ze względu na dużą ilość punktów referencyjnych jakiej należałoby użyć. Rozwiązaniem dla problemu skali mogłoby być użycie punktów referencyjnych, które już znajdują się w większości budynków w dzisiejszych czasach, a mianowicie AP (ang. *Access Point)* Wi-Fi (ang. *Wireless Fidelity*). To rozwiązanie może mieć

jednak ograniczenie w postaci małej ilości punktów referencyjnych, co musiałoby być w jakiś sposób skompensowane.

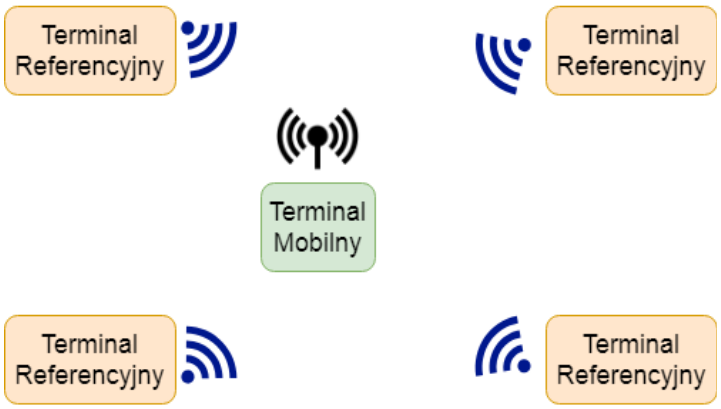
Przykładem ominięcia tego ograniczenia może być użycie systemu zaprezentowanego w [4]. System wykorzystuje jeden węzeł z anteną inteligentną ESPAR (ang. Electronically Switched Parasitic Array Radiator) do lokalizowania terminali mobilnych. Antena taka jest w stanie skanować przestrzeń w 360 stopniach, co umożliwia zmniejszenie zapotrzebowania na stacje bazowe. Rozwiązanie to zostało do tej pory oparte na protokole IEEE 802.15.4. W tej pracy zostanie stworzony podobny system, ale oparty o ogólnodostępną sieć Wi-Fi.

Prezentowany projekt będzie skupiał się na stworzeniu systemu lokalizacji używającego jedynie pojedynczego AP Wi-Fi. W celu zwiększenia ilości danych z jednego AP, zostanie użyta antena inteligentna ESPAR. Problem odbić natomiast zostanie rozwiązany przy zastosowaniu metod *fingerprintingu*. Celem pracy jest stworzenie w pełni funkcjonalnego urządzenia wbudowanego zintegrowanego z anteną ESPAR, które będzie w stanie rejestrować urządzenia pracujące w systemie Wi-Fi i wyznaczać ich pozycje.

2. SYSTEMY LOKALIZACJI BAZUJĄCE NA POMIARZE SIŁY SYGNAŁU

Lokalizacja urządzeń opiera się o wydobywanie informacji z sygnałów otrzymywanych od mobilnych terminali. Urządzenia te mogą być lokalizowane za pomocą pomiaru czasu (TOA – ang. Time Of Arrival, TDOA – ang. Time Difference Of Arrival), kąta (AOA – ang. Angle Of Arrival), fazy nośnej lub siły sygnału (RSS – ang. Received Signal Strength). Pomiary kąta i czasu są złożone obliczeniowo i wymagają skomplikowanej części sprzętowej. Pomiary siły sygnału najczęściej nie mają tych wymagań, przez co są najprostsze do zaimplementowania [5]. Z tego powodu większość wewnątrzbudynkowych systemów lokalizacji opiera się o pomiar RSS.

Siła sygnału (RSS) jest poziomem mocy, którym charakteryzuje się sygnał elektromagnetyczny. Większość urządzeń bezprzewodowych posiada funkcjonalność pomiaru RSSI (ang. Received Signal Strength Indicator), czyli indykatora mocy sygnału. Informacja o RSSI mieści się na jednym bajcie, przyjmując wartości -128 do 127 dBm. Ze względu na małe moce urządzeń wbudowanych RSSI zawsze przyjmuje wartości ujemne. Funkcjonalność pomiaru RSSI jest potrzebna urządzeniom w celu zbadania środowiska w jakim się znajdują. Na przykład pomiar RSSI może być użyty do wykrycia najlepszego AP Wi-Fi, którego sygnały mają większą moc niż inne. RSS zależy od dystansu jaki pokonała fala [6], tak więc moc świadczy o bliskości AP i jakości jego sygnału.



Rys. 2.1. Schemat prostego systemu lokalizacji

Korzystając z informacji o RSSI możliwe jest więc znalezienie szacunkowej odległości między komunikującymi się terminalami. Umożliwia to przeprowadzenie lokalizacji. Na rys. 2.1 przedstawiono prosty schemat systemu lokalizującego. Terminal mobilny jest urządzeniem lokalizowanym. Terminale referencyjne mają ustaloną pozycję i służą jako punkty odniesienia dla procesu lokalizacji. Terminal mobilny wysyła sygnały, które docierają do terminali referencyjnych. Dzięki temu, mogą one pomierzyć moc sygnału otrzymanego od terminala

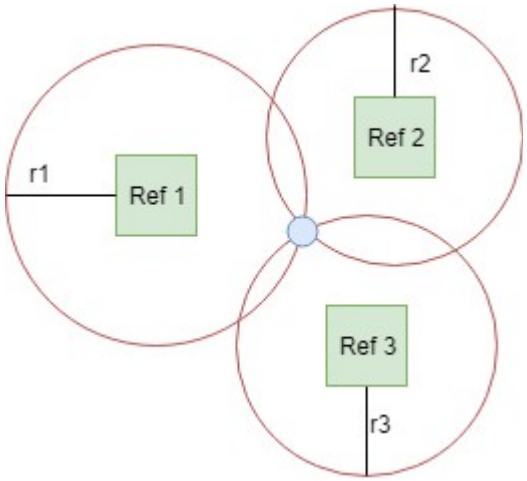
mobilnego. Pomierzone RSSI może służyć jako miara odległości między terminalem stacjonarnym i mobilnym. Do lokalizacji potrzeba jednak algorytmu, który zdefiniuje jak należy mierzyć moc i jak oszacować za jej pomocą pozycję terminala mobilnego. Istnieje kilka popularnych algorytmów.

2.1. Proximity

Proximity to najprostszy algorytm lokalizacji. Daje informację o tym czy dane urządzenie znajduje się w zasięgu sieci. Do jego realizacji używa się małych urządzeń transmitujących, tzw. *beaconów*. *Beacon* transmituje ciągle informacje o sobie do pobliskich urządzeń mobilnych. Każdy *beacon* jest w stanie wykryć jakie urządzenia są w pobliżu w danym momencie [7]. Metoda ta nie nadaje się do dokładnej lokalizacji urządzeń. Jej koszt zależy od ilości zastosowanych nadajników, więc jest również proporcjonalniy do wielkości objętego obszaru.

2.2. Trilateracja

Trilateracja jest algorytmem opierającym się o pomiary RSS z trzech stacji bazowych. Sygnał EM w powietrzu traci swoją moc wraz z przebytą odległością. Możliwe jest więc znalezienie zależności między mocą sygnału a przebytym przez niego dystansem [6]. Pomiar RSS z każdej stacji bazowej jest więc równoważny z odległością szukanego terminala od danej stacji bazowej. Dla każdej stacji bazowej można wyznaczyć okrąg na którym może znajdować się terminal mobilny. Ostateczna pozycja terminala znajduje się na skrzyżowaniu tych okręgów, tak jak widać to na rys 2.2. Metoda ta jest stosunkowo łatwa w realizacji. Wewnątrz budynków może być jednak mało dokładna ze względu na duże wahania RSS.



Rys. 2.2. Przykład trilateracji.

2.3. Fingerprinting

Obecnie najpopularniejszym algorytmem lokalizacji jest RF *fingerprinting*. Metody oparte o *fingerprinting* wykorzystują zestaw uprzednio pomierzonych danych do obliczania lokalizacji. Można wyróżnić 2 fazy *fingerprintingu*: *online* i *offline*. Faza *offline* jest fazą kalibracji algorytmu. W tej fazie zbierane są dane porównawcze dla algorytmu, tzw. *fingerprinty*. Rodzaj danych zależy od zastosowanego systemu i metody lokalizacji. Zakłada się jednak, że za każdym razem zbiera się ten sam zestaw danych tym samym sposobem. Pomiary te wykonywane są na konkretnej dyskretnej dziedzinie, np na punktach w fizycznej przestrzeni 2D (dwuwymiarowej). Przestrzeń z pobranymi w fazie offline wartościami dla każdego punktu nazywamy mapą *fingerprintingu* lub mapą *fingerprintów*.

Drugą fazą *fingerprintingu* jest faza *online*. Korzystając z zawartych w mapie *fingerprintów* danych można oszacować podobieństwo nowego pojedynczego pomiaru do pomiarów przeprowadzonych w fazie *offline*. W ten sposób można wyznaczyć najbardziej prawdopodobne miejsce, w którym został wykonany pomiar.

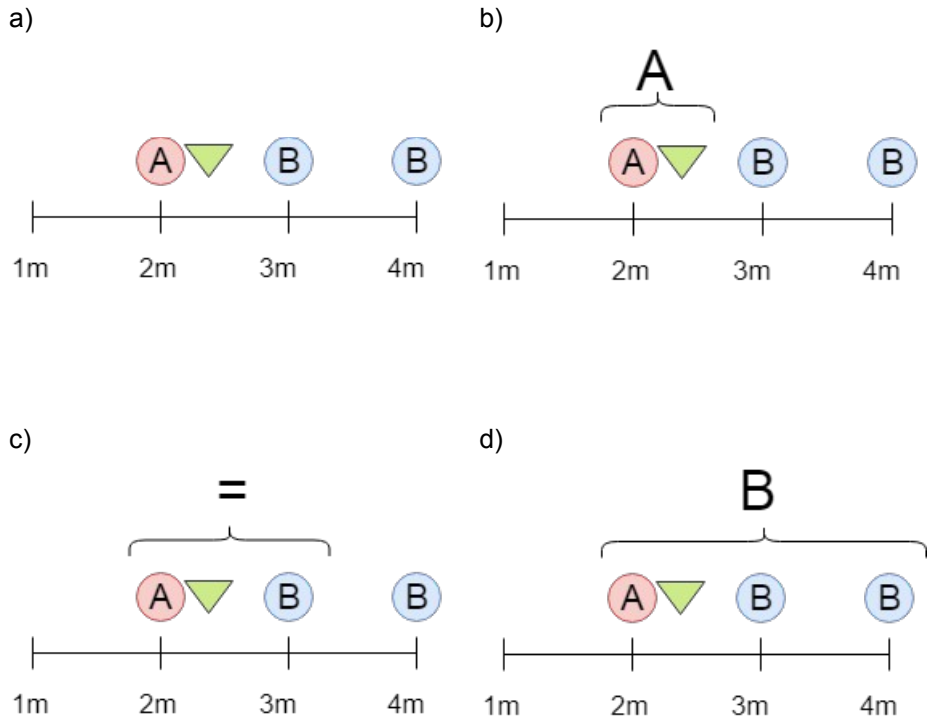
Metody fingerprintingu są najczęściej używane jeśli chodzi o lokalizację wewnątrz budynków. Środowisko o wielu odbiciach utrudnia zastosowanie metod takich jak trilateracja, ze względu na to, że droga między terminalami nie jest linią prostą. *Fingerprinting* nie ma tego ograniczenia, gdyż lokalizacja opiera się o dopasowanie do wcześniej pomierzonych wyników.

2.4. Algorytm k-Nearest Neighbours

Jednym z algorytmów wykorzystujących *fingerprinting* jest algorytm „k najbliższych sąsiadow” lub k-NN (ang. k-Nearest Neighbours). Algorytm k-NN polega na klasyfikacji pomiaru online do jednej z dostępnych klas, na podstawie jednego lub wielu parametrów [8].

Rys. 2.3 przedstawia jednowymiarowy przypadek zastosowania algorytmu. Trójkąt oznacza wartość otrzymaną w etapie *online* (zwane dalej pomiarem), a kółka wartości wyznaczone w fazie *offline* (zwane dalej wartościami referencyjnymi). Rolą algorytmu jest klasyfikacja pomiarów z etapu online do jednej z klas: *A* lub *B*. Klasy to zbiory, do których warunki przynależności są odgórnie wyznaczone w fazie offline. W przykładzie można założyć dwie klasy rozróżniające punkty w zależności od tego jak daleko są od początku osi: blisko (A) i daleko (B). Klasyfikacja pomiaru polega więc na zdecydowaniu czy jest on „daleko” czy „blisko”.

W ogólnym przypadku pomiary są klasyfikowane poprzez to jak blisko wartości referencyjne danej klasy znajdują się względem pomiaru. Decyzję tą definiuje się poprzez wprowadzenie parametru *k*, czyli ilości najbliższych sąsiadów. Najbliższymi sąsiadami nazywamy wartości najbliższe pomiarowi. Algorytm wybiera więc *k* najbliższych sąsiadów i sprawdza ile wartości danej klasy znajduje się w zbiorze. Pomiar zostaje zaklasyfikowany do tej klasy, której wartości w zbiorze jest najwięcej.



Rys. 2.3. Jednowymiarowy przykład algorytmu k-NN
a) Przedstawienie sytuacji, b) k=1, c) k=2, d) k=3

Rys. 2.3b przedstawia sytuację gdy *k*=1. Najbliższym sąsiadem pomiaru jest punkt z klasy *A*, tak więc pomiar zostanie zakwalifikowany do klasy *A*. Rys. 2.3c przedstawia sytuację gdzie ilość sąsiadów jest parzysta, mianowicie *k*=2. Występuje tutaj remis, więc pomiar nie może zostać jednoznacznie zakwalifikowany do żadnej z klas. Dla prostych przypadków, by uniknąć remisów stosuje się nieparzyste wartości *k*. Dla *k*=3, na Rys. 2.3d, pomiar zostaje zaklasyfikowany do klasy *B*, bo w zbiorze 3 najbliższych sąsiadów znalazły się dwie wartości z klasy *B*, czyli w tym wypadku większość.

W podejściu algorytmicznym można dla każdego kółka obliczyć różnicę *D* według wzoru 2.1 i znaleźć *k* najmniejszych wartości *D*. Im mniejsza wartość *D*, tym bliżej pomiar znajduje się danej wartości referencyjnej.

$$D = |S_t - S| \tag{2.1}$$

gdzie:

S_t – wartość parametru w fazie online,

S – wartość parametru w fazie offline.

Analiza *fingerprintów* k-NN znajduje swoje zastosowanie w lokalizacji bezprzewodowej. Klasyfikacja odnosi się w tym przypadku do położenia, gdzie klasami są punkty w przestrzeni. Wartością porównywaną jest siła sygnału odebrana w danej stacji bazowej. Faktycznie do lokalizacji używa się grupy stacji bazowych, dzięki czemu do dyspozycji jest więcej danych. W takim wypadku problem z jednowymiarowego przeistacza się w *K* wymiarowy, gdzie *K* to ilość stacji referencyjnych. Wielowymiarowy uogólniony odpowiednik wzoru 2.1, to wzór 2.2.

$$D_n = \sqrt{\sum_{i=1}^{K} (S_{Ti} - S_{i,n})^2} \tag{2.2}$$

gdzie:

S_T – RSSI pomierzone w fazie online,

S – RSSI pomierzone w fazie offline,

K – ilość stacji bazowych,

n – indeks punktu pomiarowego.

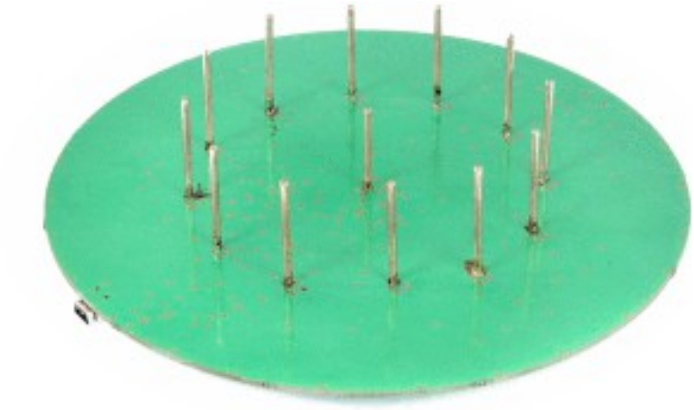
Wartość *D* jest obliczana dla każdego punktu z fazy *online*. Długość obliczeń w fazie *online* zależy więc wprost proporcjonalnie od ilości punktów w fazie *offline*. Im mniejsza wartość *D* tym dane *online* są bardziej podobne do danych *offline* danego punktu. Do estymacji wyniku wybiera się *k* najmniejszych wartości *D*. Punkty z najmniejszymi wartościami *D* używane są do estymacji współrzędnych *x* i *y* punktu wynikowego. Najbliższych sąsiadów (punktów w przestrzeni) można użyć do standardowej dyskretnej klasyfikacji, jak pokazano w przykładzie na rys 2.3. Ciągłość dziedziny wynikowej pozwala nam jednak na uzyskanie wyniku poprzez uśrednienie najbliższych sąsiadów.

3. WYKORZYSTANIE ANTENY ESPAR W LOKALIZACJI RF FINGERPRINTING

3.1. Antena Espar

Antena ESPAR jest typem anteny inteligentnej o przełączanej wiązce. Oznacza to, że charakterystyka promieniowania anteny może być zmieniana w trakcie jej działania. W przypadku anteny ESPAR zmiana polega na obrocie wiązki głównej promieniowania wokół osi anteny.

Antena składa się z prętów przylutowanych do płytki drukowanej. W środku anteny umieszczony jest monopol, pełniący rolę elementu aktywnego. Podobne monopole tworzą elementy pasywne anteny. Są one umieszczone na okręgu , którego środkiem jest element aktywny. Pod spodem anteny znajduje się część elektroniczna, stanowiąca interfejs między częścią mikrofalową i cyfrową urządzenia. Konkretny interfejs i parametry takie jak np. ilość elementów pasywnych zależą od implementacji anteny.

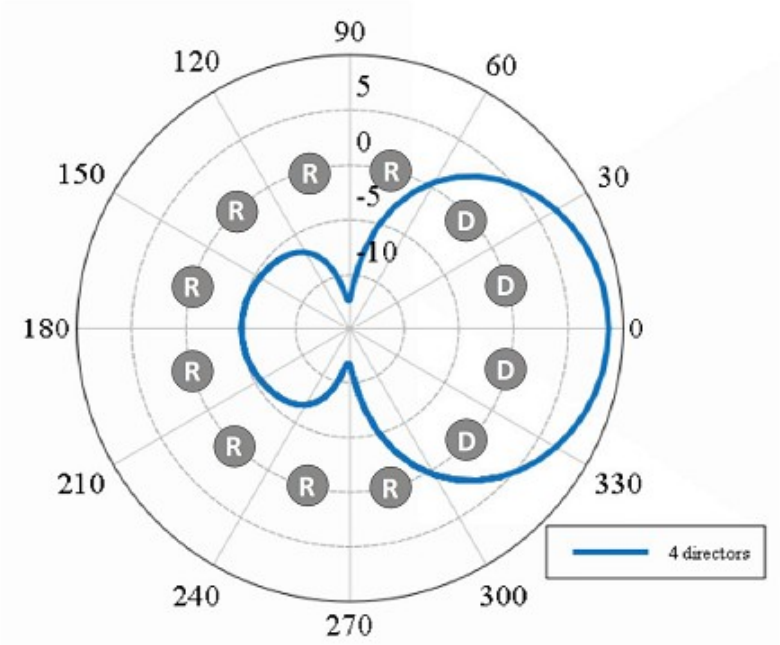


Rys. 3.1. Antena ESPAR

Użyta antena ESPAR przedstawiona jest na rys 3.1. Dysponuje ona jednym elementem aktywnym i dwunastoma elementami pasywnymi. Element aktywny jest podłączony za pomocą złącza SMA.

Każdy element pasywny ma swój klucz mikrofalowy typu SPDT (ang. *single-pole double-throw*). Klucz taki decyduje do czego podłączony jest element aktywny, umożliwiając mu przyjęcie jednego z dwóch stanów. Pierwszym stanem jest stan zwarcia do masy, czyli stan reflektora. Reflektor jest elementem odbijającym fale EM. Drugim stanem jest stan rozwarcia, czyli stan direktora. Direktor jest elementem który nie jest przeszkodą dla fali, która przez niego przechodzi [4].

Zmiana konfiguracji kluczy anteny, tj. ustawienie stanu każdego z prętów na direktor lub reflektor, pozwala na kształtowanie charakterystyki promieniowania anteny. Najbardziej podstawową konfigurację tworzy rząd sąsiadujących direktorów wraz z rzędem sąsiadujących refelktorów. Ilość direktorów w rzędzie ustalono na 4. Taka konfiguracja tworzy charakterystykę z wyraźną wiązką główną, jak widać na rys 3.2. Zmianę kierunku wiązki głównej uzyskuje się poprzez przesunięcie rzędu direktorów o jedną pozycję. Ostatecznie możliwe jest 12 podstawowych konfiguracji anteny, których wiązki główne są przesunięte względem siebie o 30 stopni.



Rys. 3.2. Przykładowa konfiguracja kluczy anteny wraz z przybliżonym kształtem charakterystyki tejże konfiguracji. D oznacza direktor, a R reflektor. Ilustracja z [9].

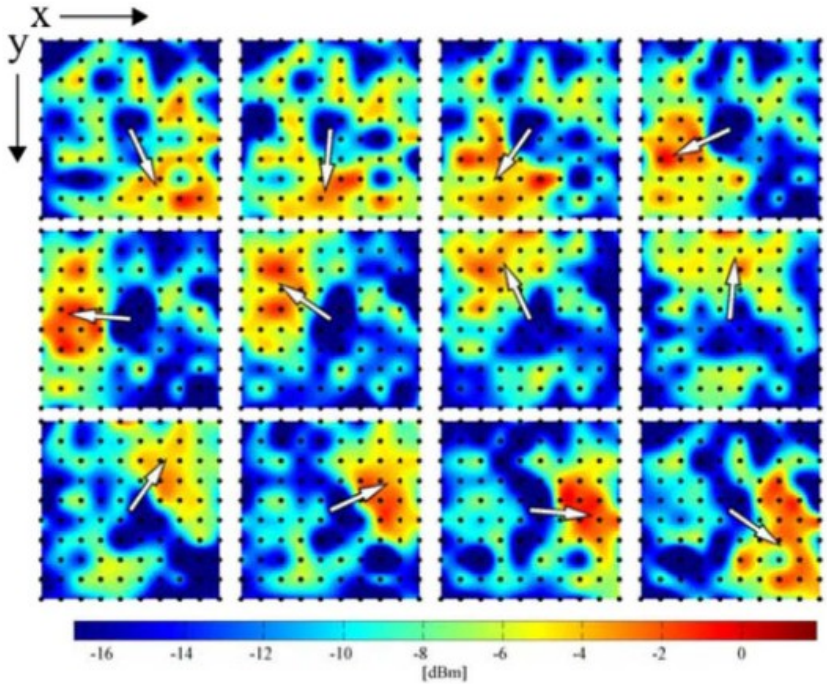
Konfigurację anteny można zapisać w postaci wektorowej **V**=*[0,0,0,0,1,1,1,1,1,1,1,1]*, gdzie każda wartość odpowiada stanowi jednego klucza mikrofalowego. Zakładamy że wartość 0 oznacza direktor, a 1 reflektor. Zmiana konfiguracji anteny jest tu równoważna z cyklicznym przesunięciem wartości w wektorze o jedną pozycję [4].

Funkcjonalność przełączania wiązki implementuje część cyfrowa anteny, której sercem jest mikrokontroler z rodziny *Stm32*. Każdy klucz mikrofalowy jest sterowany elektronicznie z pinów mikrokontrolera. Urządzenie ma wyprowadzony interfejs UART (ang. Universal Asynchronous Receiver Transmitter) w postaci konwertera na interfejs USB. Służy on komunikacji z mikrokontrolerem, a co za tym idzie, ustawianiem konfiguracji.

3.2. Fingerprinting z wykorzystaniem anteny ESPAR

Fingerprinting wymaga określonego formatu pomiarów zarówno w fazie *online* i *offline*. Można zdefiniować formę pojedynczego pomiaru, jaka będzie iteracyjnie powtarzana dla obu faz. Dla standardowego systemu z *K* stacjami bazowymi pomiar powinien być zdefiniowany jako uzyskanie przez system jednej wartości RSS z każdej stacji. Można zatem zdefiniować format jako wektor $\mathbf{V}=[v_1,v_2,.....v_K]$.

Lokalizacja odbywa się zazwyczaj w przestrzeni 2D, czyli na płaszczyźnie. Dla docelowego obszaru definiuje się siatkę punktów. Na tej siatce w fazie *offline* definiuje się mapę *fingerprintingu*, czyli zestaw pomiarów referencyjnych dla każdego punktu. Zgodnie ze zdefiniowanym formatem pomiaru, mapa każdego punktu to zestaw *K* wartości RSS, pomierzonych w fazie *offline*. W fazie online, nowy pomiar K wartości RSS jest porównywany z mapą *fingerprintingu* zgodnie ze wzorem 2.2.



Rys. 3.3. Mapa fingerprintów dla każdej z 12 konfiguracji anteny. Ilustracja z [4].

Lokalizacja k-NN może być przeprowadzona analogicznie za pomocą anteny ESPAR. W przypadku *K* stacji bazowych ze standardowym *frontendem* antenowym, każda stacja oznacza jedną wartość referencyjną dla algorytmu k-NN. *K* stacji bazowych może być zastąpione przez *K* pomiarów przeprowadzonych za pomocą anteny ESPAR, gdzie dla każdego unikalnego pomiaru stosujemy inną unikalną charakterystykę anteny. Każda charakterystyka anteny odpowiada konkretnej konfiguracji kluczy mikrofalowych. W ten

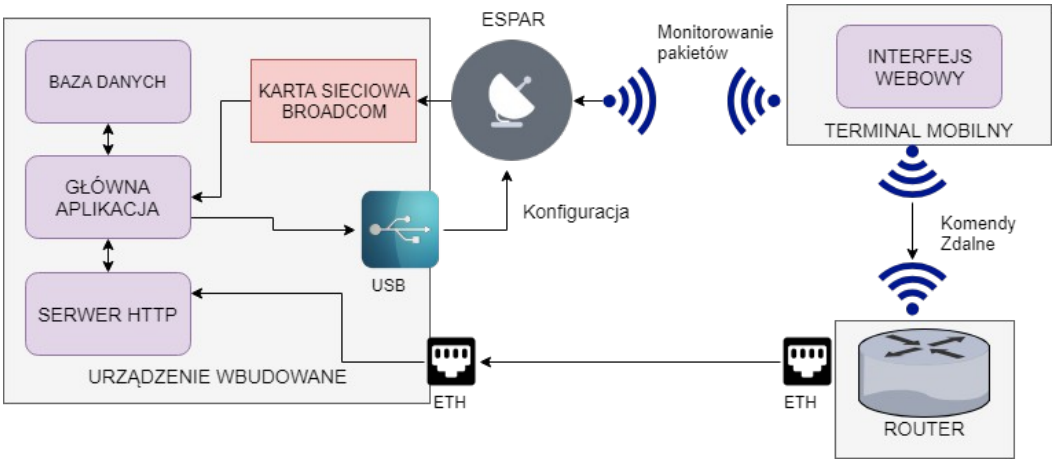
sposób każda z unikalnych konfiguracji anteny daje jedną wartość referencyjną dla algorytmu.

Przykład zastosowania anteny ESPAR w lokalizacji k-NN przedstawiono w [4]. Na siatce docelowego pomieszczenia stworzono mapy fingerprintów odebranego od próbnika RSS. Rys 3.3 przedstawia rozkład odebranego RSS dla dwunastu konfiguracji anteny ESPAR w każdym punkcie pomieszczenia. W danym punkcie dla każdej konfiguracji zmienia się moc odebrana od próbnika. Na dodatek zestaw pomiarów dla każdej konfiguracji prezentuje inne rozłożenie wartości RSS, co miałoby miejsce podczas zastosowania 12tu osobnych odbiorników. Oznacza to równowartość obu metod, w odniesieniu do ilości otrzymywanych danych.

Pod względem używanego sprzętu, zamiast n zwykłych stacji bazowych użyta zostaje jedna z anteną ESPAR. Dzięki temu możliwe jest zmniejszenie kosztu systemu lokalizacji. Rozwiązanie to minimalizuje również problemy połączenia K stacji bazowych, oraz ich synchronizacji.

4. ARCHITEKTURA SYSTEMU

W celu sprawdzenia możliwości lokalizacji wewnątrz budynków opartej o k-NN i antenę ESPAR, należy stworzyć kompletny system oparty o tą technologię. Rys 4.1 przedstawia pełną strukturę sprzętu i oprogramowania projektowanego systemu. System składa się z dwóch urządzeń, lokalizatora z anteną ESPAR i terminala mobilnego, którym może być dowolne urządzenie. Dodatkowo system uzyskuje dostęp do sieci dzięki niezależnemu routerowi Wi-Fi. Schemat przedstawia zarówno bloki software (gładkie prostokąty), jak i elementy sprzętowe (pozostałe ikony i bloki).



Rys. 4.1. Schemat oprogramowania i sprzętu

4.1. Sprzęt

Podstawą systemu musi być system wbudowany, posiadający odpowiednio wysoką moc obliczeniową i obsługujący zarówno sieć *ethernet* jak i Wi-Fi. Urządzenie to będzie wykonywać wszystkie główne funkcjonalności systemu. Wymagania te spełniają jedynie komputery jednoukładowe, tak więc do projektu wybrano komputer Intel NUC DE3815TYBE.

Komputer jednoukładowy wymaga systemu operacyjnego używającego małej ilości zasobów. Systemy takie kompensują tą cechę ograniczoną funkcjonalnością. Projekt wymaga jednak dostępu do przewodowego i bezprzewodowego internetu, co jest kluczowe przy wyborze dystrybucji. Kolejnym wymaganiem jest nieskomplikowany niskopoziomowy interfejs dla języka C. Systemy z rodziny Linux dysponują takimi bibliotekami. Wymagania te są spełnione przez system *Lubuntu 17*. Użyto wersji systemu ze środowiskiem graficznym w celu ułatwienia debugowania.

Dostęp do sieci bezprzewodowej wymaga zastosowania karty Wi-Fi. Wybrana karta musi umożliwiać monitorowanie ruchu sieciowego. W tym celu karta musi wspierać tryb monitora. Jest to tryb, w którym karta odbiera wszystkie pakiety z sieci, a nie tylko te które są skierowane bezpośrednio do niej. Trzeba mieć również na uwadze dostępność

oprogramowania dla szukanego sprzętu. Komputer wbudowany musi mieć sterowniki do użytej karty. Karta *Broadcom BCM943224HMS* spełnia te wymagania, więc została wykorzystana w projekcie.

4.2. Interfejsy między urządzeniami

Komputer musi być połączony z anteną ESPAR za pomocą dwóch interfejsów. ESPAR zastąpi normalną antenę Wi-Fi komputera, poprzez podłączenie jej do złącza RF karty Wi-Fi. Dodatkowo, komputer będzie musiał sterować kluczami anteny poprzez złącze USB za pomocą interfejsu UART.

Urządzenia systemu są niezależnymi jednostkami, tak więc będą komunikować się przy pomocy sieci lokalnej. Zadaniem routera będzie przydzielanie adresów urządzeniom i stanowienie węzła tranzytowego komunikacji. Komputer wbudowany będzie podłączony przewodowo do routera, natomiast telefon będzie komunikował się poprzez Wi-Fi. Komunikacja komputera wbudowanego z telefonem będzie odbywać się dwojako. Wszelkie komendy systemowe będą przesyłane zwykłym łączem internetowym, za pomocą protokołu TCP (ang. Transmission Control Protocol). Monitoring pakietów będzie odbywał się niezależnie od wspomnianego łącza. Należy zwrócić uwagę na fakt, iż najlepszym rozwiązaniem byłoby zastosowanie trybu AP karty sieciowej by telefon lokalizowany łączył się bezpośrednio do komputera. W ten sposób można by wyeliminować potrzebę korzystania z zewnętrznego routera Wi-Fi. Oznaczałoby to jednak zrezygnowanie z trybu monitora, co skutkowałoby brakiem możliwości otrzymywania pakietów od urządzeń spoza własnej sieci lokalnej AP. Istnieją tryby pozwalające łączyć te funkcjonalności, lecz zrezygnowano z nich w projekcie by nie komplikować zadania.

4.3. Oprogramowanie

Główną częścią oprogramowania będzie aplikacja napisana w języku C. Aplikacja będzie implementować funkcjonalność monitora sieci oraz obsługę bazy danych. Monitor będzie zapewniał interfejs dla sterownika karty Wi-Fi, pobierając dane o wszystkich pakietach w sieci. Baza danych natomiast będzie przechowywać dane fingerprintingu potrzebne dla algorytmu k-NN.

Sterowanie systemem powinno być proste i wygodne, by przyspieszyć jak tylko się da wykonywanie testów. Urządzeniem lokalizowanym jest domyślnie telefon, lecz mógłby nim być dowolny terminal Wi-Fi. Z tego powodu, by nie tworzyć osobnej aplikacji dla każdego możliwego terminala, zdecydowano się użyć technologii webowych. Strony internetowe są obsługiwane tak samo niezależnie od sprzętu, co niweluje wymieniony problem. Wadą takiego rozwiązania może być poziom trudności programowania webowego.

Fakt zastosowania technologii webowych dodaje do schematu dodatkowe bloki. Jednym z nich jest serwer HTTP (ang. HyperText Transfer Protocol), odpowiedzialny za

udostępnianie strony internetowej urządzeniom zewnętrznym. Drugim jest aplikacja webowa, wykonywana przez przeglądarkę urządzenia lokalizowanego. Dodatkowo będzie potrzebny interfejs między główną aplikacją a serwerem HTTP.

5. REALIZACJA SYSTEMU

W celach przetestowania możliwości teoretycznego systemu zostanie stworzony projekt implementujący go. System docelowo ma zbierać pomiary w celu stworzenia mapy *fingerprintingu* oraz umożliwić przeprowadzenie testów dokładności lokalizacji.

5.1. Monitor pakietów

Monitor pakietów (ang. *sniffer*) jest programem lub urządzeniem do analizowania ruchu w sieci. Działa on na podstawie stworzenia interfejsu między kanałem i siecią bezprzewodową, a warstwą aplikacji. Za jego pomocą można przechwycić informacje o pakietach w sieci jak i dane, które przenoszą.

Do lokalizacji urządzeń na podstawie ruchu w sieci potrzebna jest odpowiednia informacja - opóźnienie pakietu lub siła jego sygnału [10]. Jedną z tych informacji (RSSI) można uzyskać z samej karty Wi-Fi, a dokładniej z jej sterownika. Korzystając z tej wiedzy, projekt zrealizowano początkowo na komputerze PC, z wykorzystaniem karty jako części sprzętowej.

Informację o RSSI otrzymujemy dla większości pakietów jakie urządzenie odbiera. To jakie pakiety są przechwytywane przez *sniffer* zależy od trybu karty Wi-Fi. Standardowym trybem jest tryb *managed*, który obserwuje tylko ruch od oraz do samego urządzenia. W celu przechwycenia ruchu z innych sieci konieczne jest włączenie trybu *monitor* lub *promiscuous*. Wybrana karta obsługuje te tryby.

5.1.1.Przegląd gotowych monitorów ruchu w sieci bezprzewodowej

Drugim krokiem po ustaleniu sprzętu jest wybór oprogramowania. Istnieje wiele gotowych snifferów takich jak *Tcpdump* czy *Wireshark.* W tych rozważaniach sprawdzono oba programy jako dwa z najbardziej popularnych. Są to aplikacje konsolowe, tak więc do wydobycia danych w czasie rzeczywistym użyto *screenscrapingu*, czyli zbierania danych poprzez *parsowanie* komunikatów konsolowych.

Dla obydwu *snifferów* zaimplementowano podstawowe procedury zbierania danych do lokalizacji. Docelowy system w celu wykonania lokalizacji musi pobrać siłę sygnału obiektu tyle razy, ile ustalonych konfiguracji kierunkowych ma antena. *Sniffer* na danej konfiguracji anteny powinien pobrać minimalnie jeden pakiet, po czym antena powinna zmienić kierunek wiązki.

Sprawdzone *sniffery* pobierają pakiety do bufora, który jest opróżniany po określonym czasie. Na danej konfiguracji anteny otrzymuje się więc grupę pakietów z bufora pobieranych np. przez 1 sekundę. Nie jest więc możliwe pobieranie pakietów dla danej konfiguracji anteny przez czas krótszy niż czas buforowania.

Biorąc pod uwagę te ograniczenia sprawdzono kluczowe dla projektu parametry obu programów. *Sniffer Tcpdump* ma mało opcji odnośnie kształtowania komunikatów konsolowych przez co konieczne jest parsowanie dużej ilości danych. W dodatku czas buforowania pakietów

jest długi i wynosi sekundę. *Wireshark* (a dokładniej jego konsolowa wersja - *tshark*) ma ma natomiast dużo opcji odnośnie kształtowania komunikatów, a jego czas buforowania wynosi ok. ⅓ sekundy.

Pomimo łatwości implementacji rozwiązania z gotowym *snifferem*, jest ono nieoptymalne obliczeniowo i czasowo. Jeśli założyć 12 kierunków anteny i 0.33 sekundy buforowania (dla *tshark*), otrzymuje się 4 sekundy na wykonanie pełnego pomiaru do lokalizacji. System z takim czasem pomiaru działałby bardzo wolno. Czas ten jest o wiele bardziej kluczowy gdy wykonuje się kalibrację systemu polegającą na wykonaniu wielu pomiarów w celu uśrednienia, lub gdy chcemy lokalizować wiele urządzeń na różnych kanałach.

Tabela 5.1. Zestawienie parametrów badanych monitorów sieci

Sniffer	Czas buforowania	Opcje wyjścia konsoli
Tcpdump	1 s	ograniczone
Wireshark (tshark)	0.33 s	rozbudowane
Potencjalny własny sniffer	Brak lub minimalny	nie dotyczy

Podsumowując, czas buforowania pakietów jest wąskim gardłem dla projektu, który używałby gotowych *snifferów*. Trzeba zaznaczyć, iż jest to ograniczenie podyktowane przez dany *software*, a nie *hardware*. Obydwa programy posiadają opcje konfiguracji lecz zmiana czasu buforowania pakietów nie jest dostępna. Najbardziej obiecującym rozwiązaniem w tym wypadku jest więc zmiana oprogramowania tak by pasowało ono do projektu, lub napisanie sniffera od podstaw. Zdecydowano się na użycie drugiej opcji. Parametry sprawdzonych snifferów zaznaczono w tabeli 5.1, razem z wymaganiami dla własnej aplikacji.

5.1.2.Opracowanie szybkiego monitora ruchu sieci bezprzewodowej

Obydwa wymienione sniffery jak i wiele innych opiera się o bibliotekę *libpcap*. Biblioteka ta stanowi API (ang. Application Programming Interface) do pobierania pakietów z karty Wi-Fi w formie bajtów, czyli działa powyżej warstwy łącza. Na potrzeby projektu stworzono własną implementację *sniffera* na tej bibliotece w celu wyeliminowania czasu buforowania. Takie rozwiązanie dostosowuje *sniffer* do konkretnego przypadku i zwiększa wydajność programu.

Warstwa łącza jest najniższą warstwą modelu TCP/IP (równoważną warstwom 2 i 1 modelu OSI), która zajmuje się obsługą fizycznego łącza między dwoma urządzeniami. Protokół oparty o warstwę łącza musi m. in. adresować urządzenia po adresie MAC (ang. Media Access Control), czy rozdzielać pakiety z wyższych warstw modelu do ramek [11]. Różne protokoły mogą implementować te ramki w różny sposób.

Pcap obsługuje wiele rodzajów nagłówków warstwy łącza, czyli schematów ramek przesyłanych w najniższej warstwie. W specyfikacji *pcap* noszą one nazwę *LINKTYPE* lub *DLT* [12]. Nagłówek standardu IEEE 802.11 nosi tu nazwę *LINKTYPE_IEEE802_11* i *DLT_IEEE802_11.* Sterowniki Wi-Fi systemu Linux wspierają jednak ramkę 802.11 rozszerzoną

o strukturę *Radiotap* [13], zawierającą dodatkowe informacje o odebranym pakiecie. *Radiotap* jest to standard injekcji i odbioru pakietów dla 802.11 [14]. Typ nagłówka *Radiotap* nosi nazwy *LINKTYPE_IEEE802_11_RADIOTAP* i *DLT_IEEE802_11_RADIO*. Nagłówek *Radiotap* jest niezbędny w projekcie, gdyż stanowi interfejs, z którego można pobrać informację o sile sygnału odebranego pakietu.

Pierwszą funkcjonalnością implementowanego *sniffera* jest inicjalizacja. Funkcją inicjalizującą przechwytywanie jest funkcja *pcap_open_live* z biblioteki *pcap* [15]. Za jej pomocą można ustawić takie parametry jak długość bufora pakietów czy nazwę urządzenia monitorowanego. Istotnym parametrem z punktu widzenia projektu jest czas buforowania (tu *XPCAP_TIMEOUT*), który został ustawiony na 10 ms. Po włączeniu funkcji zwraca ona uchwyt (ang. *handle*) do nowej sesji przechwytu pakietów, rozpoczynając działanie sniffera:

```
handle = pcap_open_live(dev, SNAP_LEN, 1, XPCAP_TIMEOUT, errbuf);
```

Istnieją też inne dodatkowe opcje inicjalizacji. Możliwe jest sprawdzenie rodzaju nagłówka warstwy łącza dla urządzenia monitorowanego [12]. W wypadku tego projektu nagłówek nosi nazwę *DLT_IEEE802_11_RADIO*. Możliwe jest również skompilowanie i ustawienie *kernelowego* filtru pakietów, który przetwarza pakiety zanim zostaną one przekazane użytkownikowi.

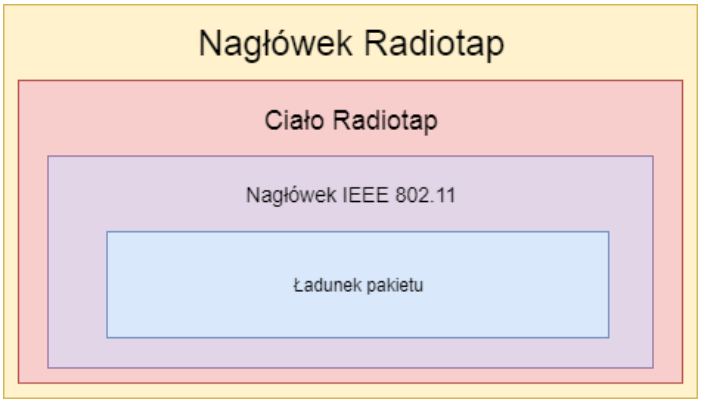
Biblioteka definiuje 3 główne funkcje do odczytu pakietów [16]: *pcap_loop*, *pcap_next_ex* i *pcap_dispatch*. *Pcap_loop* działa w trybie ciągłym i wysyła pakiety do funkcji callback użytkownika, w której odbywa się przetwarzanie danych. Jest to funkcja blokująca. *Pcap_next_ex* kończy działanie, gdy odebrany zostanie pakiet, co czyni tą funkcję nieblokującą. Odebrany pakiet jest we wskaźniku jednego z argumentów. *Pcap_dispatch* działa jak *pcap_loop* ale jest nieblokująca jak *pcap_next_ex* i również kończy działanie przy przyjściu pakietu:

```
pcap_dispatch(handle, 1, xpcap_packet_parser_x, (u_char*)packetstruc);
```

Do projektu użyto funkcji nieblokujących w celu osiągnięcia większej kontroli nad sterowaniem *snifferem pcap*. Funkcje te mają ten sam zestaw argumentów. Pierwszy argument jest uchwytem otrzymanym z funkcji *pcap_open_live*. Drugi oznacza ilość odebranych pakietów która zakończy działanie funkcji. Trzeci argument oznacza nazwę funkcji do której przekazywane będą przechwycone pakiety. Jako argument użyto funkcji parsującej pakiety. Ostatnim argumentem jest kontekst dla funkcji parsującej. Tutaj jest to struktura, do której zapisane będą wszystkie potrzebne do lokalizacji informacje o pakiecie. Funkcje *pcap* po zakończeniu działania przekażą informacje o pakiecie do zapewnionej funkcji callback.

Zamknięcie monitorowania po zakończeniu używania sniffera dokonuje się komendą *pcap_close:*

```
pcap_close(handle);
```



Rys. 5.1. Zagnieżdżona struktura ramki 802.11 z dodanym nagłówkiem *Radiotap*

Funkcja parsująca jest funkcją typu *callback* przekazywaną funkcjom *pcap*. Funkcje tego typu przekazuje się do innych funkcji w formie wskaźnika, by mogły być wywołane na niższym poziomie abstrakcji. Funkcja musi przyjmować ustalone argumenty by poprawnie odebrać dane z tego API:

```
void RSSI_sniffer::xpcap_packet_parser_x(u_char *args, const struct pcap_pkthdr
*header, const u_char *packet)
```

Pierwszym argumentem jest przekazany wcześniej kontekst. Pozostałe dwa argumenty zawierają kolejno wskaźniki do nagłówka pakietu i samego pakietu. Nagłówek *pcap_pkthdr* zawiera informacje o przechwyconym pakiecie, w tym kluczowy dla projektu czas przechwycenia pakietu (*timestamp*) [17]:

```
struct pcap_pkthdr {
        struct timeval ts;          /* timestamp */
        bpf_u_int32 caplen;
        bpf_u_int32 len;
};
```

Odebrany pakiet jest ciągiem bajtów, który wymaga interpretacji. Struktura odebranego pakietu pokazana jest na rys 5.1. Dla nagłówka typu *LINKTYPE_IEEE802_11_RADIOTAP* wyróżniamy 3 struktury. Pierwsze dwie odnoszą się do struktury *Radiotap*. Zawiera ona

dodatkowe informacje o pakiecie, m. in. RSSI. Nagłówek *Radiotap* zawiera informacje o wersji i długości całej struktury [18]:

```
struct ieee80211_radiotap_header {
        u_int8_t        it_version;
        u_int8_t        it_pad;
        u_int16_t       it_len;
        struct s_it_present       it_present;
} __attribute__((__packed__));
```

Najważniejszą częścią nagłówka *Radiotap* jest bitmapa *it_presen*t, która wskazuje, które pola są dostępne w ciele struktury. Na końcu bitmapy znajduje się też flaga rozszerzająca bitmapę w razie jej rozszerzenia w kolejnych wersjach struktury *Radiotap*. Bitmapa ma następujące pola:

```
struct s_it_present {
    int8_t      TSFT_available:1;
    int8_t      FLAGS_available:1;
    int8_t      RATE_available:1;
    int8_t      CHANNEL_available:1;
    int8_t      FHSS_available:1;
    int8_t      DBM_ANTSIGNAL_available:1;
    int8_t      DBM_ANTNOISE_available:1;
    int8_t      LOCK_QUALITY_available:1;
    int8_t      TX_ATTENUATION_available:1;
    int8_t      DB_TX_ATTENUATION_available:1;
    int8_t      DBM_TX_POWER_available:1;
    int8_t      ANTENNA_available:1;
    int8_t      DB_ANTSIGNAL_available:1;
    int8_t      DB_ANTNOISE_available:1;
    int8_t      RX_FLAGS_available:1;
    int8_t      TX_FLAGS_available:1;
    int8_t      RTS_RETRIES_available:1;
    int8_t      DATA_RETRIES_available:1;
    int16_t     padding:13;
    int8_t      EXT_available:1;
} __attribute__((__packed__));
```

Istnieje kilka zasad dotyczących ułożenia pól w ciele struktury *Radiotap* [19]. Pierwszą z nich jest kolejność występowania obecnych pól. Kolejność ta jest taka sama jak kolejność w bitmapie *it_present*. Drugą zasadą jest wyrównanie do wielkości pola. Definiując offset jako odległość w bajtach od początku ciała *Radiotap*, sprawdza się wyrównanie pól. Podczas przypisywania kolejnego pola, offset musi być podzielny przez długość pola. Jeśli nie jest, do ramki wstawione są odstępy wyrównujące. Dla przykładu, jeżeli *offset* jest równy 1, a następne pole według i*t_present* jest wielkości 2, to oznacza że to pole będzie przesunięte o 1 bajt w celu wyrównania. Pola składające się z dwóch pól, jak np. *CHANNEL*, są liczone oddzielnie, czyli muszą być wyrównane do 2 bajtów. Innym faktem o którym należy pamiętać jest odwrotne ułożenie bajtów - *little endian*. Każde pole należy czytać od jego najmłodszego bajtu do najstarszego.

W projekcie najważniejszą informacją, którą potrzeba wydobyć z nagłówka *Radiotap* jest RSSI, znajdujące się pod polem *IEEE80211_RADIOTAP_DBM_ANTSIGNAL*. Jest to wartość ze znakiem, która bezpośrednio wskazuje na moc sygnału względem miliWata [dBm]. Struktura *Radiotap* ma następujące pola:

```
/* radiotap data */
struct ieee80211_radiotap_data {
    u_int64_t     IEEE80211_RADIOTAP_TSFT;
    u_int8_t      IEEE80211_RADIOTAP_FLAGS;
    u_int8_t      IEEE80211_RADIOTAP_RATE;
    u_int16_t     IEEE80211_RADIOTAP_CHANNEL_1;
    u_int16_t     IEEE80211_RADIOTAP_CHANNEL_2;
    u_int8_t      IEEE80211_RADIOTAP_FHSS_1;
    u_int8_t      IEEE80211_RADIOTAP_FHSS_2;
    int8_t        IEEE80211_RADIOTAP_DBM_ANTSIGNAL;
    int8_t        IEEE80211_RADIOTAP_DBM_ANTNOISE;
    u_int16_t     IEEE80211_RADIOTAP_LOCK_QUALITY;
    u_int16_t     IEEE80211_RADIOTAP_TX_ATTENUATION;
    u_int16_t     IEEE80211_RADIOTAP_DB_TX_ATTENUATION;
    int8_t        IEEE80211_RADIOTAP_DBM_TX_POWER;
    u_int8_t      IEEE80211_RADIOTAP_ANTENNA;
    u_int8_t      IEEE80211_RADIOTAP_DB_ANTSIGNAL;
    u_int8_t      IEEE80211_RADIOTAP_DB_ANTNOISE;
    u_int16_t     IEEE80211_RADIOTAP_RX_FLAGS;
    u_int16_t     IEEE80211_RADIOTAP_TX_FLAGS;
    u_int8_t      IEEE80211_RADIOTAP_RTS_RETRIES;
    u_int8_t      IEEE80211_RADIOTAP_DATA_RETRIES;
```

```
} __attribute__((__packed__));
```

Ostatnim nagłówkiem przed danymi w pakiecie jest nagłówek protokołu 802.11 [20]:

```
struct ieee80211_header {
    struct ieee80211_framecontrol    Frame_control;
    u_int16_t                        Duration;
    char                             Address_1[6];
    char                             Address_2[6];
    char                             Address_3[6];
    u_int16_t                        Seq_ctl;
    char                             Address_4[6];
} __attribute__((__packed__));
```

Jego pierwszym elementem jest bitmapa *Frame_control* zawierająca informacje o rodzaju ramki [21]. *Type* definiuje ogólny typ ramki, gdzie wyróżnia się: ramki zarządzania (ang. *Management*), kontrolne (ang. *Control*) i danych (ang. *Data*). *Subtype* definiuje jej przeznaczenie np. *Probe Request* czy *Beacon*. Pozostałe informacje zawarte w bitmapie są mało ważne z punktu widzenia projektu.

```
struct ieee80211_framecontrol {
    u_int8_t    Protocol :2;
    u_int8_t     Type :2;
    u_int8_t     Subtype :4;
    u_int8_t    To_DS :1;
    u_int8_t    From_DS :1;
    u_int8_t      More_frag :1;
    u_int8_t     retry :1;
    u_int8_t    Power_management :1;
    u_int8_t      More_data :1;
    u_int8_t     WEP :1;
    u_int8_t    Order :1;

} __attribute__((__packed__));
```

Nagłówek 802.11 zawiera również do 4 adresów stron komunikujących się. Mogą one oznaczać faktyczne źródło i cel pakietu, ale również adresy odbiornika i nadajnika na danym fragmencie łącza. Ilość dostępnych w pakiecie adresów zależy od typu ramki. Z perspektywy

projektu najważniejszy jest adres aktualnego nadajnika w łączu (TA – ang. *Transceiver Address*). Adres ten jest ważny, gdyż wyznacza on źródło nadające moc, która odczytywana jest jako RSSI w odbiorniku. Bez tej informacji nie jest możliwe stwierdzenie od jakiego urządzenia zebrany został pomiar RSSI, przez co pomiar ten jest nieważny. TA jest niedostępne w niektórych ramkach kontrolnych [20]. Z tego powodu należy sprawdzić typ ramki i zapewnić informację o dostępności TA dla kolejnych części systemu.

Zadaniem parsera ramek jest dopasowanie ramek do otrzymanego ciągu bajtów i odczytanie kluczowych z punktu widzenia projektu wartości. Dane te zostają zwrócone z funkcji parsera w formie struktury *xpcap_packet*:

```
struct xpcap_packet{
  double timestamp;
  char RA[6];
  char TA[6];
  int8_t RSSI;
  int8_t frametype;
  bool TA_Available;
};
```

Struktura zawiera czas przyjścia pakietu i adresy odbiornika i nadajnika. Zmienna *TA_Available* informuje o tym czy pakiet zawierał adres nadawcy TA. *Frametype* to bitmapa określająca typ ramki, przepisana z nagłówka 802.11.

5.1.3. Monitor Pakietów jako część oprogramowania

Wszystkie funkcjonalności *libpcap* potrzebne na rzecz projektu zebrano w jednej klasie *RSSI_Sniffer*. Klasa implementuje funkcje będące API udostępniające wszystkie funkcjonalności *sniffera*. Użytkownik musi jedynie włączać i wyłączać zbieranie danych za pomocą funkcji *open_capture* i *close_capture*, które bezpośrednio wywołują funkcje biblioteki pcap. Przed tym należy jednak zainicjalizować sniffer funkcją *sniffer_setup*. Sama inicjalizacja polega na włączeniu trybu monitora za pomocą funkcji systemowych. Z poziomu języka C, funkcje takie wywołuje się za pomocą funkcji *system*. Inicjalizacja polega na wykonaniu polecenia *"sudo iwconfig <device> mode monitor"*, gdzie "*<device>*" oznacza nazwę bezprzewodowego interfejsu sieciowego na którym sniffer ma monitorować urządzenie. Przed tą operacją trzeba jednak wyłączyć ten interfejs za pomocą "*sudo ifconfig <device> down*" oraz włączyć go na koniec poleceniem "*sudo ifconfig <device> up*".

Korzystanie z API *RSSI_Sniffer* opiera się głównie o użycie nieblokującej funkcji *capture_nonblock*. Funkcja pobiera z bufora jeden z pakietów, które są przechwytywane od

momentu wykonania *open_capture*. Pakiet jest następnie parsowany i ostatecznie funkcja zwraca pakiet w postaci struktury *xpcap_packet*.

Klasa zawiera dodatkową funkcję *capture_actual* do pobrania aktualnego pakietu. Jest ona potrzebna do zdobycia pakietu odebranego po konkretnej chwili czasu. W tym celu porównywany jest czas systemowy pobrany za pomocą *clock_gettime* z biblioteki *time.h* systemu *Linux* z *timestampem* pakietu w strukturze *xpcap_packet*.

Klasa umożliwia również zmianę kanału w karcie Wi-Fi za pomocą polecenia systemowego. Funkcja *switch_channel* wywołuje polecenie "*sudo iw dev <device> set channel <number>*" gdzie „*<number>"* oznacza numer kanału.

Klasa *RSSI_Sniffer* jest wynikiem implementacji własnego sniffera przystosowanym wyłącznie do użytku w omawianym projekcie. Wymienione metody klasy są wystarczające do zapewnienia interfejsu między sprzętem, a pozostałymi częściami aplikacji.

5.1.4. Sztuczny ruch

W celu lokalizacji za pomocą anteny ESPAR urządzenie lokalizowane musi transmitować do sieci duże ilości pakietów. Szybkość pomiarów związana jest z przełączaniem anteny i szybkością działania *sniffera*. Do tego dochodzi jeszcze ilość pakietów, które *sniffer* jest w stanie odebrać z terminala mobilnego. Konieczne jest więc upewnienie się czy zapotrzebowanie systemu na pakiety jest zaspokojone. Można początkowo założyć minimalny czas trwania pełnego pomiaru na 1 sekundę – czyli konieczność odebrania 12 pakietów w tym czasie.

Podczas tworzenia systemu przeprowadzono weryfikację natężenia ruchu sieciowego telefonu *smartphone*. Terminal mobilny w stanie spoczynku z włączonym Wi-Fi emituje niewielkie ilości pakietów. Dokładne zależności czasowe i źródła tych pakietów nie zostały sprawdzone, ze względu na losową naturę ruchu sieciowego. Najważniejszą informacją jest jednak fakt, iż ilość odbieranych pakietów była mniejsza niż 12 na sekundę.

Urządzenie mobilne może być zlokalizowane jedynie, gdy pełni rolę strony nadawczej. Gdy telefon aktywnie korzysta z sieci, większość jego akcji powoduje pobieranie, a nie transmitowanie danych (np. pobieranie stron internetowych). Jedyną opcją jest więc założenie, że telefon korzysta ze stron o dużym transferze, licząc na to, że wysyłają dużo informacji zwrotnych. W tym celu sprawdzono jak zachowuje się transfer telefonu podczas korzystania z serwisów wideo (takich jak serwis *YouTube*). W tych warunkach *sniffer* jest w stanie odbierać odpowiednią ilość pakietów. Samo założenie nie będzie jednak zawsze spełnione.

Brak transmisji urządzeń mobilnych jest zrozumiały z powodu wykorzystania zasilania bateryjnego. Urządzenia podłączone do sieci zasilającej nie mają tego ograniczenia. Szczególnym przypadkiem takich urządzeń są same punkty referencyjne. Ze względu na pakiety *Beacon* okresowo transmitowane przez AP, są one łatwe do lokalizacji. Lokalizowanie nieznanych AP jest więc łatwe, o ile na terenie jest określona mapa *fingerprintingu*.

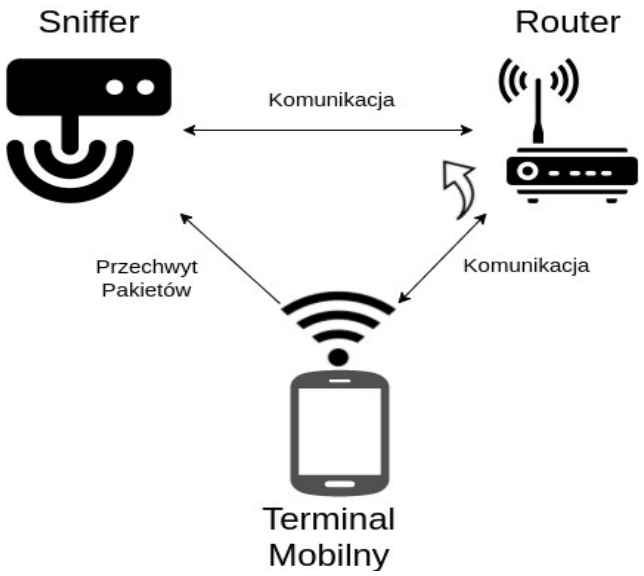
Idealnymi warunkami lokalizacji dla terminali *moblilnych* mimo wszystko byłaby transmisja dużej ilości pakietów w czasie lokalizacji. Oznaczałoby to posiadanie specjalnej aplikacji, która transmitowałaby taką ilość danych. Zdecydowano, że najlepszą opcją będzie zaimplementowanie sztucznego ruchu sieciowego, używanego wyłącznie w celach lokalizacji. Takie rozwiązanie zakłada jednak lokalizację wyłącznie terminali, które korzystają z takiej generującej ruch aplikacji. Jest to koszt, który należy ponieść, by system mógł działać niezawodnie w każdej chwili czasu.

5.2. Sieć i struktura sieciowa

5.2.1.Wymagania dotyczące sieci i interfejsu

Projekt wymaga, aby cały system był zamknięty w jednym urządzeniu wbudowanym. Jest to urządzenie wyniesione, które musi zapewniać dostęp zdalny poprzez sieć internet. System będzie lokalizował różne rodzaje urządzeń. Każde z nich powinno mieć ten sam interfejs do komunikacji z systemem. Problem dotyczy nie tylko interfejsu ale i generacji sztucznego ruchu, koniecznego do zapewnienia pakietów do pomiaru RSSI.

5.2.2.Struktura sieciowa



Rys. 5.2 Komunikacja i przechwyt pakietów w systemie

Karta sieciowa sniffera musi koniecznie być w trybie *monitora* lub *promiscuous*, by móc przechwytywać pakiety. Wiąże się z tym fakt, iż sniffer nie może jednocześnie być AP. Zdalne sterowanie systemem jest więc realizowane drogą okrężną, poprzez sieć, do której połączony jest zarówno bezprzewodowy terminal mobilny jak i przewodowy sniffer. Przechwytywanie pakietów odbywa się drogą bezpośrednią. Sytuacja jest przedstawiona na rys. 5.2.

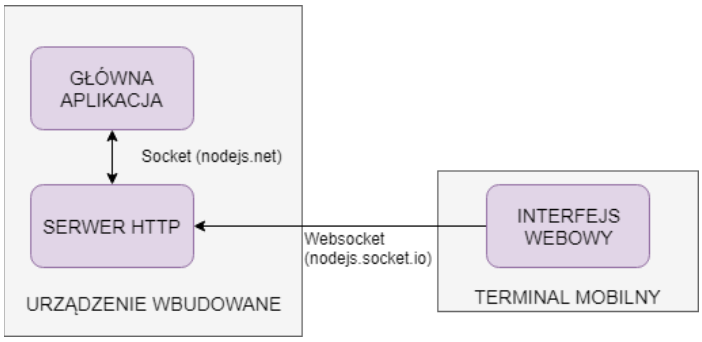
5.2.3.Struktura programowa

Schemat oprogramowania sieciowego przedstawiono na rys. 5.3. Oprogramowanie sieciowe składa się z trzech jednostek:

- głównej aplikacji,
- serwera HTTP,
- strony internetowej, *hostowanej* na serwerze HTTP.

Główna aplikacja podczas inicjalizacji tworzy nową instancję serwera HTTP. Proces serwera istnieje dopóki główna aplikacja jest z nim połączona. Skrypty do komunikacji głównego programu z serwerem HTTP są zawarte w klasie *server_communication*. Aplikacja łączy się automatycznie z serwerem używając gniazda sieciowego.

Terminal mobilny jest w stanie połączyć się z serwerem HTTP w celu pobrania strony internetowej. Połączenie między stroną i serwerem jest utrzymywane przy pomocy *websocketów*. Strona internetowa jest interfejsem pomiarowym całego systemu.



Rys. 5.3. Bloki jednostek sieciowych w systemie

Wszystkie komunikaty przesyłane między główną aplikacją a stroną internetową są w formacie JSON (ang. JavaScript Object Notation). Format ten został wykorzystany do stworzenia struktury komend. Każda komenda ma swój numer identyfikacyjny, znany przez obie strony, oraz argumenty. Argumenty mogą być zarówno tekstowe jak i liczbowe. To jakie argumenty są konieczne zależy od samej komendy. Numer komendy jest przechowywany w polu *command*, a argumenty w polu *payload*:

```
{
        "command":0,
        "payload":[1,2,3,4,5]
}
```

Celem klasy *server_communication* jest stworzenie instancji serwera HTTP i ustanowienie połączenia z nim. Połączenie to może być zrealizowane za pomocą gniazd

sieciowych (ang. *sockets*). Gniazda w systemie *Linux* są reprezentowane przez pliki. Gniazdo TCP tworzy się przy pomocy polecenia *socket*, które zwraca numer deskryptora tego pliku:

```
sock = socket(AF_INET, SOCK_STREAM, 0);
```

W celu uzyskania nieblokującego trybu funkcji *socketu*, konieczne jest operowanie na niskopoziomowym API deskryptorów plików. Funkcja *fcntl* jest używana do manipulowania deskryptorami. Ustawienie nieblokującej pracy deskryptora osiąga się ustawiając konkretną flagę za pomocą komendy *F_SETFL*:

```
fcntl(sock, F_SETFL, O_NONBLOCK);
```

Połączenie może być ustanowione przy użyciu funkcji *connect*. Funkcja jest nieblokująca, więc będzie okresowo zwracać błąd *EINPROGRESS,* co jest prawidłowym działaniem.

W celu monitorowania deskryptorów plików, używa się funkcji *select*. Funkcja przyjmuje jako argumenty trzy zbiory deskryptorów: zapisu, odczytu i wyjątków. Deskryptory mogą być wpisane do zbioru za pomocą makra *FD_SET*. Funkcja *select* sprawdza dostarczone deskryptory i zostawia w zbiorach tylko te, które są gotowe. Gotowość oznacza tu możliwość wykonania konkretnej akcji, np. dla deskryptora w zbiorze odczytu oznacza to gotowość danych do odczytu. Sprawdzenie które deskryptory pozostały w zbiorach dokonuje się za pomocą makra *FD_ISSET*.

Funkcja *select* zwraca ilość pozostawionych deskryptorów w zbiorach. Jeśli dowolny deskryptor *socketu* stwierdził gotowość, możliwe jest, że nastąpiło połączenie. Przy pomocy funkcji *getsockopt* sprawdza się czy wystąpił błąd. Jeżeli nie, połączenie zostało ustanowione poprawnie. Nieblokujące połączenie osiąga się wykonując następujący kod:

```
while(true){
   int bytes=connect(sock, (struct sockaddr *)&server, sizeof(server));
   sleep(1);
   if (bytes < 0){
      if(errno==EINPROGRESS){
         if(select(sock+1,&read_fds,&write_fds,NULL,&timeout)!=0){
               getsockopt(sock,SOL_SOCKET,SO_ERROR,&err,&len);
               if(err!=0){
                  printf("connect_to_server() ERROR:%s\n",strerror(err));
               }
               else{
```

```
                puts("Connected");
                return 0;
                break;
            }
        }
    }
    perror("connect failed. Error");
  }
}
```

Wysyłanie wiadomości za pomocą *socketu* może odbywać się w dowolnym miejscu kodu. Problem pojawia się z oczekiwaniem na wiadomości z zewnątrz. Aby sprawdzić czy *socket* jest gotowy do odczytu, można dodać deskryptor *socketu* do zbioru odczytu.

Gdy zbiór odczytu zostaje zmieniony, a dany deskryptor nadal znajduje się w zbiorze, oznacza to, że dane czekają na odczyt. Gotowe do odczytu dane pobiera się funkcją *recv*. W wypadku gdy dane będą gotowe, a *recv* nie zwróci żadnego bajtu, oznacza to, iż doszło do rozłączenia z serwerem.

Metoda zajmująca się odczytem, otrzymuje w argumencie funkcję *callback*, zajmującą się interpretacją danych odebranych. Po parsowaniu tekstu JSON do struktury, dane zostają przekazane do funkcji, gdzie następuje interpretacja komendy. Po interpretacji, do strony internetowej mogą zostać wysłane informacje zwrotne odnośnie sukcesu bądź porażki zadanego polecenia.

Serwer HTTP *hostuje* stronę internetową interfejsu. Stanowi on wyłącznie węzeł tranzytowy między główną aplikacją a przeglądarką terminala ruchomego. Nie uczestniczy on w komunikacji poza przesyłaniem danych między stronami. Serwer napisany jest przy użyciu następujących technologii:

- *JavaScript* - język programowania dla *backend* i *frontend*,
- *Node.js* - środowisko zapewniające moduły dla JS, głównie *backend*,
- *JQuery* - biblioteka ułatwiająca dostęp do pól elementów *frontendu*,
- *HTML*,*CSS* - definiują strukturę strony.

Node.js dysponuje własnym managerem modułów - *npm*. Wszystkie potrzebne moduły mogą zostać pobrane za jego pomocą z repozytorium *online*. Każdy moduł musi znajdować się w lokalnym lub globalnym katalogu [22]. Projekt wymaga pięciu modułów:

- *Node.js* - środowisko zapewniające moduły,
- *express* - *framework* do tworzenia stron [23],
- HTTP - obsługa HTTP,

- *socket.io* - biblioteka umożliwiająca dwustronną komunikację za pomocą *websocketów* [24],
- *net* - API do tworzenia serwerów TCP [25].

Dostępne moduły mogą być użyte w programie poprzez załączenie ich poleceniem *require*:

```
var http = require('http').Server(app);
```

Programowanie w *node.js* opiera się o zdarzenia. Przypisanie callbacku do zdarzenia osiąga się za pomocą metody *on*.

Serwer HTTP inicjalizowany jest przy pomocy *frameworku express* i obiektu *http.server*. Inicjalizacja serwera polega na wywołaniu funkcji *app.get,* wysyłającej stronę HTML do użytkownika oraz *http.listen* odpowiedzialną za nasłuchiwanie na połączenia na wybranym porcie. Łącząc się z urządzeniem na odpowiednim porcie, uzyska się stronę internetową z interfejsem aplikacji.

Za pomocą funkcji *createServer* obiektu *net* tworzony jest serwer TCP. W ciele funkcji ustawia się funkcje *callback* dla każdego używanego zdarzenia, zarówno wbudowanego jak i użytkownika:

```
io.on('connection',connectWebClient.bind(this,c));
c.on('end', socketEnd.bind(this,c));
c.on('data',socketData);
```

Zdarzenia *connection* i *end* oznaczają ustanowienie i zerwanie połączenia, a *data* oznacza przyjście danych. *Callbacki* definiują akcje, które należy podjąć w wypadku wystąpienia zdarzenia. Gdy serwer straci połączenie z aplikacją, program wyłączy się. Gdy nastąpi połączenie, wykona się dalsza część inicjalizacji. Nowo podłączony terminal może wywołać zdarzenia związane z rozłączeniem, przyjściem wiadomości, czy sztucznym ruchem (zwany dalej potocznie *spamem*). Te zdarzenia mają przypisane odpowiednie funkcje *callback:*

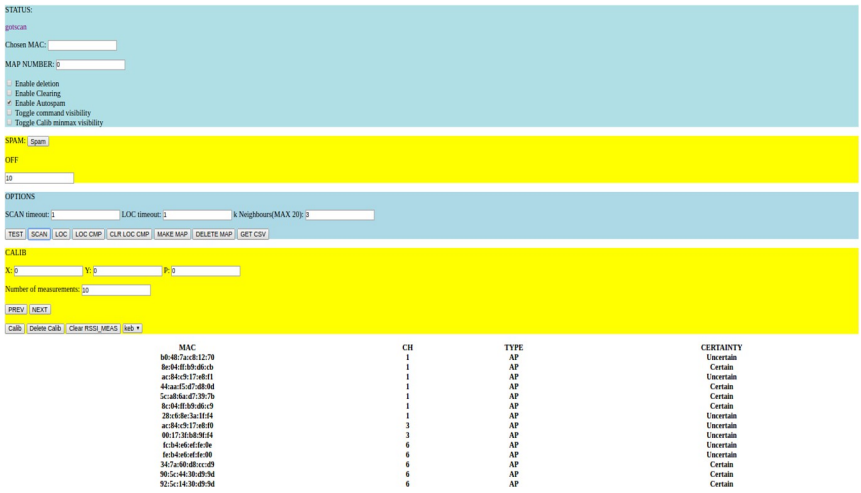
```
socket.on('disconnect',websocketDisconnect.bind(this,c,socket));
socket.on('message', websocketMesage.bind(this,c) );
socket.on('spam', socketSpam );
```

Wartym zaznaczenia jest fakt, iż funkcją *callback* zdarzenia *spam* jest pusta funkcja. Oznacza to, że serwer jedynie przyjmuje wiadomości będące *spamem*, lecz nie interpretuje ich.

Zdarzenie *'data'* odpowiada za komunikację od PC do przeglądarki, a zdarzenie *message* za przeciwny kierunek komunikacji. Za wyjątkiem *spamu*, zdarzenia te przekazują wyłącznie wiadomości między węzłami.

Strona internetowa stanowi interfejs pomiarowy systemu, przedstawiony na rys. 5.4. Za jej pomocą można wykonać kalibrację, lokalizować urządzenia, wykonywać skan oraz generować sztuczny ruch. Umożliwia też wyświetlanie wyników skanu i lokalizacji. Zapewnia informację o statusie wykonywanych operacji i ewentualnych błędach. Sztuczny ruch może być automatycznie włączony w każdym potrzebnym momencie, zależnie od zaznaczonych opcji.

Strona jest przeznaczona głównie do użytku na telefonie typu *smartphone*, więc dodane zostały zabezpieczenia przed przypadkowym kliknięciem niektórych funkcji takich jak usuwanie obiektów z bazy danych. Zabezpieczeniem jest seria *checkboxów* włączających konkretne funkcje.



Rys. 5.4. Interfejs webowy

Kontrolki interfejsu służą komunikacji z serwerem w postaci komend. Z odpowiednich pól GUI (ang. *Graphical User Interface*) pobierane są argumenty komend, po czym tworzona jest tekstowa reprezentacja komendy w formacie JSON. W tej formie wiadomość trafia do głównej aplikacji urządzenia wbudowanego i tam jest interpretowana.

Strona powstała w oparciu o technologie *HTML*, *CSS*, *JQuery* i *Javascript* z *Node.js*. *Node.js* może być używany również po stronie *frontendu*. W tym wypadku konieczne jest by *socket.io* było wykorzystane też na stronie internetowej. By korzystać z *socket.io* należy zaimportować bibliotekę na stronie:

```
<script src="/socket.io/socket.io.js"></script>
```

Następnie należy pozyskać uchwyt do socketa za pomocą polecenia „*var socket = io”*. W ten sposób za pomocą zmiennej *socket* można mieć dostęp do funkcji *socket.io*. Stworzony obiekt domyślnie łączy się z *socketem* na serwerze który hostuje stronę.

Analogicznie do odbioru poprzez *socket.on*, można transmitować komunikaty za pomocą *socket.emit*. *Callback* w *socket.on* zostanie wywołany wyłącznie gdy przyjdzie komunikat związany z tą samą nazwą zdarzenia. Za pomocą tego API zaimplementowano komunikację między węzłami i sztuczny ruch.

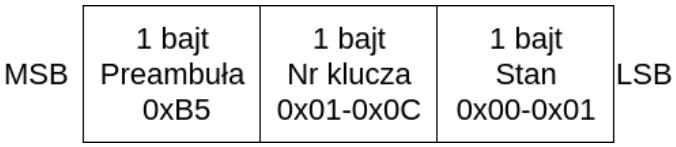
Sztuczny ruch jest obsługiwany w funkcji *spam_execute*. Używając *window.setinterval* można przygotować pętlę wysyłającą wiadomość o zdarzeniu *‘spam’*, w stałych odstępach czasu. Zależnie od stanu zmiennej *spam_on*, pętla jest inicjowana przez *setInterval* lub anulowana poprzez *clearInterval*. Komunikacja działa analogicznie, używając zdarzenia *“message”*. Funkcja wygląda następująco:

```
function spam_execute(){
        console.log("spam:"+spam_on);
        var inter=window.setInterval(function () {
                if(spam_on==false)clearInterval(inter);
                socket.emit('spam', "");
        }, $('#spam_period').val());
}
```

Strona posiada własny interpreter odpowiedzi na komendy w formacie JSON. Specjalnym przypadkiem są komendy o numerach 0 (ACK – ang. acknowledged) i 1 (NACK – ang. Not Acknowledged). Jeśli podczas komendy używającej automatycznego *spamu* strona otrzyma informację NACK, sztuczny ruch musi zostać wyłączony.

5.3. Oprogramowanie sterujące anteną ESPAR

Część cyfrowa anteny ESPAR zajmuje się starowaniem kluczy mikrofalowych do ustawiania charakterystyk promieniowania anteny. Połączenie komputera z mikrokontrolerem ARM zostało zrealizowane poprzez interfejs UART. Sterowanie odbywa się za pomocą komend przesyłanych do kontrolera ARM.



Rys. 5.5. Schemat komend wysyłanych do anteny ESPAR

Rys 5.5 przedstawia komendę wysyłaną do anteny. Komenda zawiera 3 bajty: preambułę, numer klucza i stan klucza (gdzie 1 -reflektor, 0 - direktor). Komendy są przesyłane do maszyny stanów bajt po bajcie. Po otrzymaniu komendy zmiany konkretnego klucza, osobna funkcja interpretera zajmuje się zmianą stanu klucza podłączonego do konkretnego wyjścia mikrokontrolera.

Główna aplikacja lokalizatora łączy się z anteną przy pomocy interfejsu szeregowego. Do jego obsługi użyto biblioteki *termios*. Obsługa opiera się o konfigurację, otwarcie urządzenia i odczyt z niego. Konfiguracja polega na ustawieniu flag w 4 kategoriach: wejścia, wyjścia, sterujących, lokalnych. Dla projektu wymagane są tylko 2 flagi. Używane opcje są następujące:

```
tio.c_iflag=0;
tio.c_oflag=0;
tio.c_cflag=CS8|CREAD|CLOCAL;
tio.c_lflag=0;
```

gdzie:

CS8 – ustawienie rozmiaru znaku na 8 bitów,

CREAD – włączenie odbiornika,

CLOCAL – ignorowanie linii kontrolnych modemu [26].

Otwarcie urządzenia odbywa się przy pomocy funkcji *open*. Dodatkowe flagi zapewniają otwarcie urządzenia na odczyt i zapis, oraz nieblokujące działanie tej funkcji:

```
tty_fd=open(ttyName, O_RDWR | O_NONBLOCK);
```

Konfiguracja odczytu opiera się o zmienne *VTIME* i *VMIN* . Definiują one zachowanie funkcji *read*, a mianowicie moment jej zakończenia. *VTIME* oznacza interwał między odczytami bajtów z bufora wejściowego, a *VMIN* oznacza ilość bajtów do zwrócenia. Obie wartości mają specjalne modyfikacje gdy któraś z nich jest równa 0 [27]. Wybrano kombinację, w której funkcja *read* czeka do momentu przyjścia jednego bajtu:

```
tio.c_cc[VMIN]=1;
tio.c_cc[VTIME]=0;
```

Ostatnim krokiem konfiguracji portu szeregowego jest ustawienie jego prędkości za pomocą funkcji *cfsetospeed* oraz *cfsetispeed.* Wybrano prędkość *115200.*

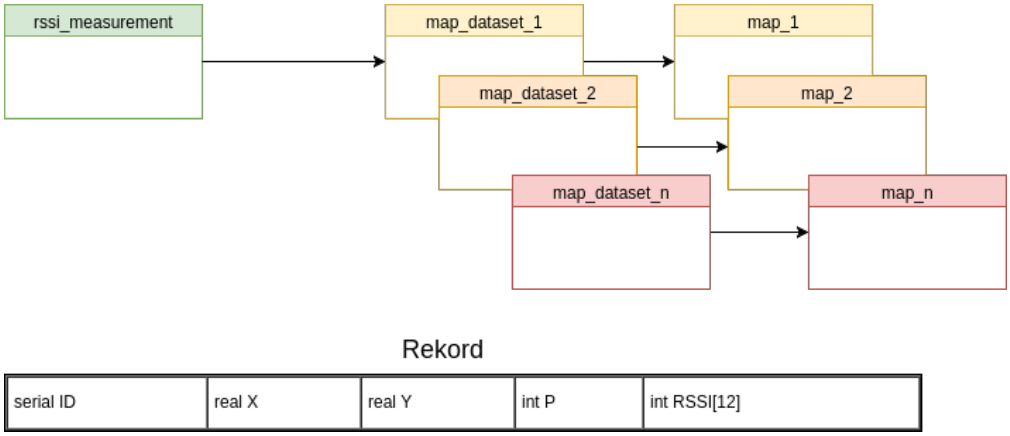
Konfigurację aplikuje się do otworzonego deskryptora za pomocą komendy „*tcsetattr(tty_fd,TCSANOW,&tio)”*. Taka konfiguracja pozwala na korzystanie z funkcji *read* i *write* w celu dostępu do portu szeregowego.

5.4. Baza danych

System potrzebuje bazy danych do przechowywania dużych ilości pomiarów RSSI związanych z fazą offline algorytmu k-NN. Użyty system zarządzania bazami danych to *postgreSQL*. Obsługa bazy jest napisana za pomocą biblioteki *libpq*. W każdej metodzie lokalizacji są używane bezpośrednie funkcje biblioteczne *libpq*. Praca aplikacji zaczyna się od połączenia z bazą za pomocą funkcji *PQconnectdb*, zwracającej uchwyt *Pgconn*:

dbase=PQconnectdb("user=<username> password=<password>
dbname=<dbname>");

Standardowym schematem przepływu danych między bazą a aplikacją jest pobranie danych, przetworzenie ich i wysłanie danych. Podczas transmisji danych występuje problem z kolejnością bajtów (ang. *endianness*). Kolejność bajtów dla hosta i sieci to kolejno *host byte order* i *network byte order*. Dokładny *endianness* dla konkretnego przypadku nigdy nie jest odgórnie ustalony. Jeżeli kolejności bajtów dla hosta i sieci się różnią, istnieje potrzeba odwrócenia bajtów przy każdej operacji transmisji danych z bazą. Istnieją do tego funkcje systemowe (np. *htons*), jednakże stworzono własną funkcję *reverse_bytes*, dla ułatwienia przenoszenia danych między zmiennymi.



Rys. 5.6. Struktura Bazy danych i przechowywane w niej rekordy

Po odwróceniu danych, komunikacja z bazą jest obsługiwana za pomocą funkcji *PQexecParams* oraz *PQexec*. Funkcje te zwracają bufor wyniku o typie *PGresult*. Każde zapytanie musi zakończyć się czyszczeniem bufora wyniku, co zapewnia funkcja *PQclear*. Dodatkowo możliwe jest sprawdzenie błędów za pomocą funkcji *PqresultStatus*. Przykładowa komunikacja z bazą wygląda następująco:

```
  sprintf(buf,"DELETE FROM %s WHERE x=%d and y=%d and pos=%d"
,tablename,x,y,p);
  result=PQexecParams(dbase,buf,0,NULL,NULL,NULL,NULL,1);
  if (PQresultStatus(result) != PGRES_COMMAND_OK) {
     PQclear(result);
     return -1;
  }
Pqclear(result);
```

Struktura bazy jest przedstawiona na rys. 5.6. Wszystkie tabele przechowujące RSSI mają tą samą strukturę krotek. Rekord zawiera informację o pozycji i orientacji urządzenia pomiarowego, oraz same pomiary RSSI.

Tabela *rssi_measurement* jest buforem, w którym przechowywane są aktualnie prowadzone pomiary. Po wykonaniu pomiarów wyniki można przenieść do *datasetów* o numerze *n*. *Dataset* o numerze *n* służy jako zestaw danych potrzebny do stworzenia mapy *fingerprintingu* numer *n*. System pozwala na stworzenie i korzystanie z dowolnej ilości map.

5.5. Lokalizacja

Funkcjonalność związana z lokalizacją jest zawarta w klasie *Localization*. Metody klasy wykorzystują dane składowane w bazie danych. Klasa zajmuje się więc również obsługą dostępu do bazy i przetwarzaniem wszelkich danych wchodzących bądź wychodzących z bazy. Obiekt klasy *Localization* tworzy przy konstrukcji obiekt klasy *RSSI_Sniffer* w celu przeprowadzania pomiarów.

Lokalizacja składa się z kilku kroków. Przed jakimikolwiek operacjami należy wykonać skan w poszukiwaniu urządzeń Wi-Fi. Skan polega na znalezieniu aktywnych urządzeń Wi-Fi. Operacja ta zapewnia niezbędne informacje o numerach kanałów operacyjnych urządzeń oraz rodzaju samych urządzeń.

Następnym krokiem są fazy *offline* i online algorytmu k-NN. Faza *offline* jest inicjalizacją systemu, w której tworzona jest mapa *fingerprintingu*. W drugiej fazie możliwy jest docelowy proces lokalizacji urządzeń na podstawie stworzonej mapy.

5.5.1. Skan

Zasada działania standardu IEEE 802.11 ma kluczowe znaczenie w projektowaniu systemu do lokalizacji terminali mobilnych. Wedle Europejskich norm, *access pointy* Wi-Fi mają do dyspozycji 13 kanałów [28]. Dany AP prowadzi komunikację na jednym kanale z wszystkimi swoimi *hostami*. Jeśli kanał nie został wybrany manualnie, AP dobiera go automatycznie. Jest to kluczowe gdy w okolicy pracuje wiele AP. Należy mieć na uwadze to, że AP może zmienić swój wybrany kanał [29].

Fakt istnienia wielu kanałów utrudnia znacznie monitorowanie ruchu w sieci Wi-Fi. Typowy konsolowy sniffer Wi-Fi ma możliwość przechwytu pakietów na jednym tylko kanale w danej chwili. Na dodatek, *sniffer* nie ma możliwości zmiany kanału [30], a czasem nawet sprawdzenia tegoż kanału. Konsolowy program nie ma również możliwości zmiany konfiguracji podczas jego pracy.

Sniffer by zlokalizować dany terminal musi pracować na odpowiednim kanale. Każdy terminal, który korzysta z sieci Wi-Fi ma przydzielony kanał, taki sam jak AP z którym jest połączony. Istnieje więc potrzeba weryfikacji kanałów, na których operują AP, co daje też informację na temat kanałów powiązanych z nimi hostów. Znalezienie tej asocjacji i zapamiętanie informacji o niej dla każdego terminala jest niezbędne. Fakt zmiany kanału wymusza również powtarzanie wspomnianej weryfikacji, w razie utraty sygnału.

Do weryfikacji kanału, na którym nadają hosty, potrzebny jest *sniffer* dostarczający taką informację. *Sniffery* konsolowe spełniają to wymaganie, lecz nie pozwalają na zmianę kanału w samym programie. To musi być dokonywane ręcznie za pomocą polecenia systemowego w *bashu*. Najbardziej opłacalne jest więc opracowanie tej funkcjonalności na podstawie stworzonego sniffera.

Skan jest jedną z funkcji potrzebnych w klasie *Localization*, realizowaną przez funkcję *full_channel_scan_output_matrix*. Skan ma zbierać informacje o okolicznych urządzeniach nadających za pomocą protokołu IEEE 802.11. Podczas skanu zbierane są wszystkie pakiety przychodzące do *sniffera*.

Skan rozpoczyna się ustawieniem anteny ESPAR w stan dookólny, umożliwiając odbiór pakietów ze wszystkich kierunków. Kolejnym krokiem jest ustawienie kanału w systemie Linux. W języku C do wywołania skryptów *bashowych* używamy funkcji *system.* Zmiana kanału Wi-Fi wymaga uprawnień *roota*. Hasło do polecenia *sudo* można przesłać poprzez potok.

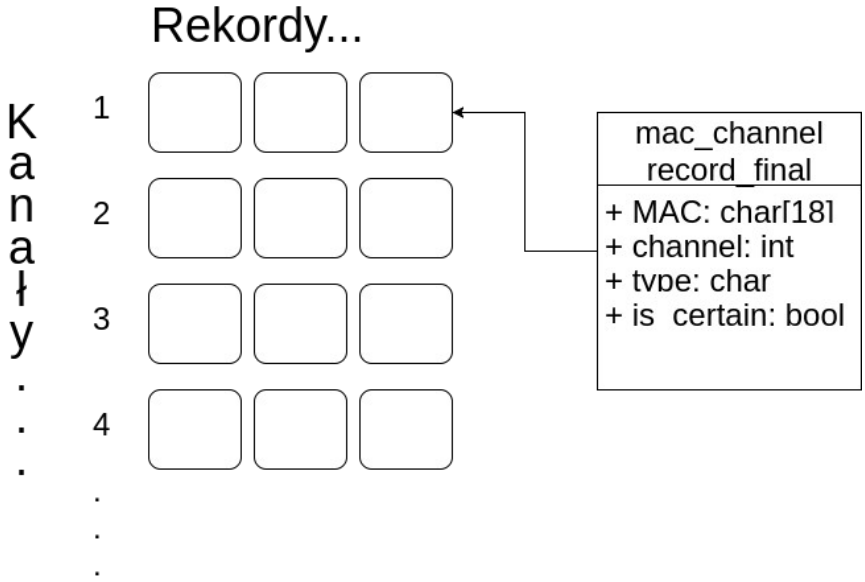
Funkcja następnie iteracyjnie przechwytuje pakiety z każdego kolejnego kanału. Pętla dla każdej iteracji zaczyna się włączeniem i kończy wyłączeniem przechwytu pakietów. Przechwyt pakietów dla każdego kanału trwa wyznaczoną przez użytkownika ilość sekund. Urządzenia są identyfikowane poprzez adres MAC. Wykrywa się je poprzez informacje o adresie nadawcy (TA) zawartych w pakiecie. O dostępności tej informacji decyduje typ i podtyp pakietu. Pakiety bez informacji o nadawcy są ignorowane.

Każde nowe urządzenie jest dodawane do wektora struktur *mac_channels_record*. Wektor przechowuje wszystkie wykryte urządzenia i informacje o nich. Pojedyncza struktura wygląda następująco:

```
struct mac_channels_record {
  char mac[6];
  int channels[13];
  char type;
```

};

Dla każdego pakietu, którego nadawca jest już w tablicy, dodaje się jeden punkt do tabeli *channels*, na indeksie odpowiadającym kanałowi (tablice indeksowane są od 0 a kanały od 1). Dodatkowo, jeżeli jakikolwiek z pakietów z danego adresu mac był pakietem typu *Beacon*, oznacza to, że nadawca jest AP (*type*='a'), gdyż tylko AP może wysyłać tego typu pakiet. W przeciwnym wypadku jest to Host (*type*='h'). Gdyby czas skanowania na każdym kanale był dużo mniejszy od sekundy (np. kilka milisekund) możliwa byłaby sytuacja nie przechwycenia pakietu *Beacon* i błędne rozpoznanie danego adresu mac jako *hosta*. Przyjmuje się, że taki skan powinien trwać kilka sekund, by nie pominąć rzadko nadających urządzeń.



Rys. 5.7. Macierz wynikowa skanu zawierająca obiekty typu mac_channel_record_final. Wiersze reprezentują konkretne kanały, a kolumny zarejestrowane urządzenia na tym kanale.

Głównym celem skanowania sieci jest przypisanie kanału do każdego znalezionego adresu MAC. Po wykonaniu pomiarów tablica *channels* zawiera ilość pakietów, które wysłało urządzenie o danym adresie MAC, na każdym z kanałów. Kanał na którym nadano najwięcej pakietów staje się kanałem wybranym dla danego urządzenia. Oprócz wyboru kanału program dokonuje szacowania pewności tego wyboru. Sprawdzany jest stosunek pakietów na wybranym kanale do sumy pakietów w innych kanałach. Przyjęto, że jeśli ten stosunek jest większy niż 2 i jeśli pobrano więcej niż 1 pakiet na tym kanale, to wybór jest pewny. Jest to szacunkowe kryterium, gdyż możliwość błędnego wyboru zawsze istnieje. W najlepszym wypadku błędna dedukcja po prostu zmniejszy ilość wysyłanych pakietów.

MAC	CH	TYPE	CERTAINTY
34:7a:60:d8:cc:d9	1	AP	Certain
44:aa:f5:d7:d8:0d	1	AP	Certain
ac:84:c9:17:e8:f0	1	AP	Uncertain
ac:84:c9:17:e8:f1	1	AP	Uncertain
b0:48:7a:c8:12:70	1	AP	Uncertain
8c:04:ff:b9:d6:c9	1	AP	Certain
8e:04:ff:b9:d6:cb	1	AP	Certain
dc:ee:06:eb:23:22	2	AP	Certain
92:5c:14:30:d9:9d	6	AP	Uncertain
90:5c:44:30:d9:9d	6	AP	Certain
d8:5d:4c:b4:c6:cc	6	AP	Certain
00:71:c2:35:0e:e6	6	AP	Certain
06:7c:34:35:1d:7f	6	AP	Uncertain
64:7c:34:35:1d:7f	6	AP	Uncertain
c0:7c:d1:f6:8b:a3	7	AP	Certain
00:71:c2:35:7e:22	9	AP	Certain
c0:4a:00:e4:6b:b5	11	AP	Uncertain
2a:be:9b:33:75:01	11	AP	Certain
28:be:9b:33:75:0f	11	AP	Uncertain
54:67:51:e5:32:f1	11	AP	Certain

Rys. 5.8. Widok wyniku skanu w interfejsie strony internetowej

Ostatnim krokiem skanu jest stworzenie finalnych rekordów z wybranym kanałem i określoną pewnością tego wyboru. Rekordy są umieszczane w dwuwymiarowym wektorze, gdzie jeden z wymiarów oznacza kanał, na którym dane urządzenie operuje. Wektor ten jest ostatecznym wynikiem skanu, przedstawionym na rys. 5.7. Graficzną reprezentację tego skanu można odczytać z interfejsu systemu, jak widać to na rys. 5.8.

5.5.2. Pomiar RSSI

Pomiar RSSI to podstawowa operacja systemu. Procedura pomiaru zestawu *K* próbek RSSI od jednego urządzenia jest używana w wielu funkcjach systemu. Poprzez zestaw rozumie się tu zmierzenie RSSI sygnału z danego urządzenia dla każdej z *K* konfiguracji anteny ESPAR. Wybrane urządzenie musi znajdować się w tablicy wynikowej skanu, by wiadome było na którym kanale należy szukać urządzenia. Poprawny pakiet musi zawierać informacje o czasie przybycia, adresie MAC nadawcy i RSSI.

Kluczowym dla pomiaru RSSI jest opracowanie sekwencji pomiaru i przełączania wiązki anteny. Standardowym rozwiązaniem jest czasowe przemiatanie wiązki anteny i odbieranie wszystkich pakietów pomiędzy przełączeniami. Takie rozwiązanie umożliwia lokalizację wielu urządzeń jednocześnie. Problemem tej metody jest możliwość nie odebrania pakietu od danego urządzenia na jednej z konfiguracji. Na tą chwilę nie jest wiadome jaki wpływ na algorytm k-NN mają niepełne wyniki. Problematyczny dla lokalizacji wielu terminali jest fakt istnienia wielu kanałów Wi-Fi. Przemiatanie anteny musiałoby odbywać się tylko na jednym kanale co ograniczyłoby lokalizację wielu urządzeń.

Inną metodą jest sekwencyjne zmienianie konfiguracji anteny po pobraniu co najmniej jednego pakietu. Ta opcja ogranicza użyteczność do jednego terminala, ale umożliwia

automatyczne dostosowanie się do ilości pakietów jakie emituje. Jest to metoda bezpieczniejsza, gdyż zawsze otrzymuje się pełny pomiar 12-stu RSSI. Ze względu na wiele problemów związanych z lokalizacją wielu terminali jednocześnie, wybrano drugą opcję i skupiono się na lokalizacji jednego terminala.

Funkcjonalność pomiaru RSSI jest realizowana przez funkcję *loc_mac_matrix*. Pierwszym krokiem funkcji jest włączenie monitoringu pakietów. Następnie dla każdego stanu anteny pobiera się pojedynczy pomiar RSSI. Konfiguracja anteny zostaje zmieniona i zostaje pobrany czas systemowy, w którym ta zmiana nastąpiła. Pomiar polega na oczekiwaniu na poprawny pakiet od danego urządzenia i odczytanie z niego informacji o RSSI. By pomiar był wiarygodny, pakiet musi zostać pobrany koniecznie po ustawieniu wiązki anteny w odpowiednim kierunku. Każdy pobrany ze *sniffera* pakiet musi więc mieć sprawdzony czas przyjścia. Wszystkie pakiety pobrane przed ustawieniem anteny są uznawane jako niepoprawne i są ignorowane.

Przyjście aktualnego pakietu kończy pomiar dla danego stanu anteny. Pakiet musi dotrzeć do anteny w sprecyzowanym przez użytkownika interwale czasowym. Jeśli antena nie odbierze pakietu na jednej z konfiguracji, pomiar zakończy się niepowodzeniem. Po pobraniu RSSI z pakietów dla każdej konfiguracji anteny przechwyt zostaje wyłączony i pomiar się kończy.

5.5.3 Kalibracja

Kalibracja systemu implementuje fazę *offline* algorytmu k-NN. *Fingerprinting* wymaga stworzenia mapy wartości referencyjnych w obszarze działania systemu. W obszarze tym tworzy się siatkę punktów pomiarowych. Dla każdego punktu należy wykonać pomiar kalibracyjny. W celu wykonania jakichkolwiek pomiarów RSSI, próbnik powinien generować sztuczny ruch, aby najszybciej uzyskać potrzebne dane.

Pomiar kalibracyjny to pomiar RSSI, które to jest bardzo podatne na fluktuację. Rodzi to problem, gdyż mapa *fingerprintingu* nie może być obarczona losowym błędem. by temu zapobiec należy wykonać uśrednianie w celu usunięcia błędu. W tym celu użyto operacji mediany. Medianę charakteryzuje tłumienie bardzo dużych odchyłek wartości uśrednianej, a takie odchyłki są spodziewane w przypadku RSSI.

Pomiar polega więc na zebraniu wielu zestawów RSSI dla każdego punktu pomiarowego. Pomiary te umieszczane są w bazie danych, w tabeli *rssi_measurement*. Ilość danych dla algorytmu lokalizacji może być zwiększona poprzez dodanie zmiennej w postaci orientacji terminala [10]. *N* przestrzennych orientacji w każdym punkcie siatki zwiększy teoretyczną ilość punktów referencyjnych N-krotnie. Sposób wykorzystania dodatkowych punków i ich efekt na wynik lokalizacji zostanie zweryfikowany podczas testów systemu.

Gdy pomiary zostaną zebrane należy zachować je w innej tablicy w celu poglądowym. Dane zostają zachowane w tablicy *map_dataset_n*, gdzie n jest numerem mapy, która zostanie

z nich stworzona. Na stworzenie mapy składa się kilka operacji. Wszystkie pomiary RSSI wykonane dla tego samego punktu i orientacji muszą być poddane operacji mediany. Każdy punkt ma mieć jeden zestaw K wartości RSSI, gdzie K to ilość użytych konfiguracji anteny. Każdy taki zestaw normalizuje się następnie do pierwszej wartości RSSI, czyli do pomiaru dla pierwszej konfiguracji anteny [10]. Tak spreparowane rekordy zostają skopiowane do bazy. Tworzona jest nowa tabela map_n, w której rekordy te tworzą mapę *fingerprintingu*. System posiadający przynajmniej jedną mapę jest gotowy do lokalizacji w fazie online.

5.5.4 Lokalizacja

Funkcjonalność lokalizacji jest implementacją fazy online algorytmu k-NN. W celu zlokalizowania urządzenie lokalizowane musi aktywnie generować ruch w sieci. Wykonywany jest jeden pomiar RSSI urządzenia, którego wyniki przekazuje się do funkcji lokalizującej. Tam pomiar jest uśredniany do pierwszego RSSI, podobnie jak przy tworzeniu mapy.

Lokalizacja przebiega iteracyjnie dla każdego rekordu (punktu na siatce) na mapie *fingerprintingu*. Oprócz współrzędnych przestrzennych w lokalizacji używa się również parametru p. Parametr p oznacza orientację urządzenia w przestrzeni. Nowy zestaw RSSI jest porównywany z zestawem z bazy według wzoru 2.2. Dla obecnego przypadku lokalizacji w przestrzeni 2D z anteną ESPAR równanie przybiera postać wzoru 5.1.

$$D[x,y,p]=\sqrt{\sum_{i=1}^{K}\left(S_T[i]-S[x,y,p,i]\right)^2} \tag{5.1}$$

gdzie:

S_T – RSSI pomierzone w fazie *online,*

S – RSSI pomierzone w fazie *offline*, pobrane z bazy danych,

K – ilość konfiguracji anteny, tu równe 12,

i – indeks konfiguracji anteny,

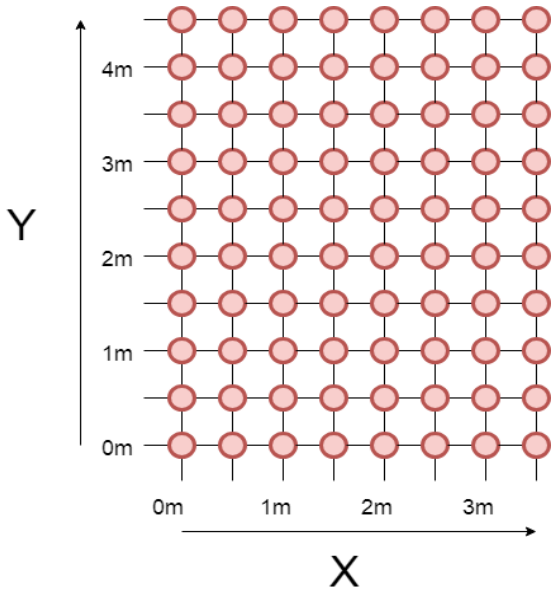
x,y – współrzędne punktu pomiarowego,

p – współrzędna orientacji.

Suma średniokwadratowych różnic RSSI z fazy *online* i *offline* dla każdej konfiguracji anteny daje wartość *D* dla danego punktu. Algorytm iterując po wszystkich zmiennych szuka punktów o najmniejszej wartości *D*. Znalezienie k takich punktów, implementuje się za pomocą wektora o długości k, gdzie na jego początku znajduje się najmniejsza wartość. Kolejne wartości reprezentują kolejne najmniejsze wartości *D*. Do wektora warunkowo wpisuje się kolejne obliczone wartości *D*. Wpisane zostają tylko te wartości, które są mniejsze niż którakolwiek wartość w wektorze. Nowo wpisana wartość przesuwa wartości większe od niej, w dół wektora. Finalnie, wektor zawiera *k* najbliższych sąsiadów estymowanego miejsca, w którym znajduje się lokalizowane urządzenie. Ostateczny wynik lokalizacji jest punktem o współrzędnych będących uśrednieniem każdej ze współrzędnych sąsiadów.

6. TESTY SYSTEMU

Celem pomiarów było sprawdzenie poprawności i dokładności algorytmu i systemu w rzeczywistym środowisku. Pomiary odbyły się w laboratorium komputerowym. Do badań użyto fragmentu pomieszczenia o wymiarach 4.5 m na 3.5 m. Wymiary te oznaczono jako Y i X. Na wyznaczonym obszarze zdefiniowano punkty co 0.5 metra. Schemat obszaru przedstawia rys 5.1.



Rys. 6.1. Siatka w środowisku pomiarowym na wykresie XY.

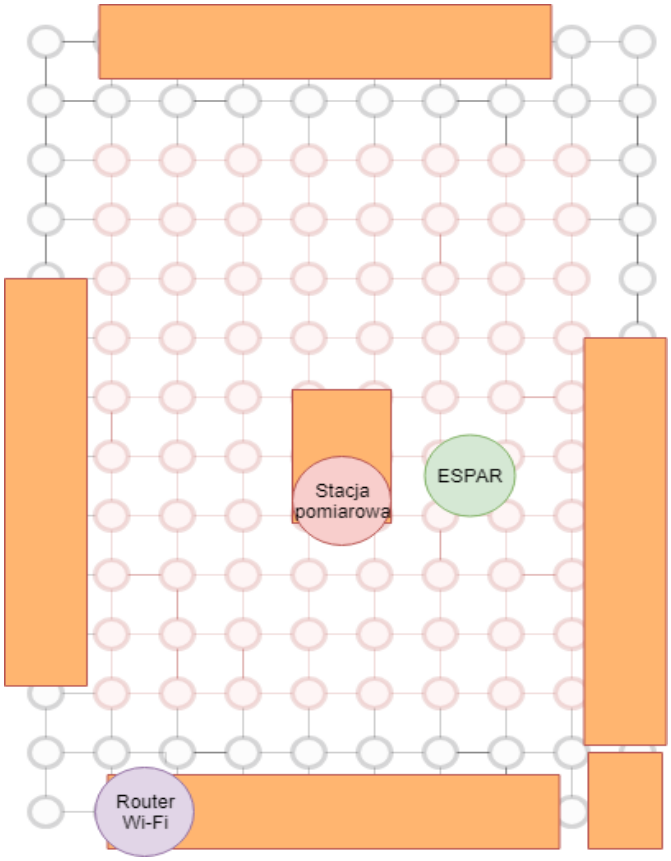
Schematyczne rozmieszczenie umeblowania pomieszczenia przedstawia rys. 6.2. W pomieszczeniu, wyznaczony obszar badany był otoczony stanowiskami komputerowymi. Dodatkowe stanowisko znajdowało się pośrodku pomieszczenia. Na całej długości lewej ściany znajdował się rząd okien.

6.1 Przygotowanie środowiska pomiarowego

Rozmieszczenie sprzętu w środowisku pomiarowym przedstawia poglądowo rys 6.2. W celu rozpoczęcia badań należało podłączyć system w docelowym środowisku. Pomieszczenie było wyposażone w zawieszony na ścianie router Wi-Fi, który został wykorzystany jako centrum do komunikowania urządzeń. Pozostałymi wymaganymi jednostkami były:

- komputer wbudowany z anteną ESPAR, zaznaczony na rysunku jako ESPAR;
- próbnik będący telefonem typu smartphone na statywie na wysokości ok. 1 m;
- stacja pomiarowa czyli laptop komunikujący się z urządzeniem wbudowanym poprzez interfejs webowy.

Zadaniem stacji pomiarowej była inicjalizacja komputera wbudowanego poprzez odpowiednie komendy zdalne. Próbnik porozumiewał się z komputerem wbudowanym za pomocą tego samego interfejsu. Komendy związane z pomiarem były wydawane z samego telefonu.



Rys. 6.2. Schematyczne rozmieszczenie umeblowania i urządzeń w pomieszczeniu. W tle różowym kolorem oznaczono przestrzeń pomiarową

Inicjalizacja urządzeń pomiarowych polega na podłączeniu próbnika i komputera wbudowanego do sieci bezprzewodowej zewnętrznego routera znajdującego się w sali. Za pomocą tej sieci próbnik jest w stanie nawiązać połączenie z komputerem wbudowanym, a dokładniej z serwerem HTTP. Pozwala to próbnikowi na pobranie strony internetowej interfejsu, oraz połączenie *socketów* odpowiedzialnych za sztuczny ruch.

6.2. Procedura pomiarowa

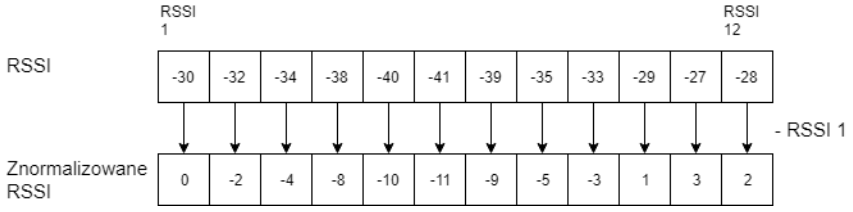
Pomiar polegał na pobraniu zestawu dwunastu RSSI w każdym punkcie pomiarowym. Dodatkowo, na każdym punkcie pobrano wyniki dla każdej z orientacji telefonu. Wybrano 4 orientacje oznaczające 4 obroty płasko leżącego telefonu ekranem do góry, w 4 kierunkach świata. Dla każdej orientacji wykonano 30 powtórzeń pomiaru, zbierając 30 zestawów RSSI. Ostatecznie w każdym punkcie zebrano 120 zestawów 12-stu RSSI. Wszystkie te pomiary

zostały zapisane w bazie danych. Dane pobrano z bazy danych w formie pliku csv. Są to surowe dane, które zostaną wykorzystane do stworzenia map fingerprintingu i testowania dokładności lokalizacji.

6.3. Wyznaczanie lokalizacji

Wszelka obróbka uzyskanych wyników pomiarów została przeprowadzona w środowisku obliczeniowym *Octave*. Zebrane próbki RSSI zapisano w macierzy pięciowymiarowej M[x,y,p,n,i]. Indeksy macierzy oznaczają kolejno współrzędne *x* i *y*, orientację *p*, numer pomiaru *n* oraz numer konfiguracji anteny *i*.

Na danych została wykonana mediana po numerze pomiaru, tak więc 30 powtórzeń pomiaru zostało zredukowane do jednego, uśrednionego. Następnie wykonana została normalizacja RSSI względem wartości pierwszej konfiguracji. Normalizacja polega na odjęciu pierwszej wartości RSSI od wszystkich wartości, tak jak pokazano to na rys 6.3. Tak przygotowane dane są gotowe do obliczania lokalizacji.



Rys. 6.3. Przykład normalizacji zestawu RSSI

Przy testowaniu algorytmu konieczne jest rozdzielenie punktów na referencyjne i testowe. Dla stworzonego środowiska pomiarowego zdefiniowano 3 mapy o różnym zagęszczeniu punktów referencyjnych. Mapy przedstawiono na rys 6.4, gdzie kolorem pomarańczowym oznaczono punkty referencyjne. Na podstawie pomiarów w tych punktach, algorytm będzie liczył lokalizację. Pozostałe punkty są punktami testowymi - w nich zostaną obliczone lokalizacje w celu określenia dokładności algorytmu.

Na podstawie wyników lokalizacji w każdym punkcie testowym, obliczono błąd lokalizacji według wzoru 6.1. Punkt rzeczywisty i punkt otrzymany z obliczeń można połączyć linią prostą. Odcinek jaki te punkty tworzą jest błędem lokalizacji wyrażonym w metrach.

$$E = \sqrt{(x_{real} - x)^2 + (y_{real} - y)^2} \tag{6.1}$$

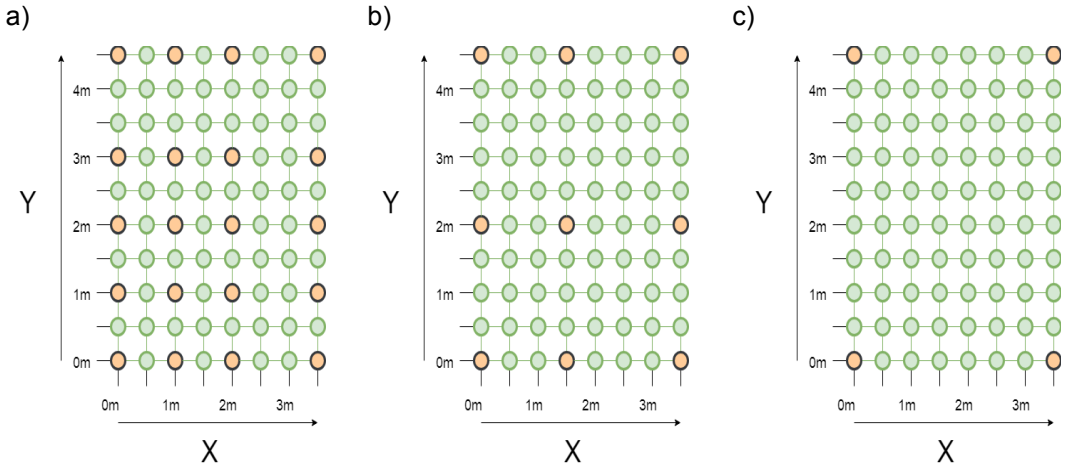
gdzie:

x_{real} – współrzędna x punktu rzeczywistego,

x – współrzędna x punktu obliczonego,

y_{real} – współrzędna y punktu rzeczywistego,

y – współrzędna y punktu obliczonego.



Rys. 6.4. Testowe mapy *fingerprintingu*. Punkty referencyjne oznaczono kolorem pomarańczowym. a) mapa 1, b) mapa 2, c) mapa 3

6.4. Wyniki

Za pomocą skryptu *Octave* wykonano szereg badań w celu sprawdzenia wpływu parametrów algorytmu k-NN i środowiska pomiarowego na dokładność lokalizacji. Wybrane parametry to liczba sąsiadów, ilość punktów referencyjnych na mapie oraz wykorzystanie współrzędnej *p* (orientacji telefonu). Na początku badań wszystkie parametry będą miały wartości domyślne:

- liczba sąsiadów n=3;
- mapa nr. 1, gdzie punkty testowe to wszystkie punkty poza referencyjnymi;
- wykorzystanie *p* – uśrednienie po wszystkich wartościach *p*.

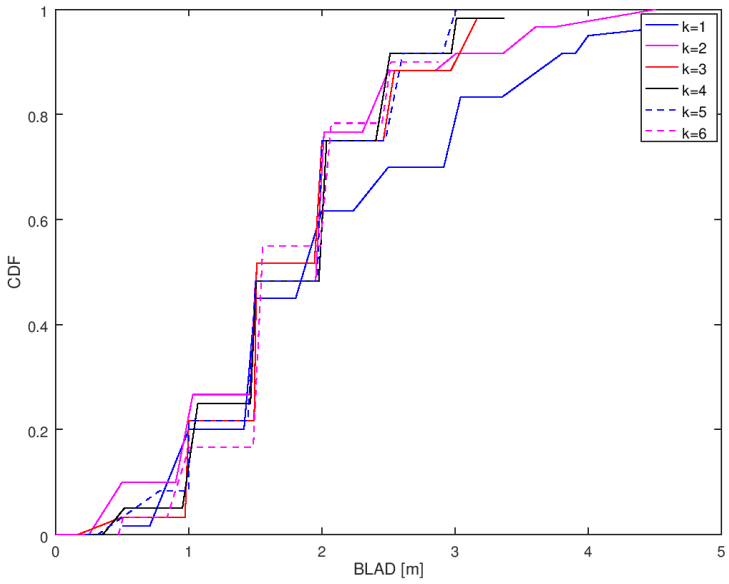
W ciągu badań, wartości parametrów zapewniających największą dokładność lokalizacji zastąpią wartości początkowe.

Wyniki będą prezentowane za pomocą CDF (ang. Cumulative Distribution Function), czyli funkcji dystrybucji prawdopodobieństwa. CDF zmiennej losowej X, oznacza rozkład prawdopodobieństwa przyjęcia przez X konkretnej wartości x lub wartości mniejszej [31]. Dla rozważanego przypadku CDF zostanie użyte na zbiorze wartości błędów lokalizacji. Zapewni to informację o częstości występowania wszystkich wartości błędów, co pozwoli na ocenę dokładności lokalizacji. Lokalizacja jest tym bardziej dokładna, im bardziej stromy jest wykres CDF błędu.

6.5. Przemiatanie liczby sąsiadów

Liczba sąsiadów oznacza ile punktów referencyjnych zostaje wzięte pod uwagę podczas estymacji pozycji telefonu. W teorii ilość sąsiadów powinna zwiększyć dokładność metody. Wynika to z tego, że zestawy RSSI otrzymane w fazie offline dla sąsiednich punktów pomiarowych powinny być podobne. Miarą podobieństwa zestawów RSSI jest różnica *D*.

Z drugiej strony, jeśli liczba sąsiadów będzie zbyt duża, może dojść do sytuacji w której algorytm znajdzie punkty o najniższym *D* również daleko od rzeczywistej pozycji. Takie odległe punkty wpłyną negatywnie na wynik lokalizacji. Dokładność metody zależy więc od stosunku znalezionych punktów, które są blisko celu, do znalezionych punktów, które są od tego celu daleko. Z tego powodu odpowiedni wybór parametru *k* jest kluczowy dla dokładności algorytmu.



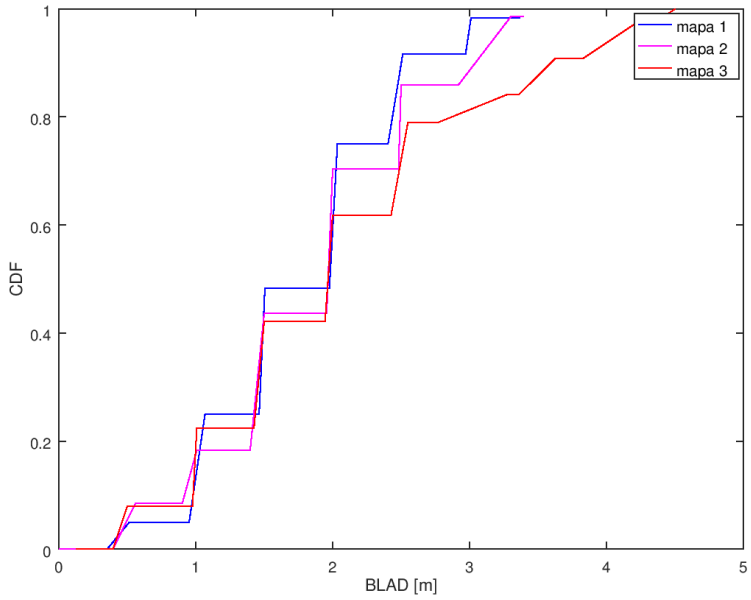
Rys. 6.5. Wykres CDF przedstawiający różnice między symulacjami z różnymi wartościami k (1-6)

Uzyskane wyniki przedstawia wykres na rys 6.5. Praktyczne testy wykazały poprawność teorii do pewnego stopnia. Faktycznie dla wzrostu *k* wykres staje się bardziej stromy, za czym idzie zwiększenie dokładności lokalizacji. Dla *k* równego 3 lub 4 efekt ten nasyca się a dokładność lokalizacji przestaje rosnąć ze wzrostem *k*. Z tego powodu wybrano *k*=4 jako najlepszą opcję i ta wartość parametru zostanie użyta w następnych badaniach.

6.6. Przemiatanie gęstości mapy

Zagęszczenie mapy *fingerprintingu* , czyli zwiększenie ilości punktów na mapie powinno zwiększyć dokładność lokalizacji. Można by stwierdzić, że jest to prosty sposób na zwiększenie dokładności, jednak może on być problematyczny. Zwiększona ilość punktów oznacza dłuższą fazę offline, czyli kalibrację systemu. Dla przypadku pomiarów w jednym pomieszczeniu nie

stanowi to dużego problemu, jednak chcąc mapować całe budynki, czas kalibracji ma ogromne znaczenie. Sprawdzenie rzędu wielkości wzrostu dokładności lokalizacji wraz ze wzrostem ilości punktów jest więc ważnym badaniem.



Rys. 6.6. Wykres CDF przedstawiający różnice między wynikami lokalizacji uzyskanymi dla trzech różnych map kalibracyjnych. Mapy te zostały wprowadzone w podrozdziale „Wyznaczanie lokalizacji".

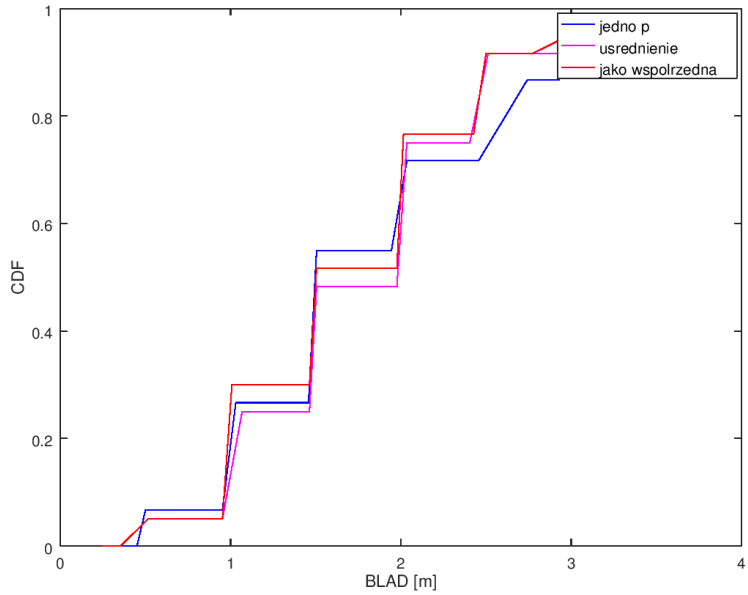
Uzyskane wyniki przedstawia wykres na rys 6.6. Wedle przypuszczeniom, dokładność lokalizacji rośnie ze wzrostem ilości punktów. Najrzadsza mapa ma najbardziej obarczony błędami wynik. W idealnym przypadku dąży się do zmniejszenia ilości punktów w celu ułatwienia kalibracji. W tym wypadku jednak wzrost dokładności jest zbyt duży by pozwolić na takie ułatwienie. Do następnych badań zostanie użyta mapa nr. 1.

6.7. Sposób wykorzystania pomiarów dla różnych orientacji terminala mobilnego

Teoria k-NN nie precyzuje jednoznacznie wybranego wykorzystania współczynnika *p*, czyli orientacji terminala mobilnego [10]. Jest więc kilka opcji, które mogą poprawić lub pogorszyć dokładność algorytmu zależnie od sposobu wykorzystania współrzędnej *p*. W najlepszym wypadku, tak jak w przypadku ilości punktów, najlepszym rozwiązaniem dla czasu kalibracji byłoby mierzenie tylko jednej orientacji. Należy więc sprawdzić czy wykorzystanie współrzędnej p poprawi dokładność lokalizacji. Sprawdzono 3 opcje wykorzystania *p*:

- wybór jednego *p*, równoważne z nie wykorzystywaniem oddzielnych orientacji;
- uśrednienie *p*, gdzie liczba orientacji nie wpływa na ilość punktów na mapie;
- użycie *p* jako kolejnej współrzędnej przez co w każdym punkcie ma napie zostaną stworzone dodatkowe punkty w "wymiarze orientacji".

Punkty stworzone ze współrzędnej *p* są traktowane przez algorytm jako oddzielne punkty. Jest więc możliwe że algorytm wybierze ten sam punkt referencyjny jako punkt najbardziej zbliżony do rzeczywistego.



Rys. 6.7. Wykres CDF przedstawiający różnice między symulacjami
w których użyto parametru *p* na 3 sposoby:
1) wybor jednego *p*, 2) uśrednienie wielu *p*, 3) p jako kolejna współrzędna.

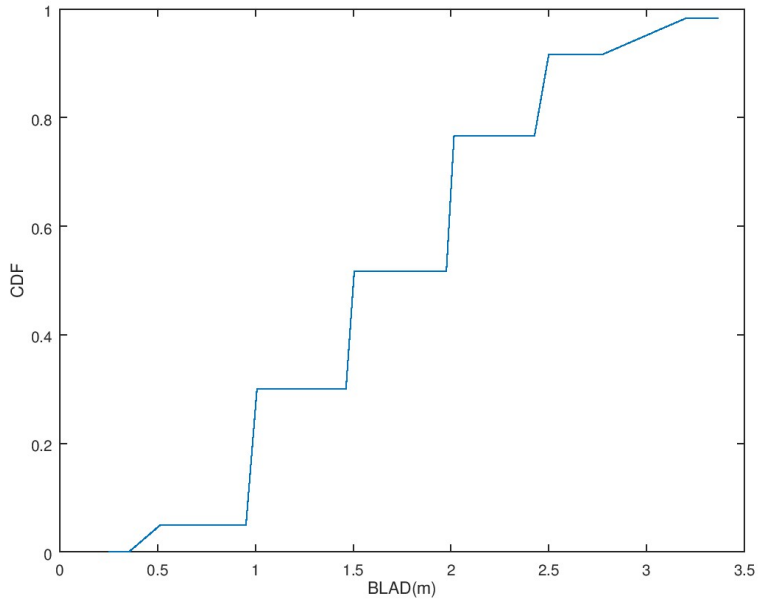
Uzyskane wyniki przedstawia wykres na rys 6.7. Brak wykorzystania współrzędnej *p* osłabia dokładność lokalizacji. Widać jednak, że dokładność nie jest wyraźnie zmieniona zależnie od sposobu wykorzystania *p*. Wynika z tego, iż same zwiększenie ilości danych, niezależnie od ich wykorzystania, zwiększa dokładność lokalizacji. Do następnych badań wybrano opcję trzecią.

6.8. Wnioski

Wybierając najlepsze parametry dla algorytmu uzyskano finalny wykres CDF, przedstawiony na rys 6.8. Jak łatwo zauważyć, zwiększenie dokładności lokalizacji wiąże się głownie ze zwiększeniem ilości danych dla algorytmu k-NN. Jedynym programowym polepszeniem dokładności jest zwiększenie liczby sąsiadów.

Uzyskana dystrybucja ukazuje umiarkowanie dobrą dokładność algorytmu. Błąd mniejszy niż metr ma zaledwie ⅓ punktów testowych. W 80% przypadków można jednak liczyć się z błędem poniżej 2 metrów. Dodatkowo obliczono błąd średni który wyniósł 1.5188m. Wartość ta jest zbliżona do błędu średniego uzyskanego w [4]. Błąd średni systemu w porównaniu do wielkości pomieszczenia może być niezadowalającym wynikiem. Wartościowym badaniem byłoby sprawdzenie jak zachowuje się system gdy obszar pomiarowy jest dużo

większy. Taki pomiar mógłby zmniejszyć znaczenie fluktuacji RSSI poprzez samą skalę zmian siły sygnału na większych odległościach.



Rys. 6.8. Wybrany wykres CDF z najmniejszym błędem lokalizacji.

Ostateczna dokładność jest mimo wszystko zadowalająca jeśli brać pod uwagę wcześniejsze wyniki z [4]. Dodatkowo, system działa wewnątrz budynków lepiej niż systemy GNSS, które są niemiarodajne w takich środowiskach.

7. PODSUMOWANIE

W ramach pracy został zrealizowany projekt urządzenia wbudowanego realizującego funkcjonalność lokalizacji terminali mobilnych wewnątrz budynków. Celem pracy było zbadanie możliwości takiego systemu. Wynikowy projekt był ograniczony do minimalnej funkcjonalności umożliwiającej testy.

Głównym wyzwaniem podczas realizacji projektu były poszukiwania odpowiedniego monitora sieci bezprzewodowej. Zastosowanie anteny ESPAR zamiast sieci odbiorników, zmienia działanie systemu lokalizacji z równoległego na sekwencyjne. Z tego powodu szybkość zbierania pakietów z sieci była wąskim gardłem systemu. Zaimplementowanie własnego monitora sieci zdjęło ograniczenia czasowe i umożliwiło wykorzystanie pełnego potencjału anteny w lokalizacji.

Równie problematyczny był brak pakietów transmitowanych przez terminale mobilne w stanie spoczynku. O ile implementacja sztucznego ruchu dla konkretnego urządzenia nie jest czymś trudnym, to oprogramowanie realizujące tą funkcjonalność dla każdego rodzaju terminala już jest. W trakcie prac okazało się, że takie rozwiązanie jest możliwe przy użyciu technologii webowych.

Rozwiązanie obydwu problemów umożliwiło stworzenie unikalnego systemu wbudowanego, który pozwala na zlokalizowanie użytkownika za pomocą dowolnego urządzenia, pracującego w standardzie Wi-Fi. To wszystko zostało osiągnięte przy użyciu tylko jednego węzła lokalizującego, który przy pomocy anteny ESPAR jest w stanie zebrać potrzebne dane.

Stworzony system może wciąż zostać ulepszony ze względu na jego możliwości i efektywność. Dokładność systemu może być poprawiona poprzez zastosowanie bardziej złożonych algorytmów *fingerprintingu*. Oprócz tego, problemy takie jak rozłączność AP i lokalizatora również mogą zostać rozwiązane. Ostateczna wersja systemu powinna również sprawnie lokalizować wiele urządzeń w tej samej chwili.

System może pełnić rolę lokalnego IPS dla systemów GNSS. Fuzja danych z systemem lokalnym może polepszyć dokładność systemu GPS w środowiskach o wielu odbiciach. Może to doprowadzić do powstania dokładniejszych systemów nawigacji, bezpieczeństwa i lokalizacji obiektów. Połączenie systemów globalnych z lokalnymi mogłoby scentralizować wszystkie tego typu usługi w jedną globalną sieć. Dziedzina lokalizacji wciąż jednak potrzebuje więcej badań i idei, które obniżą koszt i zwiększą efektywność takich systemów.

WYKAZ LITERATURY

1. Lingwen Zhang, Cheng Tao and Gang Yang (April 26th 2011). *Wireless Positioning: Fundamentals, Systems and State of the Art Signal Processing Techniques, Cellular Networks Agassi Melikov*, IntechOpen, DOI: 10.5772/14841, https://www.intechopen.com/books/cellular-networks-positioning-performance-analysis-reliability/wireless-positioning-fundamentals-systems-and-state-of-the-art-signal-processing-techniques, s. 3, (data dostępu: 14.08.2018 r.).
2. Ian Sharp, Kegen Yu: *Wireless Positioning: Principles and Practice*, Springer, 2018, s. 1-5.
3. gps.gov – oficjalna strona systemu GPS, https://www.gps.gov/systems/gps/performance/accuracy/, (data dostępu: 14.08.2018 r.)
4. M. Rzymowski, P. Woznica, and L. Kulas: *Single-Anchor Indoor Localization Using ESPAR Antenna*, In Proc. IEEE Antennas and Wireless Propagation Letters (Volume: 15), 09 November 2015, s 1183 – 1186.
5. Jill K. Nelson, George Mason: *A Sequential Detection Approach to Indoor Positioning Using RSS-Based Fingerprinting*, In Proc. 2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP), DOI: 10.1109/GlobalSIP.2016.7906017, 2016.
6. Mohd Ezanee Rusli, Mohammad Ali, Norziana Jamil, Marina Md Din: *An Improved Indoor Positioning Algorithm Based on RSSI-Trilateration techniquefor Internet of Things (IOT)*, In Proc. 6th International Conference on Computer and Communication Engineering, 2016, s 72-77.
7. David Törnqvist, *Proximity vs True Location*, August 25th, 2015, https://senion.com/insights/point-positioning-vs-true-location/, (data dostępu: 14.08.2018 r.)
8. Tavish Srivastava, *Introduction to k-Nearest Neighbors: Simplified (with implementation in Python)*, 26.03.2016, analyticsvidhya.com, https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/, (data dostępu: 26.07.2018 r.).
9. Rzymowski M., Kulas Ł.: *Design, Realization and Measurements of Enhanced Performance 2.4 GHz ESPAR Antenna for Localization in Wireless Sensor Networks*, EuroCon 2013, 1-4 July 2013, Zagreb, Croatia, s.1-3.
10. Alan Bensky, *Wireless Positioning Technologies and Applications* ,s.147-153.
11. *Data Link Layer (Layer 2)*, tcpipguide.com, http://www.tcpipguide.com/free/t_DataLinkLayerLayer2.htm, (data dostępu: 14.01.2018 r.).
12. *Link-Layer header types*, Tcpdump.org - oficjalna strona programu Tcpdump, http://www.tcpdump.org/linktypes.html, (data dostępu: 12.03.2017 r.).
13. Wireless.wiki.kernel.org, https://wireless.wiki.kernel.org/en/developers/documentation/radiotap, (data dostępu: 13.01.2018 r.).
14. Radiotap.org - oficjalna strona Radiotap, http://www.radiotap.org/, (data dostępu: 13.01.2018 r.).
15. Tim Carstens, Guy Harris, *Programming with pcap*, Tcpdump.org - oficjalna strona programu Tcpdump, http://www.tcpdump.org/pcap.html, (data dostępu: 12.03.2017 r.).
16. Dokumentacja libpcap, Tcpdump.org - oficjalna strona programu Tcpdump, http://www.tcpdump.org/manpages/pcap.3pcap.html, (data dostępu: 12.03.2017 r.).
17. Plik Pcap.h z biblioteki libpcap, https://github.com/the-tcpdump-group/libpcap/blob/master/pcap/pcap.h, (data dostępu: 12.03.2017 r.).

18. *NetBSD Kernel Developer's Manual*, March 12, 2006, netbsd.gw.com, http://netbsd.gw.com/cgi-bin/man-cgi?ieee80211_radiotap+9+NetBSD-current, (data dostępu: 12.03.2017 r.).

19. Andy Green, *How to use radiotap headers*, Kernel.org – Archiwum kerneli systemu Linux, https://www.kernel.org/doc/Documentation/networking/radiotap-headers.txt, (data dostępu: 12.03.2017 r.).

20. Sumanth Kavuri, *Understanding the Address Fields in 802.11 frames*, 21.09.2013, Blogspot.com, http://80211notes.blogspot.com/2013/09/understanding-address-fields-in-80211.html, (data dostępu: 12.03.2017 r.).

21. Matthew Gast, *802.11 Wireless Networks: The Definitive Guide*, "O'Reilly Media, Inc.", 2005, s. 47-50.

22. Tutorialspoint.com – dokumentacja i samouczek Node.js, https://www.tutorialspoint.com/nodejs/nodejs_npm.htm, (data dostępu: 26.07.2018 r.)

23. Expressjs.com – oficjalna strona *frameworku* Express, https://expressjs.com/, (data dostępu: 26.07.2018 r.).

24. Socket.io – oficjalna strona biblioteki socket.io, https://socket.io/, (data dostępu: 26.07.2018 r.).

25. Nodejs.org – dokumentacja *frameworku* Node.js, https://nodejs.org/api/net.html#net_net, (data dostępu: 26.07.2018 r.).

26. Michael Kerrisk, *Linux Programmer's Manual*, 2017-09-15, man7.org – dokumentacja systemu Linux, http://man7.org/linux/man-pages/man3/termios.3.html, (data dostępu: 21.06.2018 r.).

27. Steve Friedl, *Understanding UNIX termios VMIN and VTIME*, Unixwiz.net, http://unixwiz.net/techtips/termios-vmin-vtime.html, (data dostępu: 26.07.2018).

28. Ian Poole, *Wi-Fi / WLAN Channels, Frequencies, Bands & Bandwidths*, radio-electronics.com, https://www.radio-electronics.com/info/wireless/wi-fi/80211-channels-number-frequencies-bandwidth.php, (data dostępu: 14.08.2018 r.).

29. Stackexchange.com, http://networkengineering.stackexchange.com/questions/6534/does-wifi-router-ap-change-the-working-channel-automatically-during-the-normal-u, (data dostępu: 15.10.2017 r.).

30. ask.wireshark.org – forum oficjalnej strony programu Wireshark, https://ask.wireshark.org/questions/27463/how-to-capture-wireless-traffic-on-a-particular-channel-and-access-point, (data dostępu: 15.10.2017 r.).

31. *Cumulative Distribution Function CDF*, statisticshowto.com, http://www.statisticshowto.com/cumulative-distribution-function/, (data dostępu: 14.08.2018 r.).

WYKAZ RYSUNKÓW

WYKAZ TABEL