

Praca magisterska

Aplikacja do pomiarów parametrów łącza sieciowego
dla urządzeń z systemem operacyjnym Android

Tomasz Łakomy

Streszczenie

Abstract

Spis treści

1	Wstęp	3
2	Opis teoretyczny zastosowanych technologii	4
2.1	Transmisja danych w sieciach komórkowych	4
2.1.1	GSM i EDGE	4
	GSM	4
	EDGE	6
2.1.2	UMTS i HSDPA	8
	UMTS	8
	WCDMA	9
	HSDPA	10
2.1.3	LTE	11
2.2	Wybrane protokoły transmisji danych	12
2.2.1	Protokół HTTP	12
2.2.2	Protokół TCP	14
2.2.3	Protokół UDP	15
2.3	Charakterystyka tworzenia oprogramowania na urządzenia z systemem Android	16

2.4	Charakterystyka tworzenia oprogramowania w języku Python	18
2.5	Aktualnie dostępne narzędzia do pomiarów parametrów transmisji dla urządzeń z systemem Android	19
3	Realizacja pracy	20
3.1	Założenia projektowe	20
3.2	Opis aplikacji pełniącej funkcję serwera na komputer PC	23
3.2.1	Protokół HTTP	23
3.2.2	Protokół TCP	26
3.2.3	Protokół UDP	28
3.3	Opis aplikacji na smartfon z systemem Android	29
3.3.1	Opis realizacji programowej transmisji z wykorzystaniem zapytań HTTP	34
3.3.2	Opis realizacji programowej transmisji pakietów poprzez protokół TCP	38
3.3.3	Opis realizacji programowej transmisji pakietów poprzez protokół UDP	41
3.3.4	Sposób pomiaru czasu przesyłania pakietu	43
3.3.5	Sposób pomiaru położenia terminala mobilnego	45
3.3.6	Sposób zapisywania wyników pomiaru do pliku	48
4	Przykładowe wyniki pomiarów	53
4.1	Pomiar nieruchomego terminala mobilnego	53
4.2	Pomiar nieruchomego terminala mobilnego (nadanie 200 pakietów, sieć LTE)	54
4.3	Pomiar nieruchomego terminala mobilnego (nadanie 200 pakietów, sieć WCDMA)	56

4.4	Pomiar nieruchomego terminala mobilnego (nadanie 200 pakietów, sieć EDGE)	58
5	Podsumowanie	60

Rozdział 1

Wstęp

Rozdział 2

Opis teoretyczny zastosowanych technologii

2.1 Transmisja danych w sieciach komórkowych

2.1.1 GSM i EDGE

GSM

GSM (ang. *Global System for Mobile Communications*) jest standardem, który powstał dzięki europejskiej inicjatywie stworzenia jednego, otwartego standardu telefonii komórkowej. Jest to najstarsza wykorzystywana dziś technologia radiokomunikacji ruchomej, na obszarze europejskim rozpoczęto uruchamianie GSM już w roku 1989, rok po opublikowaniu pierwszej wersji standardu. Polska na uruchomienie pierwszej sieci GSM czekała kolejne 7 lat, została ona uruchomiona w roku 1996.

GSM jest aktualnie najpowszechniej wykorzystywanym standardem telefonii komórkowej na świecie, jest on dostępny w 219 państwach. Pierwotnie był to standard pozwalający jedynie na transmisję mowy, jednakże po latach ewolucji pojawiły się bazujące na GSM technologie transmisji danych takie jak GPRS (ang. *General Packet Radio Service*) czy jego następca, EDGE (ang. *Enhanced Data Rates for GSM evolution*).

W przypadku, gdy użytkownik terminala mobilnego wybrał korzystanie np. z sieci LTE (ang. *Long Term Evolution*) do transmisji danych i z jakiś powodów transmisja ta nie jest możliwa (przykładowo z powodu zbyt słabego zasięgu sieci LTE), transmisja może być przeprowadzona za pomocą technologii EDGE, która jest dostępna niemalże wszędzie tam, gdzie dostępna jest sieć GSM.

Tabela 2.1 Porównanie standardów sieci GSM

Standard	Częstotliwości wykorzystywane w łączu w górę [MHz]	Częstotliwości wykorzystywane w łączu w dół [MHz]	Liczba dostępnych pasm częstotliwości
GSM 400	450,4 - 457,6 lub 478,8 - 486	460,4 - 467,6 lub 488,8 - 496	35
GSM 850	824 - 849	869 - 894	124
GSM 900	880 - 915	925 - 960	174
GSM 1800	1710 - 1785	1805 - 1880	374
GSM 1900	1850 - 1910	1930 - 1990	299

Istnieje pięć głównych standardów sieci GSM, różniących się od siebie wykorzystywanym pasmem radiowym oraz liczbą dostępnych pasm częstotliwości, co przedstawia tabela 2.1

Na terenie Unii Europejskiej używany jest standard GSM 900/1800, który polega na uruchomieniu obu sieci jednocześnie na danym obszarze. Na terenach, gdzie spodziewany ruch jest niezbyt duży (np. tereny wiejskie) uruchamiana jest tylko sieć GSM 900, jednakże w miastach, gdzie ruch ten jest zdecydowanie większy, dodatkowo wdrażany jest także standard GSM 1800, który dzięki większej liczbie jednocześnie oferowanych częstotliwości jest w stanie obsłużyć większy ruch.

Aktualnie niemalże wszystkie dostępne na rynku telefony komórkowe pozwalają na pracę w obydwu zakresach częstotliwości, co sprawia, że użytkownik nie musi się obawiać o utratę zakresu np. sieci GSM 1800. GSM zakłada możliwość rozmowy w trakcie przemieszczania się pomiędzy stacjami bazowymi.

EDGE

Standard EDGE (ang. *Enhanced Data Rates for GSM Evolution*) powstał jako odpowiedź na zapotrzebowanie użytkowników na większe niż w przypadku GPRS prędkości transmisji danych pakietowych. Obecnie jest to najbardziej podstawowa technika przesyłania danych w sieciach komórkowych, wykorzystywana, gdy sieci UMTS oraz LTE nie są dostępne.

Zarówno EDGE jak i GPRS działają na bazie istniejącej infrastruktury sieci komórkowej, więc wdrożenie ich nie stwarzało konieczności budowania nowej sieci radiowej. EDGE (mimo, że powstał później) nie oferował nowych usług, ale za to oferował użytkownikom możliwości dostarczania takich usług jak Internet, korzystanie z transmisji strumieniowych audio/video czy też wideorozmowy.

W podstawowym systemie GSM, transmisja jest zorganizowana na pasmach o szerokości 200 kHz, a czas podzielony jest na 8 kolejno następujących szczelin czasowych, z których każda trwa 577 mikrosekund. Szczeliny są ponumerowane od 1 do 8 i następują cyklicznie. W przypadku transmisji mowy, kontroler stacji bazowej przypisuje terminalowi mobilnemu jedną szczelinę czasową na częstotliwości używanej do transmisji. Oznacza to, że na 1 częstotliwości można jednocześnie prowadzić do 8 rozmów telefonicznych.

Ze względu na to, że EDGE został zbudowany jako rozszerzenie standardu GSM, wykorzystuje on szczeliny czasowe na potrzeby transmisji danych. W przeciwieństwie do transmisji mowy, nie odbywa się rezerwacja szczelin czasowych na cały czas korzystania z sieci pakietowej (np. w trakcie przeglądania stron internetowych na telefonie komórkowym) - szczelina czasowa jest rezerwowana tylko na potrzeby przesłania danej paczki pakietów danych. Teoretycznie w sieci EDGE możliwe jest rezerwowanie wszystkich 8 szczelin czasowych na potrzeby transmisji pakietów, jednakże w praktyce rezerwuje się do 4 szczelin dla transmisji od terminala mobilnego i 5 szczelin dla transmisji w kierunku terminala. Wszystkie rezerwowane szczeliny dla danej transmisji muszą znajdować się na tej samej częstotliwości.

EDGE dla celów modulacji danych wykorzystuje modulację GMSK (podobnie jak GSM dla transmisji mowy), jednakże możliwe jest także wykorzystanie nowszego roz-

wiązania, jakim jest modulacja 8-PSK (ang. *8 Phase Shift Keying*), która oferuje większą przepływność, kosztem wrażliwości na warunki transmisji. EDGE zakłada 9 różnych schematów transmisji, z których każdy charakteryzuje się inną szybkością danych, co wynika z zastosowanej modulacji oraz ilości zastosowanych nadmiarowych danych (tzw. *code rate*). Tabela 2.2 przedstawia możliwe schematy transmisji. Schematy transmisji są podzielone na trzy rodziny A, B i C, których zastosowanie sprowadza się do tego, że gdy warunki dla przeprowadzenia danej transmisji są nieodpowiednie, wybierany jest inny schemat transmisji pochodzący z danej rodziny.

Tabela 2.2 Tabela przedstawiająca schematy transmisji w sieci EDGE

Schemat	Code rate	Modulacja	Transfer	Rodzina
MCS-1	0,53	GMSK	8,8 kbit/s	C
MCS-2	0,66	GMSK	11,2 kbit/s	B
MCS-3	0,85	GMSK	14,8 kbit/s	A
MCS-4	1	GMSK	17,6 kbit/s	C
MCS-5	0,37	8-PSK	22,4 kbit/s	B
MCS-6	0,49	8-PSK	29,6 kbit/s	A
MCS-7	0,76	8-PSK	47,8 kbit/s	B
MCS-8	0,92	8-PSK	54,4 kbit/s	A
MCS-9	1	8-PSK	59,2 kbit/s	A

Z powyższej tabeli wynika, że maksymalną prędkość transmisji można osiągnąć wybierając schemat MCS-9, który przy zastosowaniu 5 szczelin czasowych umożliwia transmisję do 296 kilobitów na sekundę. W praktyce prędkość ta jest zdecydowanie niższa, głównie ze względu na warunki panujące w kanale radiowym.

2.1.2 UMTS i HSDPA

UMTS

UMTS (ang. *Universal Mobile Telecommunications System*), jest to standard sieci komórkowej trzeciej generacji będący następcą systemu GSM. UMTS został zbudowany na bazie GSM, co oznacza, że nie zakłada on zmian w sieci szkieletowej, jednakże wprowadzono gruntowne zmiany w sieci radiowej. Dzięki tym zmianom (takim jak zaimplementowanie technologii HSDPA - ang. *High Speed Downlink Packet Access*) udało się uzyskać prędkości transmisji danych pakietowych dochodzące do 21,6 Mbit/s w transmisji w łączy w górę, oraz do 5,76 Mbit/s w łączy w dół.

Prędkości transmisji danych w sieciach trzeciej generacji są zdecydowanie większe od prędkości mierzonych w sieciach generacji poprzedniej ze względu na rosnące zapotrzebowanie użytkowników na korzystanie z usług internetowych w terminalach mobilnych. W czasach, gdy strumieniowanie filmów w wysokiej rozdzielczości przez Internet jest pożądaną przez użytkowników telefonów komórkowych funkcjonalnością, EDGE nie jest wystarczający dla zaspokojenia ich potrzeb.

W dzisiejszych czasach UMTS jest najpopularniejszą siecią komórkową trzeciej generacji, a polscy operatorzy komórkowi objeli jej zasięgiem niemalże cały kraj.

[4][5][6][7]



Rysunek 2.1: Zasięg sieci 3G operatora T-Mobile w wrześniu 2015 roku

WCDMA

WCDMA (ang. *Wideband Code Division Multiple Access*) jest to technika szybkiego przesyłania danych pakietowych zaimplementowana w standardzie UMTS. Pierwszy raz została ona zaimplementowana w 2001 roku, a aktualnie jest to najpopularniejsze rozwiązanie stosowane w sieciach 3G, oferujące prędkości transmisji danych do 384 kbit/s.

WCDMA bazuje na technice wielodostępu z podziałem kodowym (CDMA - *Code Division Multiple Access*), spotykanej w systemach z poszerzonym widmem [2]. W przeciwieństwie do GSM/EDGE, nie istnieje tutaj pojęcie pasm częstotliwości przydzielanych dla danej transmisji. Zamiast tego, wszystkie transmisje odbywają się na wspólnym szerokim paśmie, wykorzystywanym przez wszystkich użytkowników jednocześnie. W specyfi-

kacji standardu szerokość tego pasma wynosi 4,68 MHz, w praktyce jednak wykorzystuje się pasmo 5 MHz, aby zminimalizować efekty interferencji z innymi jednocześnie odbywającymi się transmisjami.

W systemach wykorzystujących CDMA, transmisje pochodzące od poszczególnych użytkowników sieci są przetwarzane w nadajniku za pomocą wyznaczonych kodów, które poszerzają pasmo nadawanego sygnału. Dzięki temu można umieścić wiele transmisji na jednakowym paśmie, kody te są ortogonalne wobec siebie, tak więc możliwe jest wyodrębnienie danego sygnału w odbiorniku spośród wielu innych, jednocześnie nadawanych. Ponadto, ciągi danych są modulowane z zastosowaniem modulacji cyfrowej QPSK, co także poprawia odporność transmisji na błędy.

HSDPA

HSDPA (ang. *High Speed Downlink Packet Access*) pojawiło się ze względu na wciąż rosnące zapotrzebowanie na szybką transmisję danych, prędkości oferowane przez WCDMA nie były już wystarczające dla użytkowników XXI wieku. Technologia ta powstała na bazie WCDMA, co sprawia, że operatorzy komórkowi przy jej wdrażaniu nie muszą ponosić kosztów związanych z wymianą sieci szkieletowej, a jedyne zmiany dotyczą sieci radiowej.

System HSDPA oferuje prędkości transmisji sięgające 21.6 Mbit/s w łączu w dół. Tak dużą prędkość transmisji danych udało się osiągnąć poprzez m.in. zastosowanie modulacji 16QAM (gdy warunki panujące w kanale radiowym na to pozwalają, ze względu na to, że modulacja ta jest bardziej wrażliwa na zakłócenia niż modulacja QPSK), wyodrębnienie osobnego kanału transportowego dla transmisji w łączu w dół - HS-DSCH (*High Speed Downlink Shared Channel*), oraz dzięki zmniejszeniu okresu, w którym przesyłana jest ramka danych, co pozwala na lepsze reagowanie na zmieniające się w czasie właściwości kanału radiowego.

2.1.3 LTE

Technologia LTE (ang. *Long Term Evolution*) powstała jako odpowiedź konsorcjum 3GPP (ang. *3rd Generation Partnership Project*) na rosnące zapotrzebowanie użytkowników na przepustowość łącza danych. Z racji tego, że jest ona następcą takich rozwiązań telekomunikacyjnych jak WCDMA i HSPA bywa ona bardzo często omyłkowo nazywana "technologią 4G", co nie jest do końca prawdą. W tym miejscu warto zaznaczyć, że pomimo wielu nieprawdziwych informacji docierających z mediów standard LTE nie spełnia wymogów stawianych przez ITU (ang. *International Telecommunication Union*) dla technologii 4G/IMT-Advanced. Wymogi te spełnia dopiero następca standardu LTE o nazwie LTE-Advanced.

Pierwszą implementację standardu opisuje dokument 3GPP LTE Release 8, opublikowany w grudniu 2008 roku. Opisano w nim nowo projektowany standard oraz przedstawiono jego specyfikację. Kolejny dokument (Release 9) został wydany rok później, a wydanie Release 10 zawierającego opis LTE-Advanced pozwoliło 3GPP na spełnienie wymagań dotyczących sieci 4G. W Release 8 zostały opisane podstawowe właściwości systemu LTE, które przedstawiono w tabeli 2.3

Tabela 2.3 Podstawowe właściwości systemu LTE

Parametr	Standard LTE
Max przepływność - łącze w dół	300 Mb/s
Maksymalna przepływność - łącze w górę	50Mb/s
Maksymalne opóźnienie pakietu	ok. 10ms
Wykorzystana metoda wielodostępu - downlink	OFDMA
Wykorzystana metoda wielodostępu - uplink	SC-FDMA

Aby uzyskać właściwości systemu przedstawione w tabeli 2.3 wykorzystano szereg technik:

- **MIMO** (*Multiple Input Multiple Output*) - technika umożliwiająca korzystanie z wielu anten zarówno po stronie nadawczej jak i odbiorczej.

- **SC-FDMA** (*Single Carrier – Frequency Division Multiple Access*) - metoda wielodostępu stosowana w łączu w górę pozwalająca użytkownikom na współdzielenie zasobów czasowo-częstotliwościowych.
- **HARQ** (*Hybrid Automatic Repeat reQuest*) - protokół polegający na retransmisji danych w przypadku wystąpienia trudnych warunków propagacyjnych.
- **OFDM** (*Orthogonal Frequency Division Multiplexing*) - modulacja stosowana w łączu w dół, polegająca na transmisji na wielu podnośnych, które są względem siebie ortogonalne.
- **SAE** (*System Architecture Evolution*) - poprawienie struktury szkieletowej sieci, która jest łatwa w modyfikacji i dopasowana do potrzeb przyszłego standardu LTE-Advanced.
- **SON** (*Self-Organizing Networks*) - rozwiązania pozwalające na samokonfigurację oraz samooptymalizację sieci LTE.

2.2 Wybrane protokoły transmisji danych

2.2.1 Protokół HTTP

Protokół HTTP (ang. *Hypertext Transfer Protocol*) jest podstawą działania współczesnego Internetu, ponieważ jest to protokół warstwy aplikacji odpowiedzialny za przesyłanie w Internecie stron WWW. Pracę nad nim rozpoczęto w 1989 roku w CERN (*European Organization for Nuclear Research*). Aktualnie najnowszą wersją protokołu HTTP jest HTTP/2.0, który doczekał się standaryzacji w 2015 roku.

Jednostką logiczną protokołu HTTP jest wiadomość, która składa się z zapytania od klienta do serwera oraz odpowiedzi od serwera. HTTP definiuje 9 metod, które mogą zostać wywołane na zasobie, do którego dostęp chce uzyskać klient wysyłając zapytanie HTTP. Dostępne usługi HTTP przedstawia tabela 2.4.

Tabela 2.4 Dostępne usługi protokołu HTTP

Metoda	Przeznaczenie
<i>DELETE</i>	Żądanie usunięcia zasobu z serwera
<i>GET</i>	Żądanie zasobu od serwera w postaci nagłówka danych oraz treści
<i>HEAD</i>	Żądanie zasobu od serwera w postaci nagłówka danych
<i>LINK</i>	Żądanie ustanowienia relacji między istniejącymi zasobami
<i>OPTIONS</i>	Żądanie od serwera indentyfikacji obsługiwanych metod
<i>POST</i>	Żądanie odebrania przez serwer danych do klienta
<i>PUT</i>	Żądanie odebrania przez serwer pliku od klienta
<i>TRACE</i>	Żądanie zwrócenia przez serwer nagłówków wiadomości wysłanej od klienta
<i>UNLINK</i>	Żądanie usunięcia relacji między istniejącym zasobami

Każda odpowiedź serwera na zapytanie HTTP zawiera tzw. kod stanu, który jest trzycyfrową liczbą informującą klienta o aktualnym statusie zapytania. Podział kodów stanu protokołu HTTP przedstawia tabela 2.5

Tabela 2.5 Dostępne usługi protokołu HTTP

Metoda	Rodzaj	Znaczenie
1xx	Informacyjne	Żądanie odebrane, zapytanie w trakcie przetwarzania
2xx	Sukces	Zapytanie odebrane i zakończone sukcesem
3xx	Przekierowanie	Konieczność podjęcia dalszych akcji, aby zakończyć żądanie
4xx	Błąd po stronie klienta	Żądanie ma złą składnię lub nie może zostać spełnione
5xx	Błąd po stronie serwera	Serwer nie może poprawnie obsłużyć żądania

2.2.2 Protokół TCP

Protokół TCP (ang. *Transmission Control Protocol*) jest połączeniowym, niezawodnym, strumieniowym protokołem stanowiącym podstawę działania dzisiejszego Internetu

[10]. Po raz pierwszy został on zdefiniowany w dokumencie RFC 793 z 1981 roku zatytułowanym ” *Transmission Control Protocol. Darpa Internet Program protocol specification*” [8]. Jest on protokołem warstwy transportowej modelu ISO, podobnie jak UDP.

Fakt, że protokół TCP jest protokołem niezawodnym i połączeniowym oznacza, że weryfikuje on, czy dane zostały dostarczone przez sieć poprawnie i w odpowiedniej kolejności. Dzięki temu, dane przesyłane za pośrednictwem TCP można traktować jako ciągły strumień bez obaw o ich utratę w trakcie transmisji. Za niezawodność transmisji płaci się jednak pewną cenę - transmisja TCP jest bardziej skomplikowana niż np. transmisja UDP ze względu na możliwość retransmisji pakietów oraz większym rozmiarem nagłówka.

Jednostką przesyłanych danych dla protokołu TCP jest segment, który składa się z 32-bitowych słów nagłówka oraz przesyłanych danych. Maksymalna dopuszczalna długość segmentu TCP wynosi 65 535 bajty, jednakże w zdecydowanej większości transmisji jest ona zdecydowanie mniejsza (np. dla transmisji Ethernet jest to 1500 bajtów).

Nagłówek segmentu TCP składa się z następujących pól:

- **Port źródłowy** [16 bitów]
- **Port przeznaczenia** [16 bitów]
- **Numer kolejny** [32 bity] - numer kolejny pierwszego bajtu danych w tym segmencie. Służy do numerowania bajtów w całym połączeniu, w celu zapewnienia ich poprawnej kolejności
- **Numer potwierdzenia** [32 bity] - pole to zawiera wartość kolejnego oczekiwanego numeru kolejnego. Wartość ta jest potwierdzeniem odebrania wszystkich bajtów o numerach kolejnych mniejszych niż wartość zapisana w tym polu.
- **Przesunięcie** [4 bity] - liczba 32 bitowych słów w aktualnie przesyłanym nagłówku TCP. Przesunięcie to wskazuje początek danych.
- **Zarezerwowane** [6 bitów] - zarezerwowane do wykorzystania w przyszłości

- **Flagi** [6 bitów] - każda o innym przeznaczeniu, przykładowo flaga ACK oznacza pole potwierdzenia, RST resetuje połączenie, a flaga FIN - kończy je
- **Okno** [16 bitów] - określa liczbę bajtów danych, które nadawca zgodzi się przyjąć. Pole te służy do sterowania przepływem danych.
- **Suma kontrolna** [16 bitów] - suma kontrolna nagłówka oraz danych
- **Wskaźnik pilności** [16 bitów] - pole używane, gdy ustawiona jest flaga URG. Zawiera numer kolejnego bajtu następującego po tzw. "pilnych danych"
- **Pola opcji** [0 - 44 bajtów] - dodatkowe opcje transmisji TCP. Listę wszystkich dostępnych opcji można znaleźć w [11]
- **Uzupełnienie** [X bitów] - uzupełnia zerami opcje tak, aby długość nagłówka TCP była wielokrotnością 32 bitów

2.2.3 Protokół UDP

Protokół UDP (ang. *User Datagram Protocol*) jest, podobnie jak TCP, protokołem warstwy transportowej OSI. W przeciwieństwie do TCP, protokół UDP realizuje usługę beipołączeniowego dostarczania danych. Oznacza to, że nie zakłada on poprawności odebrania przesłania danych oraz nie weryfikuje czy w ogóle dotarły one do celu. Dzięki czemu uzyskano zdecydowanie bardziej zawodny sposób transmisji danych, co pozwoliło zdecydowanie uprościć np. wielkość nagłówka UDP.

UDP został po raz pierwszy zdefiniowany w dokumencie RFC 768 z 1980r. zatytułowanym "*User Datagram Protocol*". Jego prostota sprawiła, że znalazł on zastosowanie w transmisji danych w przypadku której nie jest wymagana 100% pewność, że dane trafiły do adresata. Przykładem takiej transmisji może być np. transmisja filmu wideo w Internecie. Weryfikacja poprawności przesłania każdego segmentu TCP pochłania moc obliczeniową, a straty jakości obrazu wynikające z utraty części pakietów UDP są niezauważalne dla przeciętnego użytkownika. Ponadto w wielu zastosowaniach to protokoły warstw wyższych odpowiadają za gwarancję poprawności transmisji, z tego względu nie jest konieczne dublowanie tej funkcjonalności w warstwie transportowej.

Jednostką przesyłanych danych dla protokołu UDP jest pakiet, który składa się z 8 bajtów nagłówka oraz pola danych. Nagłówek UDP składa się z następujących pól:

- **Port źródłowy** [16 bitów]
- **Port docelowy** [16 bitów]
- **Długość** [16 bitów] - zawiera całkowitą długość aktualnie przesyłanego pakietu (nagłówek wraz z danymi), mierzoną w bajtach
- **Suma kontrolna** [16 bitów] - obliczana na podstawie nagłówka i danych przesyłanego pakietu

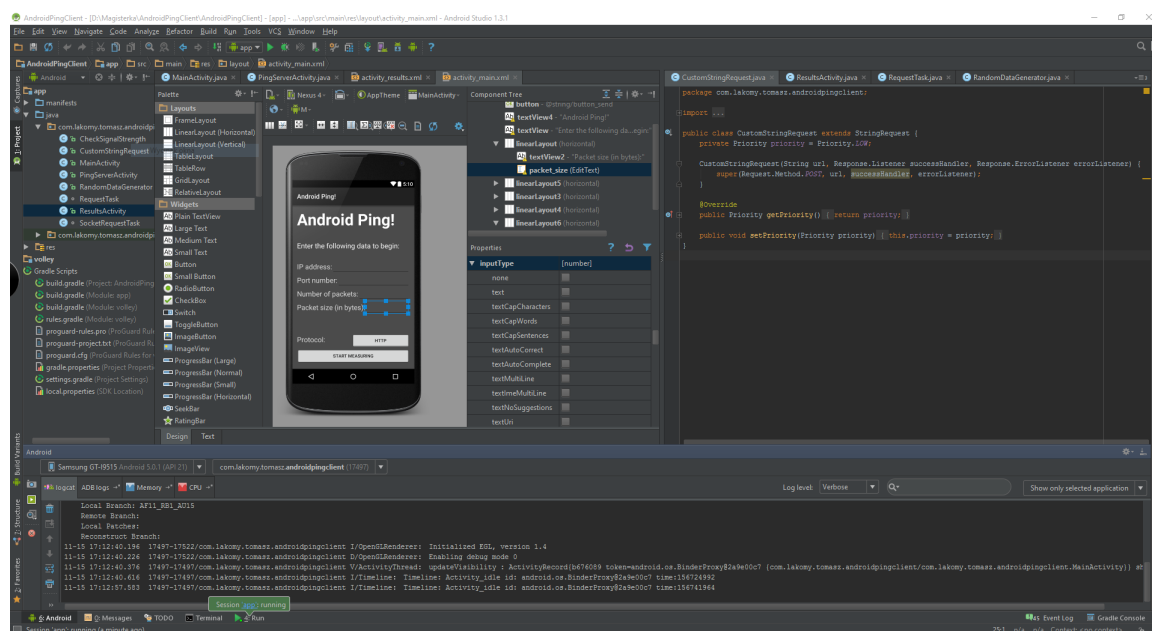
2.3 Charakterystyka tworzenia oprogramowania na urządzenia z systemem Android

Android jest systemem operacyjnym z jądrem Linux przeznaczonym dla urządzeń mobilnych takich jak telefony komórkowe, smartfony, tablety oraz przykładowo w inteligentnych telewizorach (technologia Android TV). Obecnie jest to najbardziej popularny system operacyjny na telefony komórkowe (87% udziału na rynku, dane z drugiego kwartału 2016 [12]), a w mowie potocznej określenie "smartfon" jest utożsamianie z telefonem z panelem dotykowym, na którym zainstalowany jest system operacyjny Android.

Android jest rozwiązaniem typu Open Source[9], co oznacza, że jego kod źródłowy jest powszechnie dostępny. Jest to jeden z elementów otwartej polityki aktualnego właściciela systemu operacyjnego Android - firmy Google, która rozwija Androida jako otwartą platformę zarówno dla programistów jak i dla użytkowników.

Najczęściej używanym środowiskiem programistycznym do pisania aplikacji na system operacyjny Android jest program Android Studio, przedstawiony na rysunku 2.2. Środowisko to jest dostępne na systemy operacyjne Windows, Mac OS X oraz Linux i stanowi ono kompletną platformę programistyczną dla potrzeb tworzenia aplikacji na Androida.

Zarówno system operacyjny Android jak i środowisko Android Studio wymagają zainstalowanego środowiska do programowania w języku Java, ze względu na to, że jest to główny język programowania wykorzystywany na tej platformie. Istnieją rozwiązania pozwalające kompilować i uruchamiać kod napisany w innych językach (np. Objective-C lub JavaScript) na platformie Android, jednak zaleca się tworzenie aplikacji w języku Java, ze względu na najlepsze wsparcie ze strony firmy Google.



Rysunek 2.2: Tworzenie aplikacji na system operacyjny Android w środowisku Android Studio

Poza środowiskiem programistycznym, Android Studio zawiera także wizualny edytor służący do przygotowania wyglądu aplikacji, a także wbudowany emulator, który pozwala na przetestowanie powstającej aplikacji na różnych typach urządzeń (np. tablety o różnych przekątnych ekranu) bez konieczności zakupu sprzętu.

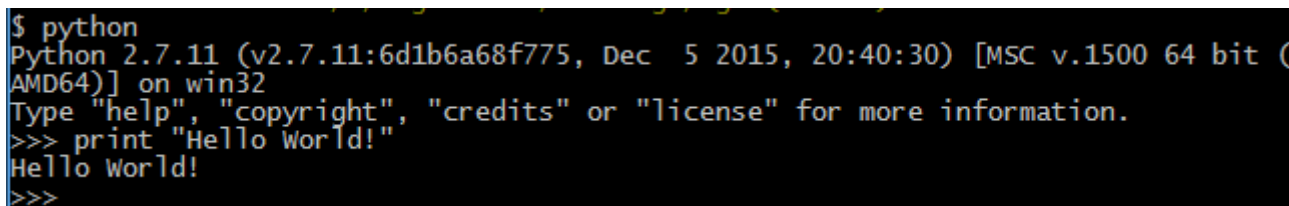
2.4 Charakterystyka tworzenia oprogramowania w języku Python

Python to powstały w 1991 roku język programowania wysokiego poziomu, którego ideą jest czystość i klarowność kodu źródłowego. Python jest językiem uniwersalnym, wieloparadygmatowym i aktualnie jednym z najpowszechniej używanych języków programowania na świecie [13].

Do zalet programowania w języku Python należy zaliczyć ogromną ilość dostępnych bibliotek, obszerną bibliotekę standardową, prostotę pisania i czytania kodu oraz możliwość uruchamiania stworzonego w tym języku oprogramowania na wielu różnych platformach. Python jest językiem interpretowanym, co oznacza, że do uruchomienia programu napisanego w tym języku nie jest wymagana kompilacja. Pozwala to na zwiększenie szybkości pracy nad oprogramowaniem, a różnice w prędkości działania programów napisanych w Pythonie i tych napisanych w językach kompilowanych nie są widoczne w większości popularnych zastosowań.

Python jest obecnie wykorzystywany do tworzenia zarówno samodzielnych programów i gier, jak i aplikacji skryptowych o wielu różnych zastosowaniach. Znajduje zastosowanie także w systemach sztucznej inteligencji, nauczaniu maszynowym czy w programach realizujących transmisję z wykorzystaniem protokołów sieciowych.

Interpreter języka Python jest dostępny do pobrania w Internecie na oficjalnej stronie tego języka [14] w wersjach na komputery z systemem Windows, Linux, MacOS, a także z zdecydowanie rzadziej wykorzystywanymi obecnie systemami takimi jak MS-DOS, Solaris czy BeOS.



```
$ python
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec 5 2015, 20:40:30) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello world!"
Hello world!
>>>
```

Rysunek 2.3: Interpreter języka Python

Python jest językiem interpretowanym, co pozwala na pisanie programów w tym języku nawet na stronie python.org, gdzie dostępny jest Python Interpreter Online. Pozwala on na naukę języka, a także na pisanie własnych prostych programów bez pobierania jakiegokolwiek dodatkowego oprogramowania (rysunek 2.3). W celu pisania bardziej skompilowanych programów zalecane jest pobranie interpretera oraz edytora tekstu takiego jak PyCharm czy SublimeText.

2.5 Aktualnie dostępne narzędzia do pomiarów parametrów transmisji dla urządzeń z systemem Android

Rozdział 3

Realizacja pracy

3.1 Założenia projektowe

Celem pracy jest stworzenie środowiska pomiarowego do pomiaru czasu transmisji danych w sieciach komórkowych pod kątem zastosowania tych sieci w systemach pomiarowych.

W tym celu należy opracować aplikację dla urządzeń z systemem Android, a także dedykowane dla niej serwery napisane w języku Python. Serwery te umożliwiają komunikację z aplikacją zainstalowaną na terminalu mobilnym z wykorzystaniem protokołów HTTP, TCP oraz UDP.

Aplikacja na system Android umożliwia wybór następujących parametrów pomiaru:

- Adres IP serwera
- Port, na którym nasłuchuje uruchomiony uprzednio serwer
- Liczbę pakietów, która zostanie przesłana w trakcie pomiaru
- Rozmiar (w bajtach) pojedynczego przesyłanego pakietu
- Czas pomiędzy przesłaniem kolejnego pakietu. Minimalny dopuszczalny czas wynosi

500ms, użytkownik ma także możliwość wyboru pomiędzy podaniem wartości w sekundach, minutach oraz milisekundach.

- Protokół transmisji - do wyboru są protokoły HTTP, TCP oraz UDP

Pojedynczym pomiarem czasu transmisji nazywamy pomiar czasu wykonania następujących czynności:

1. Przesłanie pakietu danych o zadanej długości z terminalu mobilnego do serwera
2. Odebranie pakietu danych przez serwer
3. Odesłanie identycznego pakietu danych do terminala mobilnego
4. Odebranie pakietu danych w terminalu mobilnym

Program umożliwia retransmisję pakietu o zadanej wielkości co N czasu (N jest wartością podaną przez użytkownika). W przypadku transmisji za pośrednictwem protokołu TCP przy zadany interwale mniejszym niż minuta program realizuje transmisję w trakcie jednej sesji TCP. Jeżeli czas ten jest dłuższy niż minutę, program przy każdej transmisji otwiera nową sesję TCP.

W trakcie dokonywania pomiaru, program wyświetla następujące dane:

- Aktualne położenie terminala mobilnego na mapie (za pośrednictwem Google Maps). W zależności od szybkości aktualnej transmisji, położenie jest zaznaczane jaśniejszym bądź ciemniejszym okręgiem. Pozwala to ocenić jakość transmisji w danym miejscu w przypadku przemieszczania się w trakcie pomiaru.
- Numer aktualnie przesyłanego pakietu
- Liczbę pakietów, które zostaną przesłane w trakcie aktualnego pomiaru
- Aktualną sieć (LTE/WCDMA/EDGE)
- Aktualną moc sygnału (w dBm)
- Czas transmisji aktualnie wysłanego pakietu

- Mediana czasu transmisji pakietu
- Odchylenie ćwiartkowe czasu transmisji pakietu
- Minimalny czas transmisji pakietu w trakcie aktualnego pomiaru
- Maksymalny czas transmisji pakietu w trakcie aktualnego pomiaru
- Średni czas transmisji pakietu w trakcie aktualnego pomiaru

W trakcie pomiaru tworzony jest plik w formacie .csv w pamięci urządzenia. Plik ten zawiera dane o aktualnym pomiarze w formacie:

[Data, czas pomiaru, aktualna sieć, czas transmisji pakietu, moc sygnału, położenie terminala ruchomego]

Po skończonym pomiarze użytkownik ma możliwość wygenerowania nie tylko pliku z wynikami pomiarów, lecz także zapisanej w pliku .jpg mapy z zaznaczonymi punktami pomiarów oraz wykresu przedstawiającego zmiany czasów transmisji oraz mocy sygnału w trakcie pomiaru.

W celu weryfikacji działania aplikacji dokonano przykładowych pomiarów.

3.2 Opis aplikacji pełniącej funkcję serwera na komputer PC

3.2.1 Protokół HTTP

```
1  import os
2  from flask import Flask, render_template, request
3
4  app = Flask(__name__)
5
6
7  @app.route('/', methods=['GET', 'POST'])
8  def hello():
9      if request.method == 'POST':
10         return request.args.get('data', '')
11      else:
12         return 'Hello World!'
13
14  app.debug = True
15  app.run(host='0.0.0.0', port=int(os.environ['PORT']))
```

Rysunek 3.1: Kod źródłowy serwera odbierającego i nadającego przy pomocy zapytań HTTP

Rysunek 3.1 przedstawia kod źródłowy jednego z programów, które powstały w celu odbierania pakietów od terminala mobilnego, a następnie odsyłania ich z powrotem. Został on napisany w języku programowania Python, który jest językiem skryptowym znajdującym szerokie zastosowania w serwerach, przetwarzaniu danych, a nawet sztucznej inteligencji.

W programie tym skorzystano z biblioteki o nazwie Flask (ang. *fiolka*), która w prosty sposób pozwala na realizację obsługi zapytań HTTP. Wysoki stopień abstrakcji interfejsu programisty udostępnionego przez bibliotekę Flask sprawia, że przy odrobieniu umiejętności programistycznych można implementować zarówno proste jak i bardziej skomplikowane aplikacje - serwery przy użyciu niewielkiej ilości linii kodu.

Zasada działania tego programu pełniącego funkcję serwera HTTP jest następująca: jeżeli do serwera dotrze zapytanie typu POST, odeślij przesłane dane do nadawcy w

niezmienionej postaci. W przeciwnym przypadku odeślij tekst 'Hello World!' (ang. *Witaj Świecie!*), co pozwala w prosty sposób przetestować czy serwer działa poprawnie, oraz czy dany komputer bądź terminal mobilny może się z nim połączyć.

3.2.2 Protokół TCP

```
while True:
    # Wait for a connection
    print >>sys.stderr, 'waiting for a connection'
    connection, client_address = sock.accept()
    try:
        print >>sys.stderr, 'connection from', client_address
        # Receive the data in small chunks and retransmit it
        while True:
            data = connection.recv(packet_size)
            data = data.strip('\r\n');
            if data:
                print >>sys.stderr, 'received "%s"' % data
                if 'PACKET_SIZE:' in data:
                    packet_size = int(data.split(':')[1])
                    print >>sys.stderr, 'packet size set to %s' % packet_size
                elif 'CLOSE_SOCKET' in data:
                    break
                else:
                    print >>sys.stderr, 'sending data back to the client'
                    connection.send(data)
        finally:
            # Clean up the connection
            print >>sys.stderr, 'closing the connection'
            connection.close()
```

Rysunek 3.2: Kod źródłowy serwera odbierającego i nadającego pakiety protokołu TCP

Rysunek 3.2 przedstawia fragment kodu źródłowego programu pełniącego funkcję serwera wysyłającego i odbierającego pakiety TCP. Podobnie jak poprzedni program-serwer, został on napisany w języku programowania Python, ze względu na prostotę implementacji i szeroką gamę gotowych do użycia bibliotek.

Poprzednio wspomniana biblioteka do obsługi zapytań HTTP jaką jest Flask nie posiada możliwości obsługi transmisji pakietów TCP. Ze względu na to, zdecydowano się na zastosowanie biblioteki o nazwie socket (ang. *gniazdo*). Pozwala ona na utworzenie tzw. gniazda po stronie serwera, do którego terminal mobilny może wysyłać pakiety protokołu

TCP.

Wysyłając pakiety TCP konieczne jest określenie rozmiaru pakietu w bajtach. Program oczekuje, że wśród przesyłanych danych pojawi się sekwencja `PACKET_SIZE:N` (ang. *Rozmiar pakietu*), gdzie N jest rozmiarem pakietu danych. Następnie program (tak długo, jak wysyłane są dane) odbiera pakiety o zadanej długości i odsyła je z powrotem do terminala, który je nadał. W momencie, w którym terminal mobilny przestaje wysyłać dane, serwer pozostaje w czuwaniu i czeka na kolejne pakiety do transmisji.

Dodatkowo serwer obsługuje sekwencję `CLOSE_SOCKET` (ang. *Zamknij gniazdo*), która powoduje zamknięcie gniazda po stronie serwera. Pozwala to na zapewnienie zestawiania nowego połączenia przy każdej próbie transmisji przy pomocy protokołu TCP.

3.2.3 Protokół UDP

```
1  import socket
2  import sys
3
4  # Create a UDP socket
5  sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6
7  # Bind the socket to the port
8  port = int(sys.argv[1]) if len(sys.argv) > 1 else 8000
9  server_address = ('0.0.0.0', port)
10 print >>sys.stderr, 'starting up on %s port %s' % server_address
11 sock.bind(server_address)
12
13 while True:
14     print >>sys.stderr, '\nwaiting to receive message'
15     data, address = sock.recvfrom(65565)
16
17     print >>sys.stderr, 'received %s bytes from %s' % (len(data), address)
18     print >>sys.stderr, data
19
20     if data:
21         sent = sock.sendto(data, address)
22         print >>sys.stderr, 'sent %s bytes back to %s' % (sent, address)
```

Rysunek 3.3: Kod źródłowy serwera odbierającego i nadającego datagramy UDP

Rysunek 3.3 przedstawia kod źródłowy aplikacji *server_tcp.py* będącej serwerem zaimplementowanym w języku Python pozwalającym na odbiór i retransmisję datagramów protokołu UDP. Podobnie jak w przypadku poprzednio opisane serwera obsługującego pakiety TCP zdecydowano się na skorzystanie z dostępnej w języku Python biblioteki *socket* do przeprowadzenia transmisji.

Użytkownik uruchamiając serwer ma możliwość zdefiniowania pod jakim portem zostanie on uruchomiony, w przypadku braku zdefiniowania portu domyślnym portem jest port 8000. Następnie tworzona jest nieskończona pętla, która oczekuje na wiadomości przesyłane przez aplikację mobilną a następnie retransmituje je używając tego gniazda z którego nadano pierwotne dane. Serwer umożliwia odebranie w jednym datagramie UDP 65565 bajtów danych, ponieważ taki jest maksymalny rozmiar datagramu protokołu UDP.

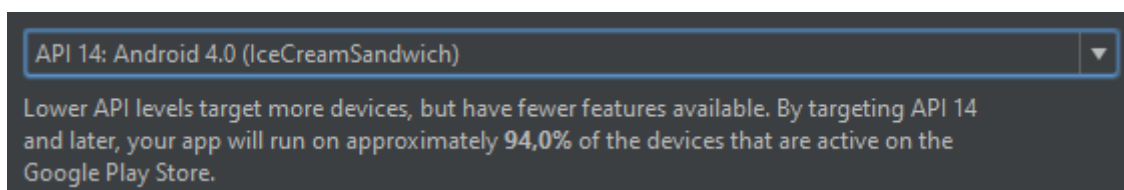
Zarówno program pełniący funkcję serwera dla zapytań HTTP jak i serwery transmitujący pakiety TCP lub datagramy UDP można uruchomić na każdym komputerze z systemem Windows, Linux lub OSX na którym zainstalowany został język programowania Python i biblioteka Flask [16].

3.3 Opis aplikacji na smartfon z systemem Android



Rysunek 3.4: Główny ekran aplikacji

Aplikacja na smartfon z systemem Android została przygotowana w środowisku Android Studio w wersji 1.3.1 i zaimplementowana w języku programowania Java. Aplikacja została napisana wspierając wersję systemu operacyjnego Android począwszy od 4.0.0, co w praktyce oznacza, że ponad 94% telefonów z systemem Android jest w stanie ją poprawnie uruchomić. Dane te potwierdza rysunek 3.5.



Rysunek 3.5: Wsparcie sprzętowe aplikacji na system Android

Rysunek 3.4 przedstawia główny ekran aplikacji napisanej na system Android. Aplikacja ta została nazwana *Android Ping!*, co podkreśla fakt, że jej działanie zbliżone jest do programu *ping* z systemów Unixowych. Przed rozpoczęciem pomiaru, aplikacja prosi użytkownika o podanie następujących danych:

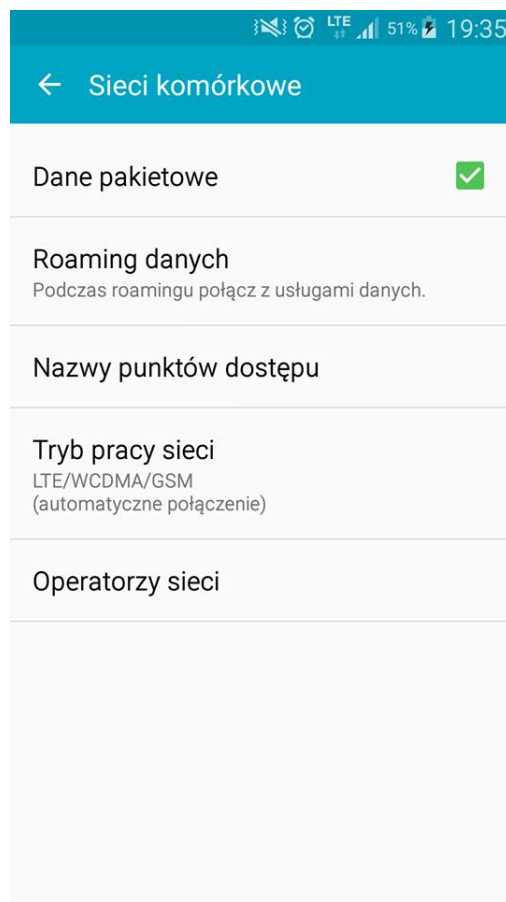
1. Adres IP serwera
2. Port, na którym nasłuchuje uruchomiony uprzednio serwer
3. Liczbę pakietów, która zostanie przesłana w trakcie pomiaru
4. Rozmiar (w bajtach) pojedynczego przesyłanego pakietu
5. Czas pomiędzy przesyłaniem kolejnego pakietu. Minimalny dopuszczalny czas wynosi 500ms, użytkownik ma także możliwość wyboru pomiędzy podaniem wartości w sekundach, minutach oraz milisekundach.
6. Protokół transmisji - do wyboru są protokoły HTTP, TCP oraz UDP

Aplikacja weryfikuje podane przez użytkownika dane przed rozpoczęciem pomiaru. Kryteria, jakie muszą spełniać parametry pomiaru są następujące:

- Rozmiar pakietu (w bajtach) musi być większy niż 0
- Rozmiar pakietu (w bajtach) musi być mniejszy niż 65000 bajtów
- Czas pomiędzy transmisją kolejnych pakietów musi być większy niż 500ms
- Liczba nadanych pakietów w trakcie pomiaru musi być większa niż 0

Po wciśnięciu klawisza MENU telefonu użytkownikowi pokazuje się przycisk *NETWORK SETTINGS* (ang. *Ustawienia sieci*), który przenosi użytkownika do ekranu ustawień systemu operacyjnego dotyczącego ustawień sieci komórkowej. Przykładowy ekran ustawień sieci komórkowej w systemie Android (w telefonie Samsung Galaxy S4) przedstawia rysunek 3.6. Dzięki dostępowi do tego okna użytkownik może przeprowadzać pomiary następujących sieci:

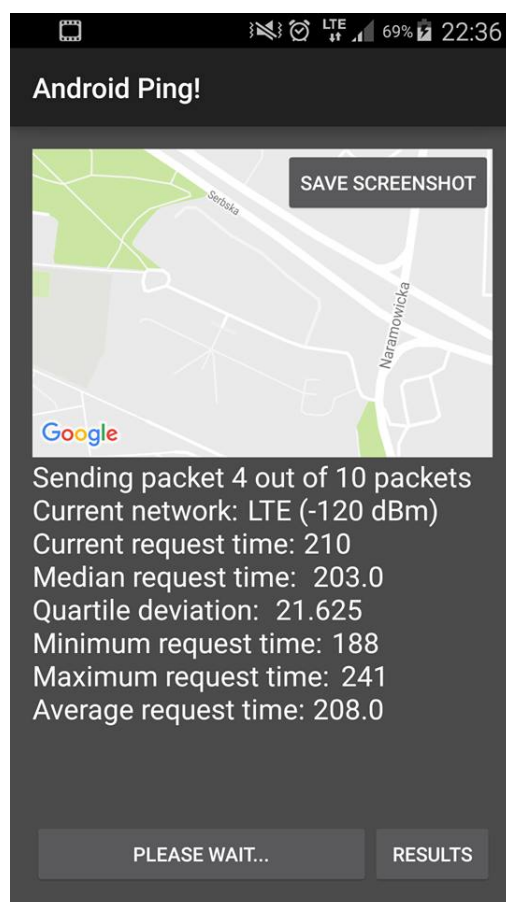
- LTE/WCDMA/GSM (automatyczne połączenie)
- WCDMA/GSM (automatyczne połączenie)
- Tylko WCDMA
- Tylko GSM



Rysunek 3.6: Ustawienia sieci komórkowej w systemie Android

Po ustawieniu parametrów pomiaru i wciśnięciu przycisku *START MEASURING* (ang. *rozpocznij pomiar*) aplikacja przechodzi do ekranu pomiaru (rysunek 3.7). W jego centralnym punkcie (zakładając, że użytkownik ma włączoną usługę nawigacji w telefonie) znajduje się mapa pochodząca od usługi Google Maps przedstawiająca aktualne położenie terminala mobilnego.

W trakcie trwania pomiaru terminal mobilny może zmienić swoją pozycję - jest ona oznaczana w trakcie kolejno dodawanych do mapy punktów. Każdy punkt oznacza pomiar dokonany w danym miejscu, a jego kolor oznacza czas transmisji danego pakietu - barwa jaśniejsza oznacza krótszy czas transmisji, barwa ciemniejsza - dłuższy. Dzięki temu rozwiązaniu, w trakcie trwania pomiaru istnieje możliwość śledzenia pozycji terminala mobilnego i wykonywania pomiarów w trakcie np. podróży samochodem lub pociągiem.

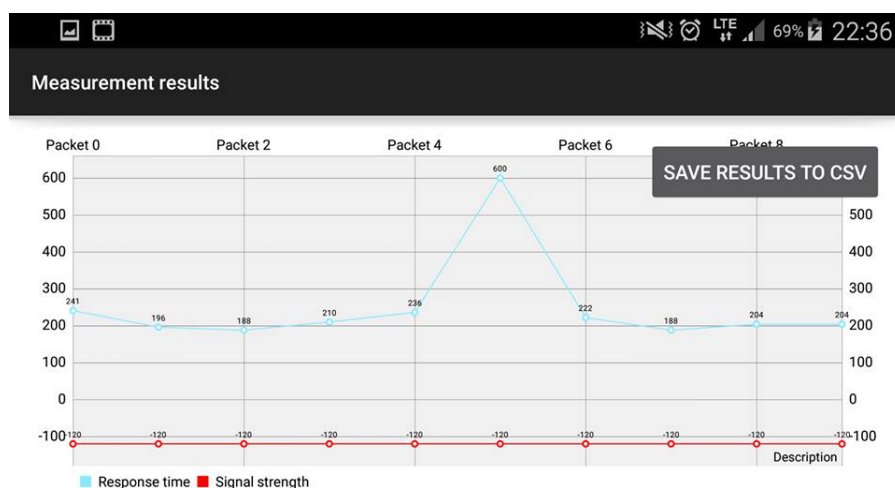


Rysunek 3.7: Ekran pomiaru

Pod mapą znajdują się informacje na temat bieżącego pomiaru takie jak:

- Ilość pakietów, które zostaną przesłane w trakcie bieżącego pomiaru
- Numer aktualnie przesyłanego pakietu
- Średni czas przesyłu pakietu dla aktualnego pomiaru

Po zakończeniu pomiaru użytkownik może zobaczyć średni czas transferu pakietu dla uprzednio ustalonych parametrów, a także użyć przycisku *RESULTS* (ang. *wyniki*), aby zobaczyć wykres obrazujący w jaki sposób zmieniały się czasy przesyłu poszczególnych pakietów. Przykład takiego wykresu obrazuje rysunek 3.8. Dodatkowo aplikacja posiada możliwość zapisu wyników pomiaru do pliku z rozszerzeniem csv (ang. *Comma Separated Values* - wartości oddzielone średnikiem). Plik tego typu może być później odczytany w programach do analizy danych takich jak Microsoft Office Excel czy Matlab. Pozwala to na dowolną analizę wyników pomiaru już po ich wykonaniu i np. zamknięciu aplikacji.



Rysunek 3.8: Ekran pomiaru

3.3.1 Opis realizacji programowej transmisji z wykorzystaniem zapytań HTTP

Rysunek 3.9 przedstawia sposób realizacji programowej transmisji z wykorzystaniem zapytań HTTP. Pokazuje on funkcję *performHttpRequests()* (ang. *wykonajZapytaniaHttp()*), która odpowiada za realizację transmisji za pomocą protokołu HTTP.

```
public void performHttpRequests() {  
    final TextView pingInfo = (TextView) findViewById(R.id.ping_info);  
    isInProgress = true;  
    HttpRequestTask httpRequestTask = new HttpRequestTask(url, packetSize, queue, pingInfo);  
    httpRequestTask.execute();  
}
```

Rysunek 3.9: Realizacja programowa transmisji z wykorzystaniem zapytań HTTP

Wewnątrz definicji funkcji *performHttpRequests()* powstaje obiekt zaimplementowanej na potrzeby aplikacji klasy *HttpRequestTask*. Jest to klasa, która dziedziczy po wbudowanej w system Android klasie *AsyncTask*. Klasa *AsyncTask* pozwala programistom aplikacji na system Android na tworzenie kodu, który wykonywany jest asynchronicznie, bez blokowania głównego wątku aplikacji. Dzięki temu, pętla, która wysyła zapytania HTTP co określony interwał nie blokuje innych funkcji aplikacji (np. śledzenia pozycji terminalu mobilnego).

```
final Response.Listener<String> successHandler = new Response.Listener<String>() {  
    @Override  
    public void onResponse(String response) throws IOException {  
        PingServerActivity.updateRequestStatistics();  
        PingServerActivity.updateCurrentResults(pingInfo);  
    }  
};  
  
final Response.ErrorListener errorHandler = new Response.ErrorListener() {  
    @Override  
    public void onErrorResponse(VolleyError error) {  
        pingInfo.setText("HTTP request error: " + error.toString());  
    }  
};
```

Rysunek 3.10: Funkcje pomocnicze odpowiadające za obsługę poprawnej lub błędnej transmisji HTTP

Wewnątrz definicji klasy *HttpRequestTask* tworzone są m.in. funkcje pomocnicze przedstawione na rysunku 3.10, które są wykonywane w przypadku poprawnego przesyłu pakietu lub błędu (odpowiednio: *successHandler* oraz *errorHandler*). W przypadku błędu wyświetlany jest odpowiedni komunikat, w przypadku poprawnej transmisji wykonywane są funkcje odpowiedzialne za aktualizację wyników i wyświetlenia aktualnego średniego

czasu transmisji na ekranie.

```
public CustomStringRequest getStringRequest() {
    CustomStringRequest stringRequest =
        new CustomStringRequest(url, successHandler, errorHandler) {
        protected Map<String, String> getParams()
        {
            String data = generator.generateRandomData(packetSize);
            params.put("data", data);
            params.put("timestamp", "" + calendar.getTimeInMillis());
            return params;
        }
    };
    stringRequest.setPriority(Request.Priority.IMMEDIATE);
    stringRequest.setShouldCache(false);
    return stringRequest;
}
```

Rysunek 3.11: Definicja funkcji *getStringRequest*

Sama transmisja odbywa się za pomocą obiektu *stringRequest*, który jest tworzone za pomocą funkcji *getStringRequest* (rysunek 3.11). Obiekt ten odpowiada za wysyłanie zapytań POST protokołu HTTP, dołączając do parametrów tego zapytania zarówno unikalny identyfikator czasowy jak i losowo wygenerowane dane o zadanym przez użytkownika rozmiarze. Każde takie zapytanie otrzymuje najwyższy możliwy priorytet (*Priority.IMMEDIATE* - ang. *natychmiast*), co gwarantuje najszybszy możliwy czas startu transmisji danego pakietu.

```
public void sendHttpRequest() {
    PingServerActivity.timeBeforeRequest = System.currentTimeMillis();
    queue.add(getStringRequest());
}

@Override
protected Void doInBackground(Void... params) {
    while (!PingServerActivity.shouldCancelNextRequest()) {
        sendHttpRequest();
        try {
            Thread.sleep(PingServerActivity.requestInterval);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    return null;
}
```

Rysunek 3.12: Pętla wysyłająca zapytania HTTP do serwera

Funkcja *doInBackground* przedstawiona na rysunku 3.12) tworzy tzw. kolejkę zapytań co oznacza, że dana transmisja za pośrednictwem zapytania HTTP będzie dodawana na kolejkę po upływie interwału określonego przez użytkownika aplikacji w trakcie ustalania parametrów pomiaru. Przykładowo, ustawienie czasu pomiędzy pakietami na 5 sekund spowoduje dodanie nowego pakietu na kolejkę po upływie 5 sekund od poprzednio wysłanego pakietu.

3.3.2 Opis realizacji programowej transmisji pakietów poprzez protokół TCP

```

@Override
protected Void doInBackground(Void... arg0) {
    try {
        socket = new Socket(dstAddress, dstPort);
        outputStream = new PrintWriter(new BufferedWriter(
            new OutputStreamWriter(socket.getOutputStream()), true);

        while (!PingServerActivity.shouldCancelNextRequest()) {
            reopenSocketIfNeeded();
            sendData();
            receiveData();
            closeServerSocketIfNeeded();
            closeClientSocketIfNeeded();
            Thread.sleep(PingServerActivity.requestInterval);
        }

        resetPacketSize();

    } catch (UnknownHostException e) {
        e.printStackTrace();
        Log.d("aping", "UnknownHostException: " + e.toString());
    } catch (IOException e) {
        e.printStackTrace();
        Log.d("aping", "IOException: " + e.toString());
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    return null;
}

```

Rysunek 3.13: Realizacja programowa transmisji poprzez protokół TCP

Rysunek 3.13 przedstawia definicję klasy *TcpSocketRequestTask*, która jest sercem realizacji programowej transmisji pakietów poprzez protokół TCP. Klasa ta (podobnie jak *HttpRequestTask*) dziedziczy po wbudowanej w system operacyjny Android klasie *AsyncTask*. Klasa *AsyncTask* pozwala na realizację asynchronicznych zadań wykonujących się w tle bez blokowania głównego wątku aplikacji. Dzięki czemu istnieje możliwość wysyłania i odbierania pakietów TCP bez przerywania pracy głównego wątku interfejsu użytkownika.

Najważniejszą metodą klasy *SocketRequestTask* jest przedstawiona na rysunku 3.13 metoda *doInBackground* (ang. *Wykonuj w tle*), która określa jakie instrukcje mają być wykonane w osobnym wątku aplikacji bez blokowania wątku interfejsu użytkownika.

Instrukcje te rozpoczynają się od utworzenia obiektu klasy *Socket*. Klasa ta jest częścią biblioteki standardowej języka Java i pozwala na implementację gniazda TCP po stronie klienta. Utworzone w ten sposób gniazdo należy podłączyć pod strumień danych, w tym przypadku jest to kolejny strumień z biblioteki standardowej języka Java jakim jest *PrintWriter*.

Następnie w pętli wykonywane są następujące kroki:

1. Otwórz gniazdo po stronie klienta (w przypadku odstępu czasu pomiędzy kolejnymi pakietami większego niż 1 minuta, gniazdo jest zamykane po każdym przesłanym pakiecie danych).
2. Wyślij dane
3. Odbierz dane
4. Zamknij gniazdo po stronie serwera (jeżeli jest to konieczne)
5. Zamknij gniazdo po stronie klienta (jeżeli jest to konieczne)

```
protected void sendData() {
    inputString = generator.generateRandomData(packetSize);
    inputString = inputString.trim();

    // Store time before request:
    PingServerActivity.timeBeforeRequest = System.currentTimeMillis();

    if (PingServerActivity.numberOfRequests == 0) {
        // Set packet size on the server side in a first request
        outputStream.println("PACKET_SIZE:" + packetSize);
    }

    // Send pingTimesEntries:
    Log.d("aping", "Sending pingTimesEntries: " + inputString);
    outputStream.println(inputString);
}
```

Rysunek 3.14: Wysyłanie pakietów TCP po stronie klienta

Rysunek 3.14 przedstawia sposób realizacji wysyłania pakietów TCP w aplikacji. Jak wspomniano w rozdziale poświęconym aplikacji pełniącej funkcję serwera odbierającego i wysyłającego pakiety TCP na komputer PC, pierwsze dane jakich spodziewa się serwer zawierają ciąg znaków *PACKET_SIZE:N* (ang. *ROZMIAR_PAKIETU:N*), gdzie

N określa liczbę przesyłanych bajtów danych (określoną przez użytkownika w trakcie ustalania parametrów danego pomiaru).

Dane te są wysyłane asynchronicznie, a więc wpisanie do strumienia danych *PrintWriter* ciągu określającego rozmiar wysyłanych pakietów powoduje przesłanie tych danych natychmiast na serwer. Po ustaleniu rozmiaru wysyłanych pakietów, program zapisuje w pamięci aktualny czas, co pozwala zmierzyć i wyświetlić czas nadania i odbioru danego pakietu w sieci w głównym wątku programu.

```
protected void receiveData() throws IOException {
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream(packetSize);
    byte[] buffer = new byte[packetSize];
    response = "";

    int bytesRead;
    InputStream inputStream = socket.getInputStream();

    while ((bytesRead = inputStream.read(buffer)) != -1) {
        byteArrayOutputStream.write(buffer, 0, bytesRead);
        response += byteArrayOutputStream.toString("UTF-8");

        if (response.length() == inputStream.length()) {
            break;
        }
    }
    publishProgress();
    Log.d("aping", "Received pingTimesEntries: " + response);
}
```

Rysunek 3.15: Odbieranie pakietów TCP po stronie klienta

Po wysłaniu danych tworzony jest bufor odbiorczy klasy *InputStream* (ang. *Strumień Wejściowy*), który odbiera od serwera dokładnie ten sam ciąg danych, który został wysłany poprzednio (rysunek 3.15). W programie została zaimplementowana także obsługa błędów transmisji - w przypadku wystąpienia takiego błędu, użytkownik zostanie o nim poinformowany stosownym komunikatem.

3.3.3 Opis realizacji programowej transmisji pakietów poprzez protokół UDP

Realizacja programowa transmisji pakietów poprzez protokół UDP jest zbliżona do TCP. Odpowiada za nią klasa *UdpSocketRequestTask*, która podobnie jak opisane po-

przednio *HttpSocketRequestTask* oraz *TcpSocketRequestTask* dziedziczy po klasie *AsyncTask*, co pozwala na asynchroniczne przesyłanie danych do serwera oraz ich odbiór.

```
@Override
protected Void doInBackground(Void... params) {
    try
    {
        while (!PingServerActivity.shouldCancelNextRequest()) {
            sendData();
            receiveData();
            Thread.sleep(PingServerActivity.requestInterval);
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    finally
    {
        if (datagramSocket != null)
        {
            datagramSocket.close();
        }
    }
    return null;
}
```

Rysunek 3.16: Realizacja programowa transmisji poprzez protokół UDP

Zaimplementowana w aplikacji transmisja danych (rysunek 3.16) za pomocą protokołu UDP nie wymaga otwierania bądź zamykania gniazda po stronie serwera i składa się z następujących kroków:

1. Przed rozpoczęciem transmisji otwórz gniazdo do przesyłania datagramów UDP
2. Wyślij dane
3. Odbierz dane
4. Po zakończonej transmisji zamknij gniazdo

```
protected void sendData() throws IOException {  
    message = generator.generateRandomData(packetSize);  
    message = message.trim();  
  
    senderPacket = new DatagramPacket(  
        |         message.getBytes(), message.length(), ipAddress, dstPort);  
    datagramSocket.setBroadcast(true);  
  
    // Store time before request:  
    PingServerActivity.timeBeforeRequest = System.currentTimeMillis();  
    datagramSocket.send(senderPacket);  
}
```

Rysunek 3.17: Wysyłanie pakietu UDP po stronie klienta

Na rysunku 3.17 został przedstawiony sposób realizacji wysyłania danych za pomocą protokołu UDP w aplikacji. Na początku tworzony jest ciąg losowych danych o długości zadanej przez użytkownika (*packetSize*). Następnie tworzony jest obiekt typu *DatagramPacket*.

DatagramPacket jest to klasa będąca częścią systemu Android, która pozwala programiście na realizacji transmisji poprzez protokół UDP w aplikacji. Aby utworzyć obiekt tej klasy należy przekazać do konstruktora tej klasy: bajty przesyłanych danych, długość przesyłanych danych, adres oraz port na który dane mają być przesłane. Następnie odczytywany jest czas w celu późniejszego zmierzenia czasu transmisji i wysłanie pakietu do serwera.

```
protected void receiveData() throws IOException {  
    receiveBuffer = new byte[packetSize];  
    receiverPacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);  
  
    datagramSocket.receive(receiverPacket);  
    publishProgress();  
}
```

Rysunek 3.18: Odbieranie pakietu UDP po stronie klienta

Rysunek 3.18 pokazuje sposób odbioru uprzednio przesłanego datagramu UDP po jego retransmisji przez serwer. Polega on na utworzeniu bufora danych o długości takiej samej jak wysłany pakiet, a następnie stworzenie obiektu *DatagramPacket*, którego

zadaniem jest odebranie przesyłanego przez serwer datagramu. Po jego odebraniu odbywa się aktualizacja aktualnych wyników pomiaru.

3.3.4 Sposób pomiaru czasu przesyłania pakietu

W poprzednich rozdziałach opisano w jaki sposób aplikacja na system operacyjny Android wysyła dane do serwera, który je odbiera i wysła z powrotem do terminalu mobilnego. Zadaniem aplikacji jest także pomiar czasu przesyłania pakietu przez sieć oraz obliczanie statystyk aktualnie dokonywanego pomiaru.

Rysunki 3.19 oraz 3.20 przedstawiają metodę pomiaru czasu przesyłu pakietu przez sieć komórkową. Przed każdym takim zapytaniem (za pośrednictwem dowolnego z dostępnych w aplikacji protokołów) pobierany jest aktualny czas (rysunek 3.19) systemowy i zapisywany do zmiennej *timeBeforeRequest* (ang. *czasPrzedZapytaniem*). Pozwala to na późniejsze porównanie czasu aktualnego (jest to zmienna zawierająca liczbę sekund, które upłynęły od 1 stycznia 1970 roku) z czasem zmierzonym jako czas bezpośrednio przed wysłaniem pakietu.

```
// Store time before request:  
PingServerActivity.timeBeforeRequest = System.currentTimeMillis();
```

Rysunek 3.19: Pobieranie aktualnego czasu przed wysłaniem pakietu

```
public static void updateRequestStatistics() throws IOException {  
    lastKnownDeltaTime = System.currentTimeMillis() - timeBeforeRequest;  
  
    pingTimes.add(lastKnownDeltaTime);  
    signalStrengths.add(signalStrength);  
    longitudes.add((float) currentLongitude);  
    latitudes.add((float) currentLatitude);  
  
    calculateStatistics(pingTimes);  
    appendCurrentResultsToFile();  
}
```

Rysunek 3.20: Pomiar czasu transmisji pakietu oraz pobieranie statystyk pomiaru

```
Median median = new Median();
medianRequestTime = median.evaluate(resultsDoubleArray);

sumOfRequestTimes += lastKnownDeltaTime;
numberOfRequests++;
averageRequestTime = sumOfRequestTimes / numberOfRequests;

maxRequestTime = Collections.max(results);
minRequestTime = Collections.min(results);
calculateQuartileDeviation(resultsDoubleArray);
```

Rysunek 3.21: Obliczanie statystyk aktualnego pomiaru

Po każdym poprawnym odbiorze pakietu wykonywana jest funkcja *updateRequestStatistics* przedstawiona na rysunku 3.20. Odpowiada ona za:

- Wyznaczenie aktualnego czasu transmisji
- Dodanie aktualnego czasu transmisji do tablicy zawierającej pozostałe czasy transmisji w trakcie aktualnego pomiaru
- Dodanie aktualne zmierzonej mocy sygnału do tablicy zawierającej moce sygnału zmierzone w trakcie aktualnego pomiaru
- Dodanie aktualnej pozycji terminala mobilnego do tablicy zawierającej informacje o pozycji terminala mobilnego w trakcie aktualnego pomiaru
- Wyznaczenie statystyk aktualnego pomiaru: średniej, mediany, odchylenia ćwiartkowego, maksymalnego i minimalnego czasu transmisji (rysunek 3.21)
- Dodanie poprzednio zebranych danych do pliku w pamięci urządzenia

3.3.5 Sposób pomiaru położenia terminala mobilnego

Aplikacja *Android Ping* posiada funkcjonalność pomiaru aktualnej pozycji terminala mobilnego, wyświetlania jej na mapie, oraz zapisywania pozycji do pliku. W tym celu skorzystano z usługi udostępnianej przez twórców Androida jaką jest Google Maps [15].

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Rysunek 3.22: Ustawienia pliku AndroidManifest.xml pozwalające na pomiar pozycji terminala mobilnego

Aby korzystać z usług lokalizacji konieczne jest ustawienie w pliku *AndroidManifest.xml* uprawnień aplikacji przedstawionych na rysunku 3.22. Jest to wymagane, ze względu na fakt, że w trakcie instalacji aplikacji użytkownik musi zgodzić się na mierzenie pozycji terminala mobilnego.

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="AIzaSyD61_gzeOIq_NeaQH3ie8ou2Va8EyR3hNU" />
```

Rysunek 3.23: Wygenerowany klucz do usługi nawigacji Google umieszczony w pliku AndroidManifest.xml

```
public synchronized void buildGoogleApiClient() {
    googleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .addApi(LocationServices.API)
        .build();
}
```

Rysunek 3.24: Sposób realizacji połączenia aplikacji z usługą nawigacji Google

W celu rozpoczęcia pomiaru aktualnej pozycji użytkownika niezbędne jest wygenerowanie klucza usługi nawigacji Google [15] (rysunek 3.23), dodanie go do pliku *AndroidManifest.xml* oraz połączenie się z usługą nawigacji. Połączenie to jest przedstawione na rysunku 3.24 i polega na stworzeniu nowego obiektu typu *GoogleApiClient*, dodaniu funkcji, które będą wykonywane w przypadku zmiany pozycji, błędu transmisji itd.

```
private void setUpMap() {
    LocationManager locationManager =
        (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);

    LocationListener locationListener = new LocationListener() {
        public void onLocationChanged(Location location) {
            if (isInProgress) {
                updateCamera(location.getLatitude(), location.getLongitude(), false);
            }
        }

        public void onStatusChanged(String provider, int status, Bundle extras) {}
        public void onProviderEnabled(String provider) {}
        public void onProviderDisabled(String provider) {}
    };

    String locationProvider = LocationManager.GPS_PROVIDER;
    locationManager.requestLocationUpdates(locationProvider, 0, 0, locationListener);
}
```

Rysunek 3.25: Sposób realizacji nasłuchiwanie na zmiany pozycji terminala mobilnego

```
public void updateCamera(double latitude, double longitude, boolean doNotMark) {
    currentLatitude = latitude;
    currentLongitude = longitude;
    CameraUpdate center =
        CameraUpdateFactory.newLatLngZoom(new LatLng(latitude, longitude), 16);
    googleMap.moveCamera(center);
    if (!doNotMark) {
        googleMap.addCircle(new CircleOptions()
            .center(new LatLng(latitude, longitude))
            .radius(6)
            .fillColor(getColorBasedOnDeltaTime())
            .strokeWidth(0));
    }
}
```

Rysunek 3.26: Sposób realizacji rejestracji aktualnej pozycji oraz aktualizacji mapy

Po połączeniu się z usługą nawigacji należy zdefiniować funkcje przedstawione na rysunku 3.25, które zostaną wykonane w przypadku np. zmiany pozycji terminala mobilnego. Najbardziej istotną z nich jest *onLocationChanged*, która wykonywana jest za każdym razem, gdy usługa nawigacji Google odczyta nową pozycję.

Po każdej zmianie aktualnej pozycji wykonywana jest funkcja *updateCamera* (ry-

sunek 3.26, która odpowiada za aktualizację danych o aktualnej pozycji, oraz wyświetlenie pozycji na mapie widocznej w aplikacji.

3.3.6 Sposób zapisywania wyników pomiaru do pliku

Aplikacja *Android Ping* posiada możliwość zapisywania wyników aktualnie wykonywanego pomiaru do pliku z rozszerzeniem .csv (ang. *Comma Separated Values* - wartości rozdzielone przecinkiem). Wybrano ten typ pliku ze względu na możliwość importowania go do dalszej analizy danych w takich programach jak Microsoft Excel czy MatLAB. Plik ten jest automatycznie w trakcie dokonywania pomiaru, istnieje także możliwość manualnego wygenerowania pliku na ekranie przedstawiającym wykres wyników pomiaru.

Do zapisywania wyników pomiaru służy zaimplementowana na potrzeby aplikacji klasa *ResultsSaver*, która odpowiedzialna jest za utworzenie pliku, przygotowanie danych do zapisania oraz wykonanie zapisu pliku do pamięci urządzenia.

```
public void createFilePath() {
    String baseDir =
        android.os.Environment.getExternalStorageDirectory().getAbsolutePath();
    String currentTime = SimpleDateFormat.format(calendar.getTime());
    fileName = "AndroidPingData " + currentTime + ".csv";

    filePath = baseDir + File.separator + fileName;
}

public boolean createFile(boolean shouldAppend) throws IOException {
    boolean isSuccessful = true;
    createFilePath();

    try {
        writer = new CSVWriter(new FileWriter(filePath, shouldAppend), ',');
    } catch (IOException e) {
        Log.d("aping", "Creating file failed");
        isSuccessful = false;
    }

    return isSuccessful;
}
```

Rysunek 3.27: Tworzenie pliku z rozszerzeniem .csv w pamięci urządzenia

Rysunek 3.27 przedstawia sposób tworzenia pliku .csv w systemie Android. Zaimplementowano dwie funkcje *createFilePath* oraz *createFile*. Pierwsza z nich tworzy ścieżkę do pliku pobierając ścieżkę do głównego katalogu pamięci urządzenia i dodając do niej nazwę pliku o formacie:

AndroidPingData

AKTUALNY_CZAS

.csv

Funkcja *createFile* tworzy pusty plik w pamięci urządzenia. W przypadku błędu w trakcie tworzenia pliku (np. brak wolnej pamięci wewnątrz urządzenia), dalszy zapis danych nie będzie kontynuowany.

```
void writeDataToFile() {  
    assignCurrentValues();  
    try {  
        writer.writeNext(generateStatisticsArray(), false);  
        writer.writeNext(generateFirstRow(), false);  
        writeResultsArray();  
        writer.close();  
    } catch (IOException e) {  
        Log.d("aping", "Save failed");  
    }  
}
```

Rysunek 3.28: Sposób realizacji zapisu danych do pliku .csv

Za zapis danych odpowiada funkcja *writeDataToFile* przedstawiona na rysunku 3.28. Funkcja ta jest wykonywana za każdym razem, gdy przesłany przez serwer pakiet zostanie poprawnie odebrany przez aplikację. W pierwszej kolejności pobierane są z pamięci aplikacji aktualne wartości pomiaru. Następnie tworzony jest pierwszy wiersz pliku, który zawiera informacje o aktualnie dokonywanym pomiarze:

- Protokół transmisji
- Aktualna sieć (EDGE/WCDMA/LTE)
- Rozmiar pakietu

- Liczba nadawanych pakietów
- Odstęp czasu pomiędzy pakietami
- Średni czas transferu pakietu
- Mediana czasu transferu pakietu
- Minimalny czas transferu pakietu
- Maksymalny czas transferu pakietu
- Odchylenie ćwiartkowe czasu transmisji pakietu

```
String[] generateFirstRow() {
    String[] firstRow = new String[4];
    firstRow[0] = "Response time [ms]";
    firstRow[1] = "Signal strength [dBm]";
    firstRow[2] = "Longitude: ";
    firstRow[3] = "Latitude: ";

    return firstRow;
}

void writeResultsArray() {
    String[] resultsArray;
    for(int i = 0; i < pingTimes.length; i++) {
        resultsArray = new String[4];
        resultsArray[0] = String.valueOf(pingTimes[i]);
        resultsArray[1] = String.valueOf(signalStrengths[i]);
        resultsArray[2] = String.valueOf(longitudes[i]);
        resultsArray[3] = String.valueOf(latitudes[i]);
        writer.writeNext(resultsArray);
    }
}
```

Rysunek 3.29: Realizacja dodania wyników pomiaru do pliku

Kolejno do pliku dodawana jest tabela wyników pomiaru, której sposób tworzenia zaprezentowano na rysunku 3.29. Składa się ona z następujących kolumn:

- Czas transmisji pakietu [ms]

- Moc sygnału [dBm]
- Długość geograficzna
- Szerokość geograficzna

Przykładowe wyniki pomiaru zapisane do pliku przedstawia rysunek 3.30.

A	B	C	D
Protocol: HTTP	Network type: LTE	Packet size: 0	Number of packets: 10
Response time [ms]	Signal strength [dBm]	Longitude:	Latitude:
302	-87	16.944283	52.431717
157	-87	16.944283	52.431717
171	-87	16.944283	52.431717
211	-87	16.944283	52.431717
155	-87	16.944283	52.431717
222	-87	16.944283	52.431717
185	-87	16.944283	52.431717
168	-87	16.944283	52.431717
176	-87	16.944283	52.431717
156	-87	16.944283	52.431717

Rysunek 3.30: Przykładowe wyniki pomiarów zaimportowane do programu Microsoft Excel

Rozdział 4

Przykładowe wyniki pomiarów

4.1 Pomiar nieruchomego terminala mobilnego

Poniższe wyniki przedstawiają pomiary dokonane w środowisku miejskim, terminal mobilny pozostawał nieruchomy przez cały czas pomiaru. Nadano 200 pakietów o rozmiarze 1024 bajtów, z odstępem 2s pomiędzy kolejnymi pakietami.

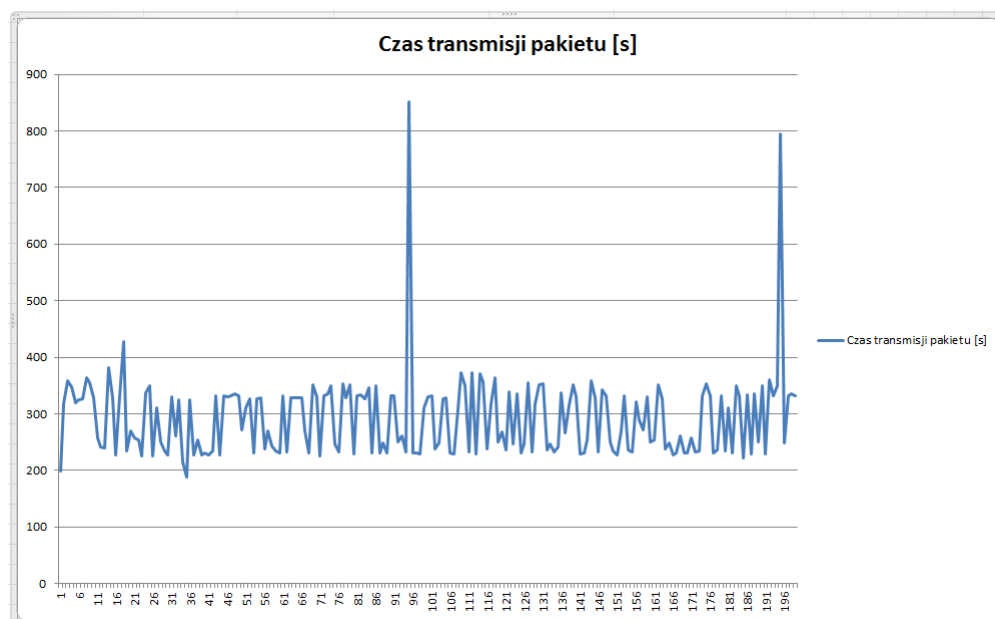
4.2 Pomiar nieruchomego terminala mobilnego (nadanie 200 pakietów, sieć LTE)



Rysunek 4.1: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć LTE, widok w aplikacji

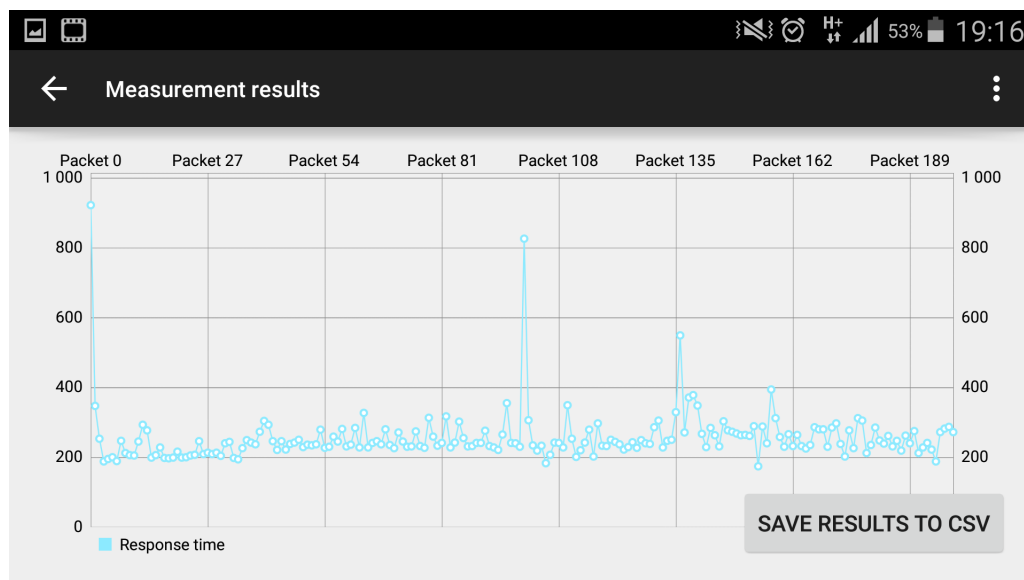
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Measurement results:	Protocol: http	Packet size: 16	Number of packets: 200	Request interval: 2	Average request time: 292	199	317	358	348	320	324	326	364	354	328	257
2																	
3																	
4																	
5																	

Rysunek 4.2: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć LTE, zapisanie wyniki pomiarów



Rysunek 4.3: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć LTE, wykres utworzony w programie Excel

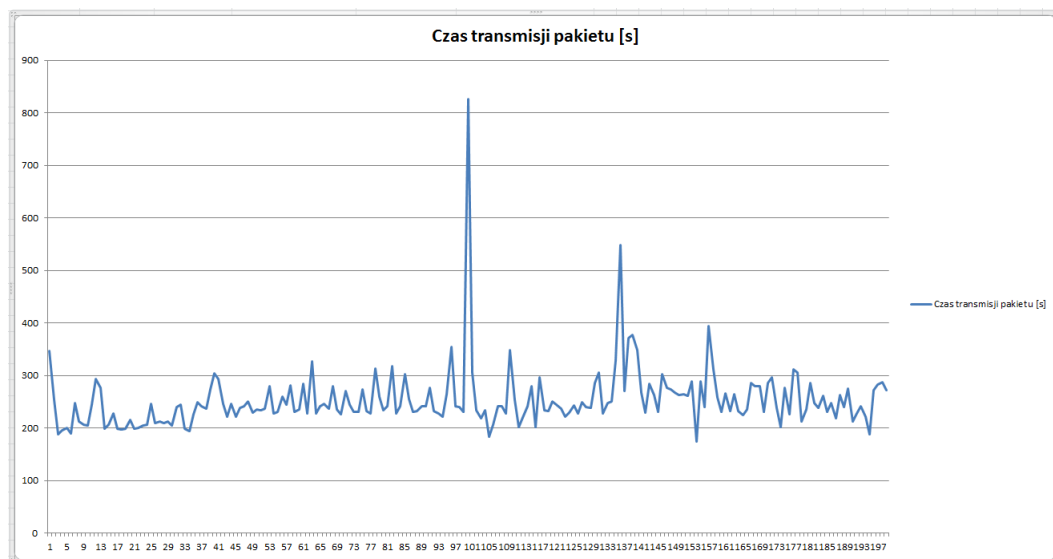
4.3 Pomiar nieruchomego terminala mobilnego (nadanie 200 pakietów, sieć WCDMA)



Rysunek 4.4: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć WCDMA, widok w aplikacji

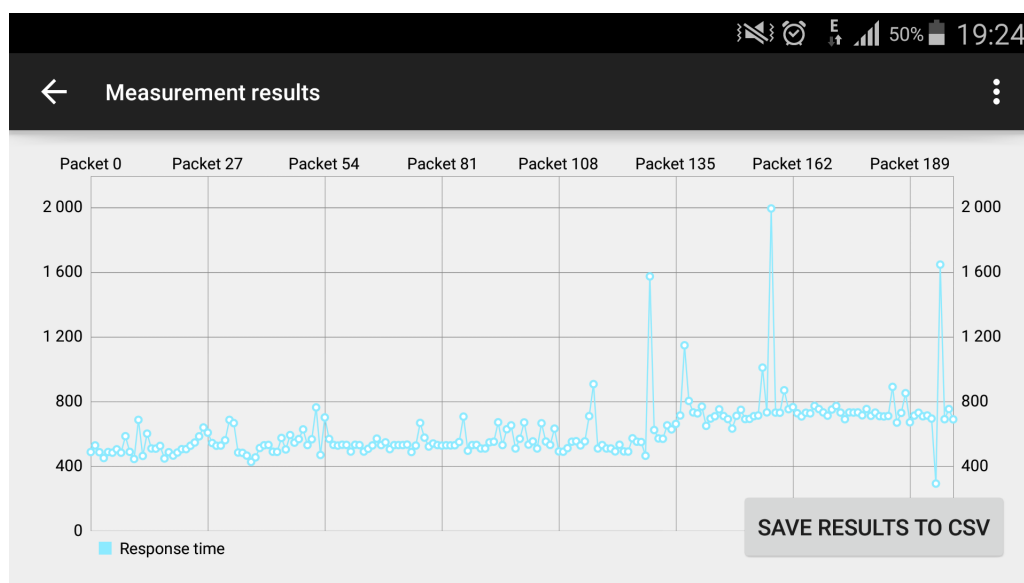
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Measurement results:	Protocol: http	Packet size: 16	Number of packets: 200	Request interval: 2	Average request time: 256	347	253	188	195	200	189	247	212	206

Rysunek 4.5: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć WCDMA, zapisanie wyniki pomiarów



Rysunek 4.6: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć WCDMA, wykres utworzony w programie Excel

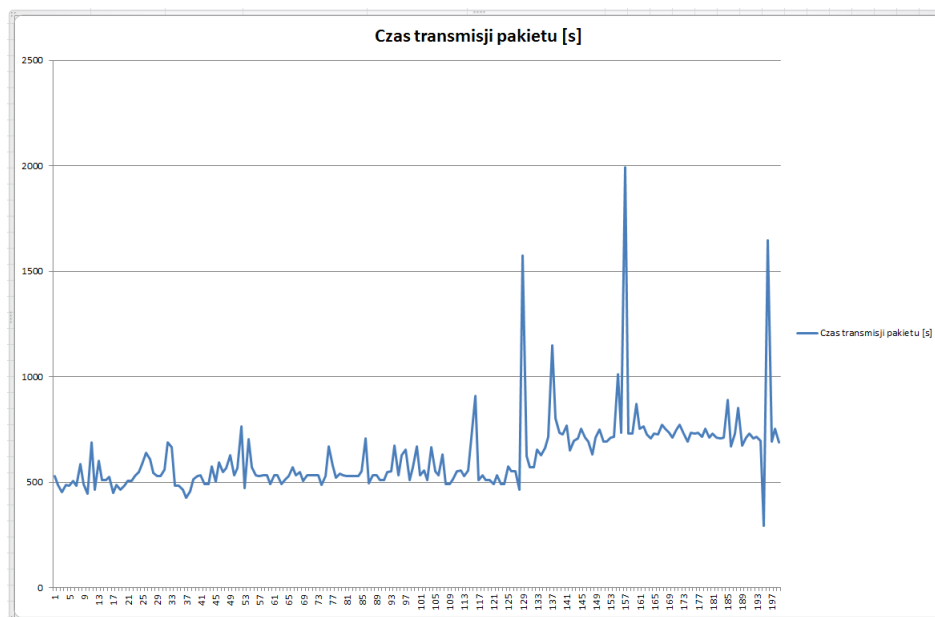
4.4 Pomiar nieruchomego terminala mobilnego (nadanie 200 pakietów, sieć EDGE)



Rysunek 4.7: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć EDGE, widok w aplikacji

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Measurement results:	Protocol: http	Packet size: 16	Number of packets: 200	Request interval: 2	Average request time: 622	529	487	451	488	484	505	484	586	489	445	68

Rysunek 4.8: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć EDGE, zapisanie wyniki pomiarów



Rysunek 4.9: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć EDGE, wykres utworzony w programie Excel

Rozdział 5

Podsumowanie

Bibliografia

- [1] *Podstawy cyfrowych systemów telekomunikacyjnych* - Krzysztof Wesołowski, WKŁ, Warszawa 2006
- [2] *Systemy radiokomunikacji ruchomej* - Krzysztof Wesołowski, WKŁ, Warszawa 2006
- [3] Źródło internetowe: *"MIMO transmission schemes for LTE and HSPA networks"*
http://www.3gamericas.org/documents/Mimo_Transmission_Schemes_for_LTE_and_HSPA_Networks_June-2009.pdf
- [4] Źródło internetowe: *Zasięg sieci trzeciej i czwartej generacji operatora Play*
<http://internet.playmobile.pl/maps/>
- [5] Źródło internetowe: *Zasięg sieci trzeciej i czwartej generacji operatora Orange*
<http://zasieg-orange.wp.pl/?ticaid=1c93f>
- [6] Źródło internetowe: *Zasięg sieci trzeciej i czwartej generacji operatora T-Mobile*
http://www.t-mobile.pl/pl/indywidualni/stali_klienci/uslugi_do_telefonu/mapa_zasiegu
- [7] Źródło internetowe: *Zasięg sieci trzeciej i czwartej generacji operatora Play*
http://www.plus.pl/mapa_zasiegu_plusa
- [8] Źródło internetowe: *Dokument RFC793 definiujący protokół TCP*
<http://tools.ietf.org/html/rfc793>
- [9] Źródło internetowe: *Android Open Source Project*
<https://source.android.com/>
- [10] *Sieci Komputerowe* - Karol Krysiak, Helion, Gliwice 2005

- [11] Źródło internetowe: *TCP, Transmission Control Protocol*
<http://networksorcery.com/enp/protocol/tcp.htm>
- [12] Źródło internetowe: *Smartphone OS market share*
<http://www.idc.com/prodserv/smartphone-os-market-share.jsp>
- [13] *Python. Leksykon kieszonkowy* - Mark Lutz, Helion, Gliwice 2014
- [14] Źródło internetowe: *Strona internetowa Python Software Foundation*
<https://www.python.org/>
- [15] Źródło internetowe: *Dokumentacja usługi Google Maps*
<https://developers.google.com/maps/documentation/android-api/>
- [16] Źródło internetowe: *Dokumentacja biblioteki języka Python Flask*
<http://flask.pocoo.org/docs/0.11/>