Praca magisterska

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

Tomasz Łakomy

Streszczenie

Abstract

Spis treści

1 Wstęp

2 Transmisja danych w sieciach komórkowych

2.1 GSM i EDGE

2.1.1 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 transmisje 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. *Enchanced Data Rules 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.

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

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

Tabela 1: Tabela porównująca standardy sieci GSM

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.

2.1.2 EDGE

Standard EDGE (ang. *Enhanced Data Rates for GSM Evolution*) powstał jako odpowiedź na zapotrzebowanie użytkowników na większe niż w przypadku GPRS prędkości transmsji danych pakietowych. Obecnie jest to najbardziej podstawowa technika przesyłana 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 se tą ponumerowane od 1 do 8 i następują cyklicznie. W przypadku transmisji mowy, kontroler stacji bazowej przypisuje terminalowi mobilnemu jedną szczelinę czasową na częstotwliwości używanej to 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 rozwiązania, 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 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.

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

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

Tabela 2: Tabela przedstawiająca schematy transmisji w sieci EDGE

2.2 UMTS i HSDPA

2.2.1 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 zmianiom (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 łączu w górę, oraz do 5.76 Mbit/s w łączu 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 trze-

ciej generacji, a polscy operatorzy komórkowi objeli jej zasięgiem niemalże cały kraj. [4][5][6][7]

Rysunek 1: Zasięg sieci 3G operatora T-Mobile w wrześniu 2015 roku

2.2.2 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 specyfikacji 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 wyobrębnienie 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.

2.2.3 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 musza 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 prędkośc 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.

2.3 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 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

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

Tabela 3: Podstawowe właściwości systemu LTE

Aby uzyskać właściwości systemu przedstawione w tabeli 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 wie-

lodostępu 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.

3 Transmisja danych w Internecie

3.1 Protokół TCP

Protokół TCP (ang. *Transmission Control Protocol*) jest połączeniowym, niezawodnym, strumieniownym protokołem stanowiącym podstawę działania dzisiejszego Internetu. Jest to protokół czwartej warstwy modelu OSI. Po raz pierwszy został on zdefiniowany w dokumencie RFC 793 z 1981 roku zatytułowanym "*Transmission Control Protocol. Darpa Internet Program protocol specification*" [8].

Protokół TCP jest częścią stosu TCP/IP, korzystającego z protokołu IP (ang. *Internet Protocol*) do przeprowadzania bezbłędnej transmisji danych w obu kierunkach.

3.2 Protokół HTTP

4 Opis realizacji programowej

4.1 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 należało 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 zapytań HTTP jak i za pomocą gniazd protokołu TCP/IP.

Należało dokonać pomiaru dla nieruchomego terminala mobilnego oraz dla terminala, którego użytkownik pozostaje w ruchu. Przy założeniu, że aplikacja posiada dwa możliwe sposoby komunikacji, oznacza to cztery możliwe sposoby przeprowadzania pomiaru:

- Pomiar terminala nieruchomego z wykorzystaniem zapytań HTTP
- Pomiar terminala nieruchomego z wykorzystaniem protkołu TCP/IP
- Pomiar terminala przemieszczającego się z wykorzystaniem zapytań HTTP
- Pomiar terminala przemieszczającego się z wykorzystaniem protkołu TCP/IP

Aplikacja posiada także możliwość wyboru pracy sieci telefonu komórkowego, wybór ten przedstawia się następująco:

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

Pojedyńczym pomiarem 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

Przygotowana aplikacja na system operacyjny Android posiada możliwość wyświetlenia wykresu prezentującego czasy transmisji zarejestrowane w czasie pomiaru. Ponadto, wszystkie pomiary są zapisywane w postaci pliku .csv w pamięci telefonu, co pozwala na ich późniejszą analizę z wykorzystaniem innych narzędzi takich jak programy Microsoft Office Excel czy Matlab.

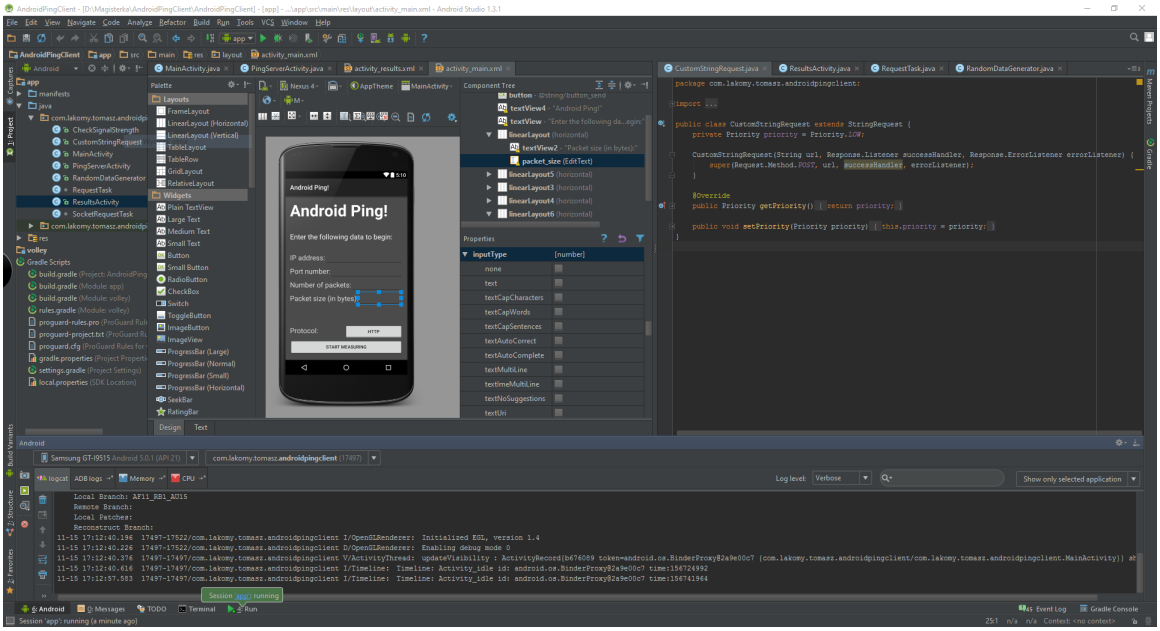
4.2 Cechy charakterystyczne programowania urządzeń z systemem Android

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 (dane na rok 2015), 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. Środowisko te 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: 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.

4.3 Opis aplikacji pełniącej funkcję serwera na komputer PC

Rysunek 3 przedstawia kod źródłowy jednego z dwóch 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

```python
import os
from flask import Flask, render_template, request

app = Flask(__name__)


@app.route('/', methods=['GET', 'POST'])
def hello():
    if request.method == 'POST':
        return request.args.get('data', '')
    else:
        return 'Hello World!'

app.debug = True
app.run(host='0.0.0.0', port=int(os.environ['PORT']))
```

Rysunek 3: Kod źródłowy serwera odbierającego i nadającego przy pomocy zapytań HTTP

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 realizacje obsługi zapytań HTTP. Wysoki stopień abstrakcji interfejsu programisty udostępnionego przez bibliotekę Flask sprawia, że przy odrobinie 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.

Rysunek 4 przedstawia kod źródłowy 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 wspomniania biblioteka do obsługi zapytań HTTP jaką jest Flask nie

```python
import socket
import sys

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the port
server_address = ('0.0.0.0', 8000)
print >>sys.stderr, 'starting up on %s port %s' % server_address
sock.bind(server_address)

# Listen for incoming connections
sock.listen(1)
packet_size = 16

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');
            print >>sys.stderr, 'received "%s"' % data
            if data:
                if 'PACKET_SIZE' in data:
                    packet_size = int(data.split(':')[1])
                    print >>sys.stderr, 'packet size set to %s' % packet_size
                else:
                    print >>sys.stderr, 'sending data back to the client'
                    connection.sendall(data)
            else:
                print >>sys.stderr, 'no more data from', client_address
                break

    finally:
        # Clean up the connection
        connection.close()

```

Rysunek 4: Kod źródłowy serwera odbierającego i nadającego pakiety protokołu TCP

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 koneiczne 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.

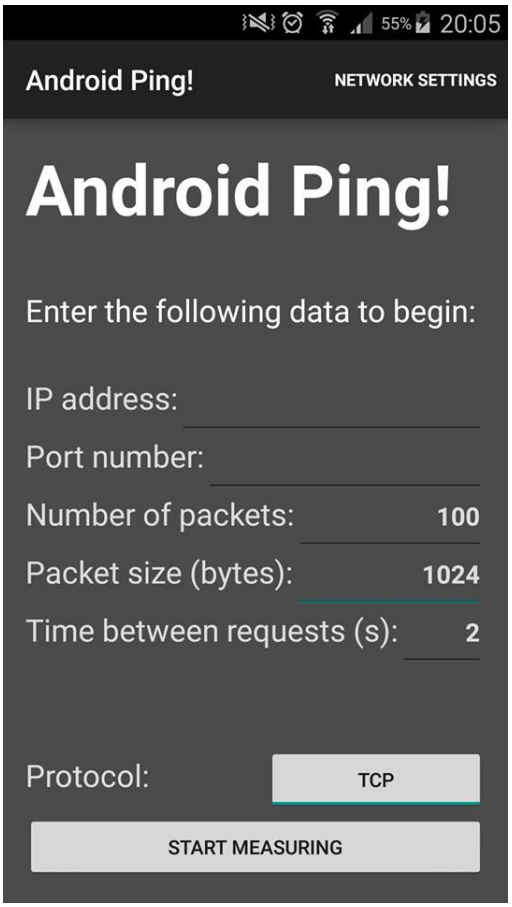
Zarówno program pełniący funkcję serwera dla zapytań HTTP jak i serwer transmitujący pakiety TCP można uruchomić na każdym komputerze z systemem Windows, Linux lub OSX na którym zainstalowany został język programowania Python i biblioteka Flask.

4.4 Opis aplikacji na smartfon z systemem Android

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 6.

Rysunek 5 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:

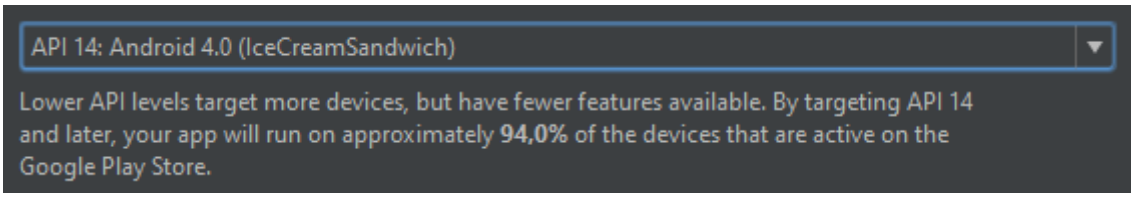
- Adresu IP serwera - jest to komputer, na którym uruchomiona jest jedna z dwóch aplikacji pełniących funkcję serwera opisanych w poprzednim rozdziale
- Numer portu



Rysunek 5: Główny ekran aplikacji

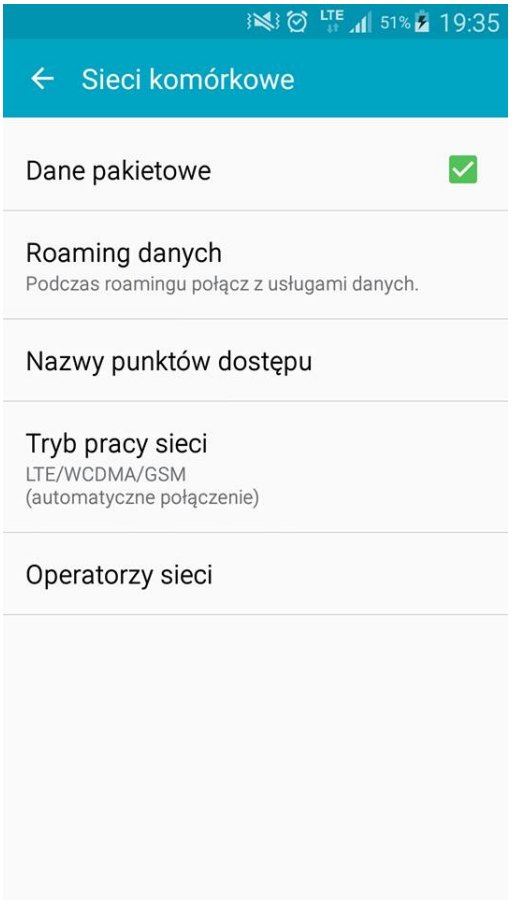
- Ilość pakietów, które zostaną przesłane w trakcie pomiaru
- Rozmiar pojedyńczego pakietu w bajtach
- Czas pomiędzy kolejnymi transmisjami pakietów (w sekundach)
- Typ protokołu, za pośrednictwem którego będzie realizowana transmisja - dostępne opcje to TCP oraz HTTP.

W prawym górnym rogu aplikacji znajduje 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 7. Dzięki dostępowi do tego okna użytkownik może przeprowadzać pomiary następujących sieci:



Rysunek 6: Wsparcie sprzętowe aplikacji na system Android

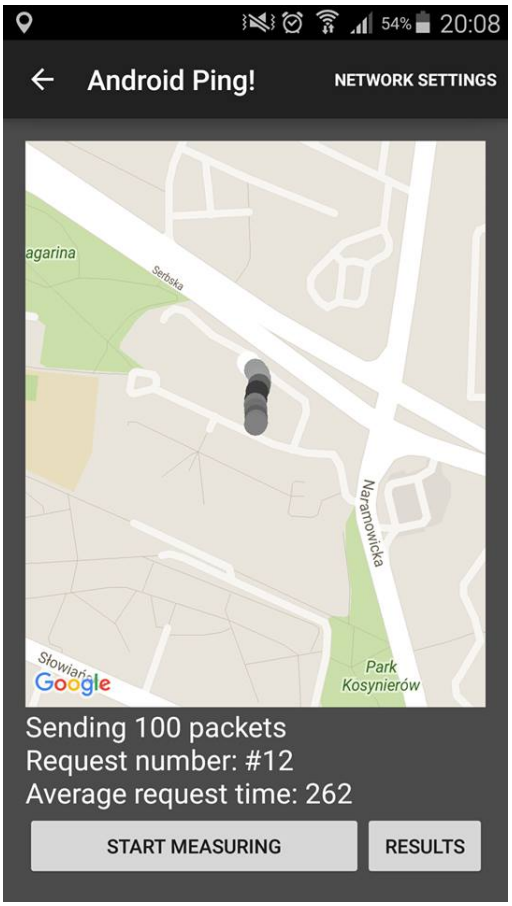
- LTE/WCDMA/GSM (automatyczne połączenie)
- WCDMA/GSM (automatyczne połączenie)
- Tylko WCDMA
- Tylko GSM



Rysunek 7: Ustawienia sieci komórkowej w systemie Android

Po ustawieniu parametrów pomiaru i wciśnięciu przycisku *START MEASURING* (ang. *rozpocznij pomiar*) aplikacja przechodzi do ekranu pomiaru (rysunek 8). W jego centralnym punkcie (zakładajac, 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 szybszy czas transmisji, barwa ciemniejsza, wolniejszy. Dzięki temu rozwiązaniu, w trakcie trwania pomiaru istnieje możliwośc śledzenia pozycji terminala mobilnego i wykonywania pomiarów w trakcie np. podróży samochodem lub pociągiem.



Rysunek 8: 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

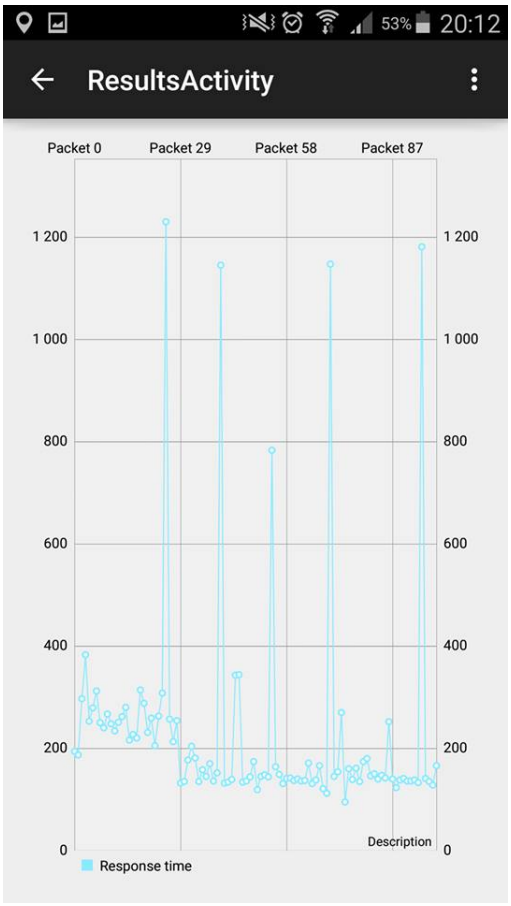
Po zakończeniu pomiaru użytkownik może zobaczyć średni czas transferu pakietu dla uprzednio ustalonych parametrów, a także użyć przycisku *RESULTS* (ang. *wyniki*), aby zobaczyć wykres obrazujący w jaki sposób zmieniały się czasy przesyłu poszczególnych pakietów. Przykład takiego wykresu obrazuje rysunek 9. 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.

4.5 Opis realizacji programowej transmisji z wykorzystaniem zapytań HTTP

Rysunek 10 przedstawia sposób realizacji programowej transmisji z wykorzystaniem zapytań HTTP. Pokazuje on funkcję *performHttpRequests()* (ang. *wykonajZapytaniaHttp()*), która odpowiada za realizacje transmisji za pomocą protokołu HTTP.

Na początku definicji tej funkcji tworzone są funkcje pomocnicze, 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 wykonwane są funkcje odpowiedzialne za aktualizacje wyników i wyświetlenia aktualnego średniego czasu transmisji na ekranie.

Sama transmisja odbywa się za pomocą obiektu *stringRequest*, który odpowiada za wysyłanie zapytań POST protokołu HTTP, dołączając do parametrów tego zapytania zarówno unikalny identyfikator czasowy jak i losowo wygenerowane dane o zadanym



Rysunek 9: Ekran pomiaru

przez użytkownika rozmiarze. Każde takie zapytanie otrzymuje najwyższy możliwy priorytet (*Priority.IMMEDIATE* - ang. *natychmiast*), co gwarantuje najszybszy możliwy czas startu transmisji danego pakietu.

Ostatnia linia tworzy tzw. kolejkę zapytań co oznacza, że dana transmisja za pośrednictwem zapytania HTTP będzie dodawana na kolejkę po upłynięciu 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.

```
183     public void performHttpRequests() {
184         final TextView mTextView = (TextView) findViewById(R.id.ping_info);
185 
186         final Response.Listener successHandler = new Response.Listener<String>() {
187 
188             @Override
189             public void onResponse(String response) {
190                 updateRequestStatistics();
191                 updateCurrentResults(mTextView);
192             }
193         };
194 
195         final Response.ErrorListener errorHandler = new Response.ErrorListener() {
196             @Override
197             public void onErrorResponse(VolleyError error) {
198                 mTextView.setText("That didn't work!" + error.toString());
199             }
200         };
201 
202         // Request a string response from the provided URL.
203         final CustomStringRequest stringRequest = new CustomStringRequest(url, successHandler, errorHandler)
204         {
205             protected Map<String, String> getParams()
206             {
207                 Map<String, String> params = new HashMap<>();
208                 RandomDataGenerator generator = new RandomDataGenerator();
209 
210                 String data = generator.generateRandomData(packetSize);
211                 params.put("data", data);
212                 params.put("timestamp", "" + Calendar.getInstance().getTimeInMillis());
213                 return params;
214             }
215         };
216 
217         stringRequest.setPriority(Request.Priority.IMMEDIATE);
218         stringRequest.setShouldCache(false);
219         RequestTask task = new RequestTask(stringRequest, queue);
220 
221         // Add the request to the RequestQueue.
222         timer.scheduleAtFixedRate(task, new Date(), requestInterval * 1000);
223     }
```

Rysunek 10: Realizacja programowa transmisji z wykorzystaniem zapytań HTTP

4.6 Opis realizacji programowej transmisji pakietów poprzez protokół TCP/IP

Rysunek 11 przedstawia definicję klasy *SocketRequestTask*, która jest sercem realizacji programowej transmisji pakietów poprzez protokół TCP/IP. Klasa ta dziedziczy po wbudowanej w system operacyjny Android klasie *AsyncTask*. Klasa *AsyncTask* pozwala na realizacje asynchronicznych zadań wykonujących się w tle bez blokowania głównego wątku aplikacji. Dzięki czemu istnieje możliwośc wysyłania i odbierania pakietów TCP bez przerywania pracy głównego wątku interfejsu użytkownika.

Najważniejszą metodą klasy *SocketRequestTask* jest przedstawiona na rysunku 11 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

```
class SocketRequestTask extends AsyncTask<Void, Void, Void> {
    String dstAddress;
    int dstPort;
    String response = "";
    int packetSize;
    TextView textView;

    SocketRequestTask(String addr, int port, int pSize, TextView txtView) {
        Log.d("aping", "SocketRequestTask: " + addr + " port:" + port);
        dstAddress = addr;
        dstPort = port;
        packetSize = pSize;
        textView = txtView;
    }

    public String getResponse() {
        return response;
    }

    @Override
    protected Void doInBackground(Void... arg0) {
        Socket socket = null;

        try {
            socket = new Socket(dstAddress, dstPort);
            RandomDataGenerator generator = new RandomDataGenerator();
            String str = generator.generateRandomData(packetSize);
            str = str.trim();
            PrintWriter out = new PrintWriter(new BufferedWriter(
                    new OutputStreamWriter(socket.getOutputStream())), true);

            // Set packet size on the server side:
            out.println("PACKET_SIZE:" + packetSize);

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

            // Send data:
            out.println(str);

            ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream(packetSize);
            byte[] buffer = new byte[packetSize];

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

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

Rysunek 11: Realizacja programowa transmisji poprzez protokół TCP/IP

tym przypadku jest to kolejny strumień z biblioteki standardowej języka Java jakim jest *PrintWriter*.

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. Następnie dane są wysyłane i tworzony jest bufor odbior-

czy klasy *InputStream* (ang. *StrumieńWejściowy*), który odbiera od serwera dokładnie ten sam ciąg danych, który został wysłany poprzednio. W programie została zaimplementowana także obsługa błędów transmisji - w przypadku wystąpienia takiego błędu, użytkownik zostanie o nim poinformowany stosownym komunikatem.

4.7 Sposób pomiaru rejestracji czasu przesyłania pakietu

W poprzednich rozdziałach opisano w jaki sposób aplikacja na system operacyjny Android wysyła dane, zarówno za pośrednictwem protokołu HTTP jak i TCP, do serwera, który je odbiera i wysła z powrotem do terminalu mobilnego.

Rysunki 12 oraz 13 przedstawiają metodę pomiaru czasu przesyłu pakietu przez sieć komórkową. Przed każdym takim zapytaniem (zarówno za pośrednictwem protokołu HTTP jak i TCP) pobierany jest aktualny czas (rysunek 12) 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.

```
public void run() {
    PingServerActivity.timeBeforeRequest = System.currentTimeMillis();
    if (type.equals("http")) {
        queue.add(request);
    } else {
        SocketRequestTask tcpRequest = new SocketRequestTask(ipAddress, portNumber, packetSize, textView);
        tcpRequest.execute();
    }
}
```

Rysunek 12: Pobieranie aktualnego czasu przed wysłaniem pakietu

Po każdym poprawnym odbiorze pakietu wykonywane są funkcje przedstawione na rysunku 13. Odpowiadają one za:

- Weryfikację numeru aktualnie odebranego pakietu (tj. czy dany pakiet jest ostatnim pakietem w danym pomiarze i należy zaprzestać wysyłania kolejnych pakietów).
- Porównanie aktualnego czasu z poprzednio zapisanym czasem sprzed wysłania pakietu oraz wyznaczenie aktualnego średniego czasu transferu pakietu przez sieć komórkową

- Aktualizację wyników wyświetlanych na ekranie - jeżeli pomiar się jeszcze nie zakończył wyświetlany jest aktualny numer pakietu, ilość wysyłanych pakietów oraz średni czas transferu. W przypadku przesłania już określonej przez użytkownika liczby pakietów wyświetlana jest informacja o zakończeniu pomiaru

```
public boolean shouldCancelNextRequest() {
    return numberOfRequests == numberOfPackets;
}

public static void updateRequestStatistics() {
    lastKnownDeltaTime = System.currentTimeMillis() - timeBeforeRequest;
    results.add(lastKnownDeltaTime);
    sumOfRequestTimes += lastKnownDeltaTime;
    numberOfRequests++;
    averageRequestTime = sumOfRequestTimes / numberOfRequests;
}

public void updateCurrentResults(TextView textView) {
    if (shouldCancelNextRequest()) {
        textView.setText("Measurement finished!\n" +
                "Average request time: " + averageRequestTime);
        timer.cancel();
    } else {
        textView.setText("Sending " + numberOfPackets + " packets"
                + "\nRequest number: #" + numberOfRequests
                + "\nAverage request time: " + averageRequestTime);
    }
}
```

Rysunek 13: Pomiar czasu przesyłania pakietu po jego odbiorze

4.8 Sposób pomiaru położenia terminala mobilnego

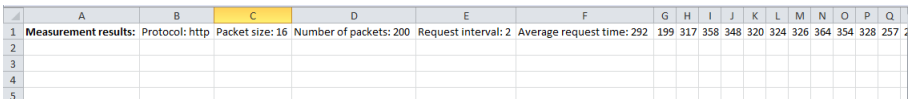
5 Wyniki pomiarów

5.1 Pomiar nieruchomego terminala mobilnego

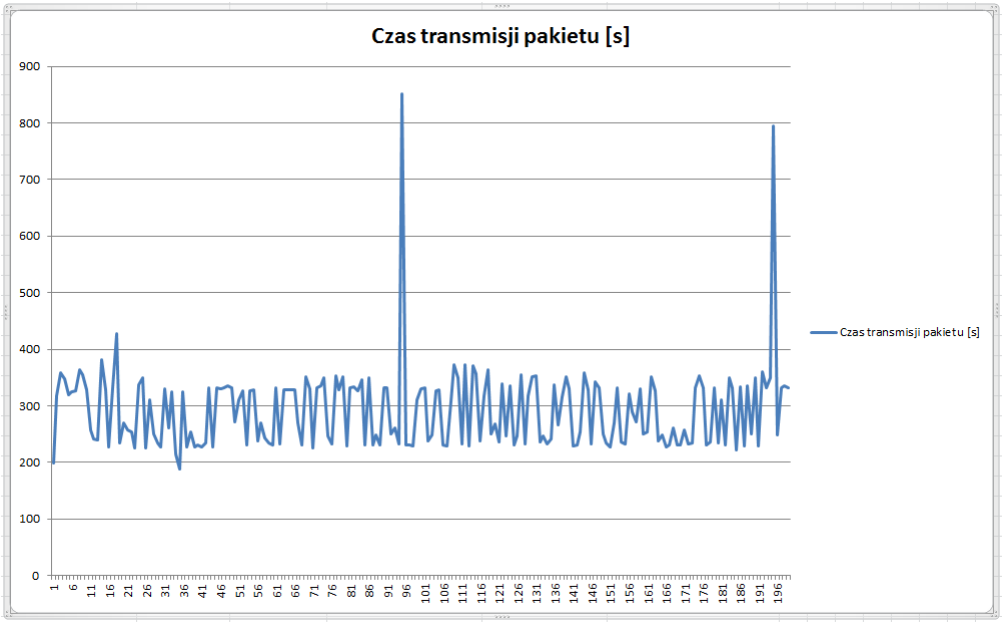
Poniższe wyniki przedstawiają pomiary dokonane w środowisku miejskim, terminal mobilny pozostawał nieruchomy przez cały czas pomiaru. Nadano 200 pakietów o rozmiarze 1024 bajtów, z odstępem 2s pomiędzy kolejnymi pakietami.

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

Rysunek 14: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć LTE, widok w aplikacji

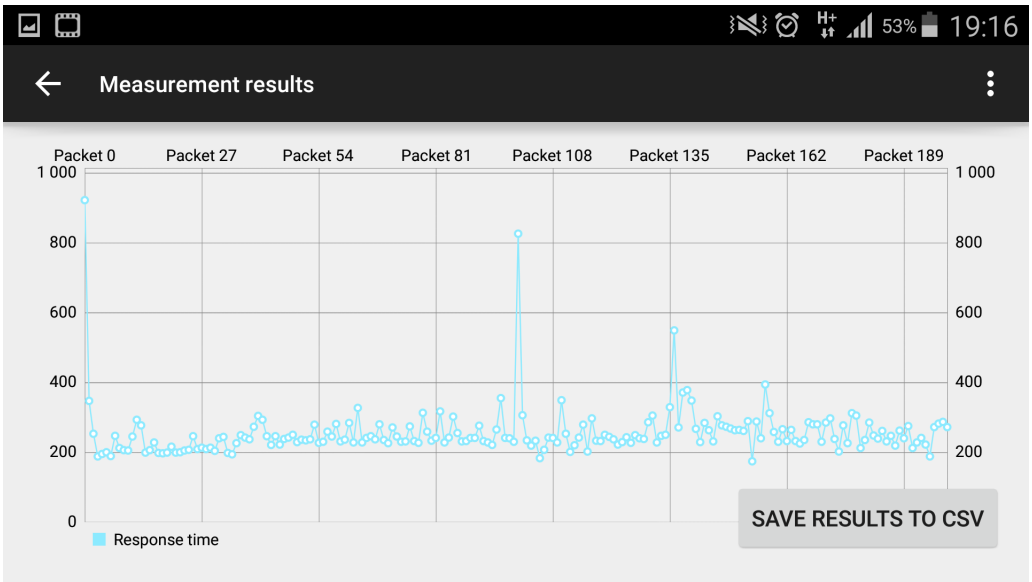


Rysunek 15: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć LTE, zapisanie wyniki pomiarów



Rysunek 16: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć LTE, wykres utworzony w programie Excel

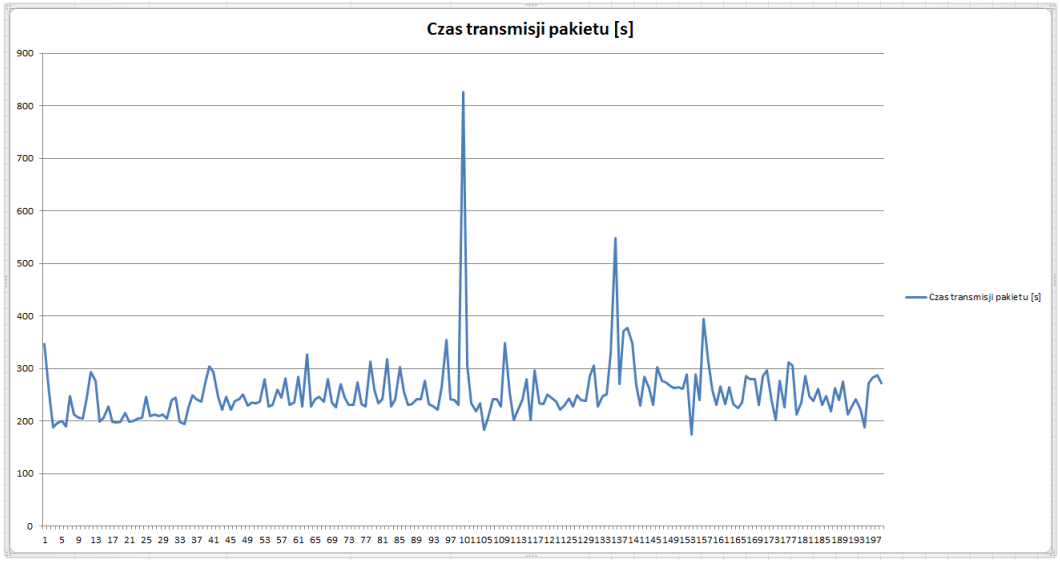
5.1.2 Pomiar nieruchomego terminala mobilnego (nadanie 200 pakietów, sieć WCDMA)



Rysunek 17: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć WCDMA, widok w aplikacji

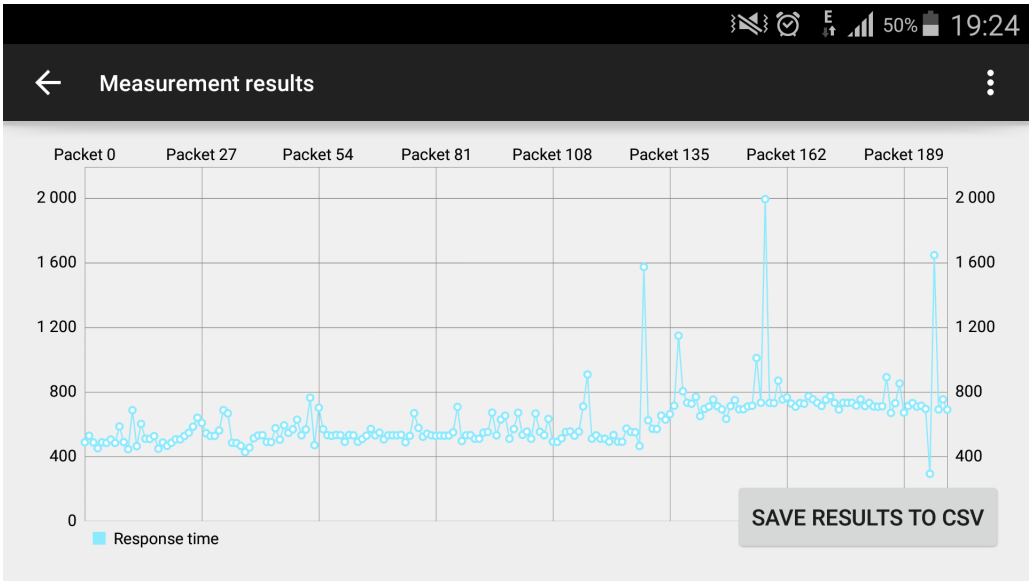
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Measurement results:	Protocol: http	Packet size: 16	Number of packets: 200	Request interval: 2	Average request time: 256	347	253	188	195	200	189	247	212	206

Rysunek 18: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć WCDMA, zapisanie wyniki pomiarów

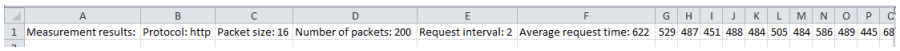


Rysunek 19: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć WCDMA, wykres utworzony w programie Excel

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

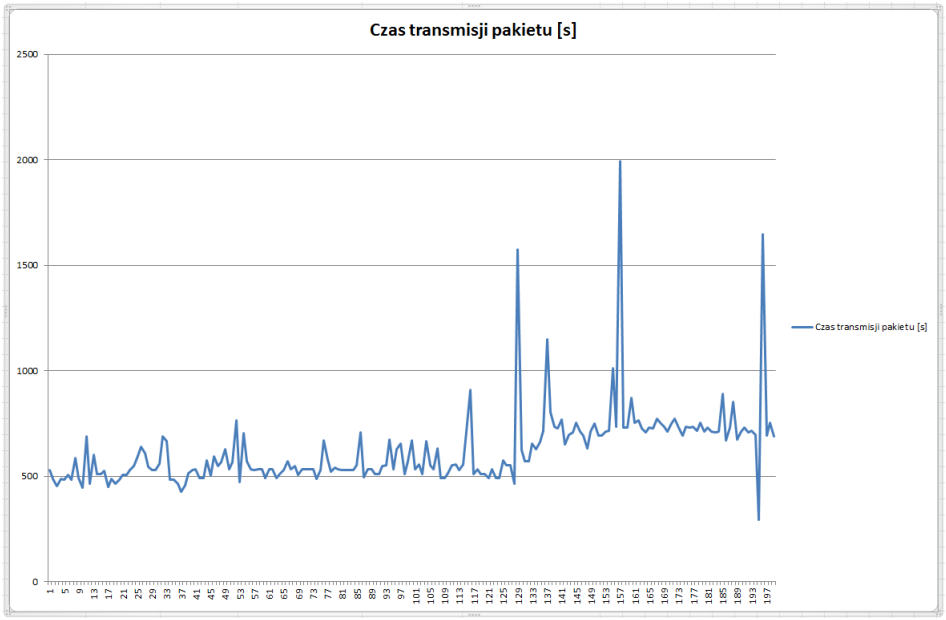


Rysunek 20: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć EDGE, widok w aplikacji



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	C
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 21: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć EDGE, zapisanie wyniki pomiarów



Rysunek 22: Wyniki pomiaru nieruchomego terminala mobilnego - nadanie 200 pakietów, sieć EDGE, wykres utworzony w programie Excel

5.2 Pomiar ruchomego terminala mobilnego w środowisku miejskim

Literatura

[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/