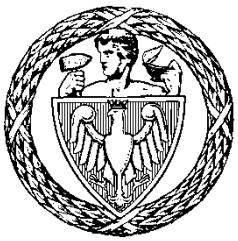
POLITECHNIKA WARSZAWSKA
Wydział Elektroniki i Technik Informacyjnych
Instytut Automatyki i Informatyki Stosowanej



PRACA DYPLOMOWA INŻYNIERSKA

Michał Pawluczuk

Prototypowa implementacja serwisu społecznościowego peer-to-peer na urządzenia mobilne

Opiekun pracy:
dr inż. Mariusz Kamola

.....
ocena pracy

.....
data i podpis Przewodniczącego
Komisji Egzaminacyjnej

Warszawa, styczeń 2013

Specjalność:

Systemy Informacyjno-Decyzyjne

Data urodzenia:

03.03.1989

Data rozpoczęcia studiów:

01.10.2008

Życiorys

Urodziłem się 03.03.1989 r. w Ciechanowie. Po ukończeniu szkoły podstawowej i gimnazjum kontynuowałem naukę w Liceum Ogólnokształcącym im. Komisji Edukacji Narodowej w Przasnyszu, gdzie uczęszczałem do klasy o profilu matematyczno-fizycznym. W październiku 2008 roku rozpocząłem studia na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej na kierunku Informatyka. W roku akademickim 2010/2011 brałem udział w kursach prowadzonych na uniwersytecie Universidad de Granada w Granadzie. W roku 2011 odbyłem trzymiesięczne praktyki w firmie AdOcean Sp. z o.o. Od stycznia 2012 r. pracuję w firmie jPALIO Business Solutions jako projektant oprogramowania.

STRESZCZENIE

Coraz większa popularność serwisów społecznościowych powoduje, że wśród ich użytkowników powstaje obawa związana z kwestiami prywatności. Na ogół, prywatne dane użytkownika, będące kluczowym elementem serwisów społecznościowych, są narażone na odczyt przez osoby trzecie, które nie są mu znane. Podobnym problemem jest kwestia kontroli wspomnianych danych wówczas, gdy są one przechowywane na urządzeniach niedostępnych fizycznie użytkownikowi. Urządzenia mobilne, również zyskują na popularności w ostatnich czasach, i otwierają rynek dla nowych aplikacji. W pracy zostanie zaprezentowana prototypowa implementacja przeznaczonego na urządzenia mobilne serwisu społecznościowego, który będzie dbał o zachowanie prywatności danych osobistych użytkownika dzięki zastosowaniu technologii peer-to-peer, ze szczególnym naciskiem na mechanizmy uwierzytelniania.

Słowa kluczowe: serwis społecznościowy, prywatność, urządzenia mobilne, aplikacja mobilna, peer-to-peer

---

Social network service for mobile using peer-to-peer technology

Rapidly growing popularity of social network services causes its users to draw attention to privacy. In many cases user private data, which is a core of social network, is accessible by third party that user does not know. Furthermore, user data is not always removable from service, especially in case when the data is stored on a device, which user cannot reach physically. Mobile devices also became popular recently, and open a new market for new mobile applications. The thesis will present a prototype mobile application of social network service, which take care of user personal data privacy using peer-to-peer technology. The thesis focuses especially on authentication.

Keywords: social networking, privacy, mobile devices, mobile application, peer-to-peer

SPIS TREŚCI

1 WSTĘP

Serwisy społecznościowe znacznie zyskały na popularności w ostatnich latach. Pozwalają ludziom w łatwy i tani sposób komunikować się ze swoimi znajomymi, również zagranicznymi. Powodem do obaw może jednak być ilość informacji osobistych jakie gromadzą. Popularnym rozwiązaniem jest przechowywanie danych o użytkownikach w bazie danych należącej do właściciela serwisu, co wymusza od nich uznania zaufanej trzeciej strony (ang. trusted third party) w celu wymiany informacji z innymi użytkownikami. Takie podejście budzi kontrowersje ponieważ prywatne dane zmienione lub usunięte przez użytkownika mogą pozostawać (choć niewidoczne dla innych) w bazie danych serwisu łatwe do odtworzenia.

W celu ochrony prywatności użytkowników można zastosować rozwiązanie niewymagające ufania nikomu poza własnymi znajomymi oraz ograniczające problem usuwania własnych danych. Opiera się ono na założeniu, że każdy użytkownik serwisu społecznościowego przechowuje swoje dane wyłącznie na swoim urządzeniu (mobilnym) i udostępnia je na żądanie autoryzowanym przez siebie pozostałym użytkownikom. Takie podejście można zrealizować przy pomocy technologii peer-to-peer.

1.1 Cel pracy

Celem pracy dyplomowej jest zbadanie technicznych możliwości wykonania serwisu społecznościowego w technologii peer-to-peer przeznaczonego dla urządzeń mobilnych oraz zaimplementowanie prototypu pełniącej tę rolę aplikacji mobilnej na platformę Android. Aplikacja ma pełnić rolę klienta sieci peer-to-peer oraz rolę serwera udostępniającego dane użytkownika, w szczególności jego dane personalne oraz multimedia. Z założenia dostęp do danych użytkownika ma być ograniczony do osób, którym właściciel tych danych dał uprawnienie. Aplikacja oprócz wymiany multimediów powinna udostępniać funkcję wysyłania i odbierania wiadomości przesyłanych między użytkownikami. Celem powstania aplikacji jest udostępnienie użytkownikowi narzędzia do komunikacji i wymiany danych ze znajomymi z maksymalnym zachowaniem prywatności.

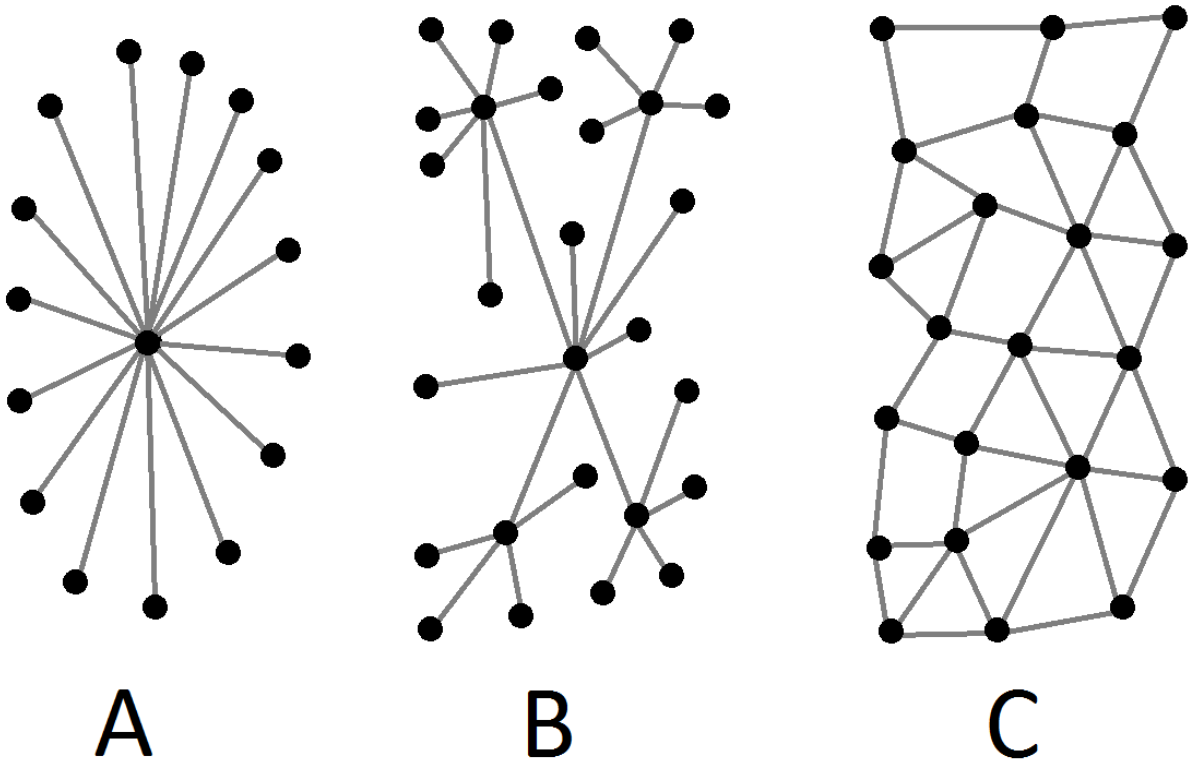
W przyjętym modelu serwisu nie ma technicznych możliwości gromadzenia danych o wszystkich użytkownikach, a co za tym idzie niemożliwy technicznie będzie ewentualny handel danymi osobowymi przez właściciela serwisu. Taka sytuacja wynika z cechy sieci peer-to-peer, którą jest rozproszenie danych. Nie istnieje punkt centralny, który mógłby gromadzić dane o wszystkich użytkownikach. Z punktu widzenia właściciela serwisu może to być cecha niepożądana, jednak zdecentralizowany model w opisanym przypadku nie wymaga

żadnych nakładów pieniężnych potrzebnych na finansowanie urządzeń pełniących role serwerów, ponieważ nie ma żadnych serwerów. Oprócz tego awaria części serwisu nie powoduje awarii jego reszty. W szczególności możliwe jest utworzenie aplikacji niewymagającej dostępu do Internetu, a do komunikacji używającej wyłącznie sieci lokalnych bez połączenia internetowego.

Co prawda istnieją już serwisy społecznościowe oparte na architekturze peer-to-peer ale żaden nie jest przeznaczony na urządzenia mobilne (Newebe http://newebe.org), jak również istnieją serwisy społecznościowe przeznaczone na urządzenia mobilne ale nie są oparte na peer-to-peer (Facebook http://facebook.com, Twitter http://twitter.com). Największym konkurentem mojego serwisu społecznościowego wydaje się być komunikator Skype (http://skype.com), ponieważ spełnia funkcje serwisu społecznościowego, używa architektury peer-to-peer, umożliwia użytkownikom komunikację z ich znajomymi oraz dzielenie się mediami (wysyłanie plików). Skype posiada implementacje na platformy mobilne: Android, iPhone, Windows Phone. To, czym dodatkowo odbiega celu mojej pracy to fakt, że korzysta z centralnego serwera w celu rejestracji i logowania użytkowników oraz zestawu dodatkowych serwerów do celu zestawienia połączenia peer-to-peer między użytkownikami. To podejście ma bezpośrednią konsekwencję w postaci przechowywanych danych użytkowników na centralnym „zaufanym” serwerze utrzymywanym przez Skype, co może budzić kontrowersje w kwestii prywatności.

2 ARCHITEKTURA PEER-TO-PEER

Peer-to-peer jest jednym z modeli komunikacji w sieci komputerowej. Zakłada on, że wszystkie węzły posiadają jednakowe uprawnienia. Peer-to-peer jest szczególnym przypadkiem architektury klient-serwer, gdzie każdy z węzłów pełni jednocześnie rolę klienta oraz serwera. Na poniższym Rysunku 2.1 ten model został oznaczony literą *C*.



Rysunek 2.1. Modele sieci komputerowych (kropki oznaczają węzły a linie oznaczają połączenia)

Literą A oznaczono scentralizowany model sieci, gdzie jest jeden serwer i wiele klientów (typowy model klient-serwer). Literą B oznaczono model zdecentralizowany, który w odróżnieniu od modelu klient-serwer A posiada kilka węzłów - komputerów podłączonych do sieci - centralnych, które mogą pełnić rolę serwerów pośredniczących w celu zapewnienia niskiego czasu odpowiedzi (np. Google datacenter, BitTorrent tracker). Z punktu widzenia modelu rozproszonego C model B posiada węzły wyróżnione, które mogą mieć większe uprawnienia od pozostałych węzłów.

W celu organizacji sieci peer-to-peer stosuje się algorytmy pozwalające na lokalizację wyszukiwanych danych. W sieciach typu C (rozproszonych) klasą tych algorytmów jest Rozproszona tablica haszująca (ang. Distributed hash table).

2.1 Rozproszona tablica haszująca

W celu umożliwienia szybkiego lokalizowania zasobów w rozproszonych sieciach peer-to-peer stosuje się Rozproszoną tablicę haszującą (ang. Distributed hash table), która w niniejszym dokumencie będzie nazywana w skrócie DHT. DHT pozwala na przeszukiwanie zasobów bez konieczności odpytywania każdego węzła.

DHT powinna udostępniać przynajmniej dwie funkcje, którymi są

- PUT – Umieszcza w DHT daną pod zdefiniowanym kluczem
- GET – Pobiera z DHT daną znajdującą się pod podanym kluczem

Ponadto pożądanymi cechami DHT są:

- Skalowalność
- Logarytmiczna (lub mniejsza) złożoność algorytmu przeszukiwania
- Zapewnianie poprawnego działania nawet w przypadku gdy nieustannie dołączają/odłączają się węzły od sieci
- Umiejętność odbudowy swojej struktury w przypadku jednoczesnej awarii kilku (kilkunastu) węzłów
- Równomiernie obciążać węzły przechowywanymi danymi, a przez to i zapytaniami

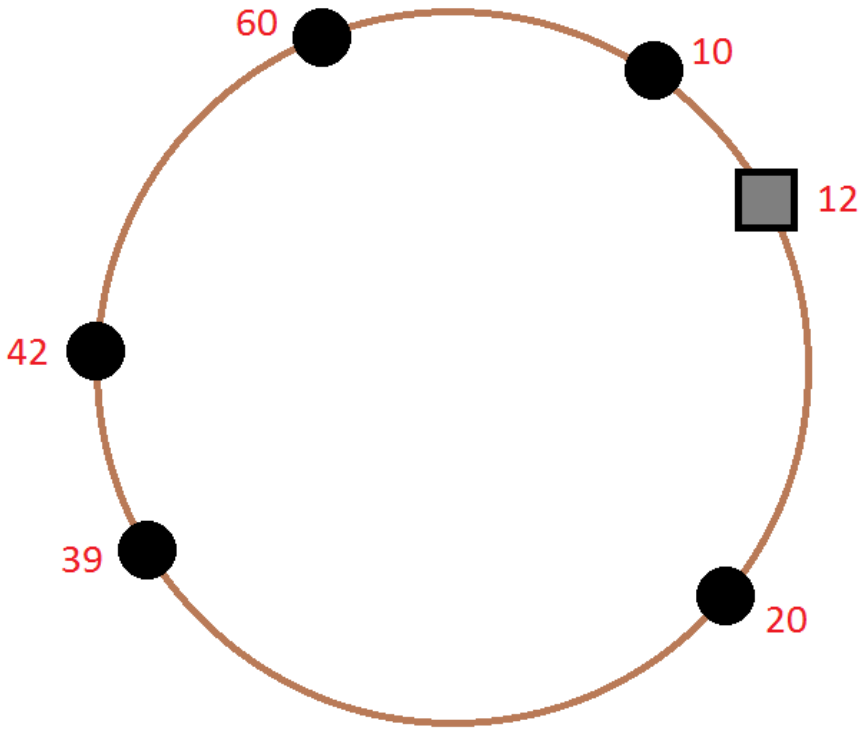
Istnieje wiele protokołów DHT np. Kademlia, Chord, Pastry, Tapestry. Do implementacji serwisu społecznościowego wybrałem Chord, ponieważ ma wszystkie pożądane cechy DHT oraz opiera się na intuicji. Dzięki temu łatwiej jest przewidzieć zachowanie węzłów oraz wykrywać błędy.

2.2 Protokół Chord

Chord [1] jest protokołem służącym do organizacji sieci peer-to-peer w celu przechowywania danych. W celu identyfikacji węzłów nadaje każdemu unikatowy numer ID z ograniczonego zakresu. Następnik jest odpowiedzią na pytanie: „Który węzeł ma najniższe ID, ale większe od X ?”, gdzie X jest argumentem pytania. Dla przykładu: jeśli węzeł *A* posiada ID=10, a węzeł *B* ID=23, oraz w sieci nie ma w chwili bieżącej żadnego innego węzła o ID z przedziału [11,22] to znaczy, że węzeł *B* jest następnikiem (ang. successor) węzła *A*.

Każda dana przeznaczona do przechowania w DHT Chord posiada klucz (indeks, podobnie jak w tablicach asocjacyjnych), na podstawie którego jest wyliczany hasz za pomocą funkcji skrótu (ang. hash function). Wyliczony hasz *H* należy do przestrzeni numerów ID węzłów. Węzłem, który będzie przechowywał tę daną jest następnik *H*. Dla przypadku pokazanego na Rysunku 2.2 wyliczony hasz wynosi 12, jego następnikiem jest

węzeł od ID=20 i to on będzie przechowywał tę daną. Funkcja skrótu powinna być zdefiniowana jednakowo u każdego węzła.

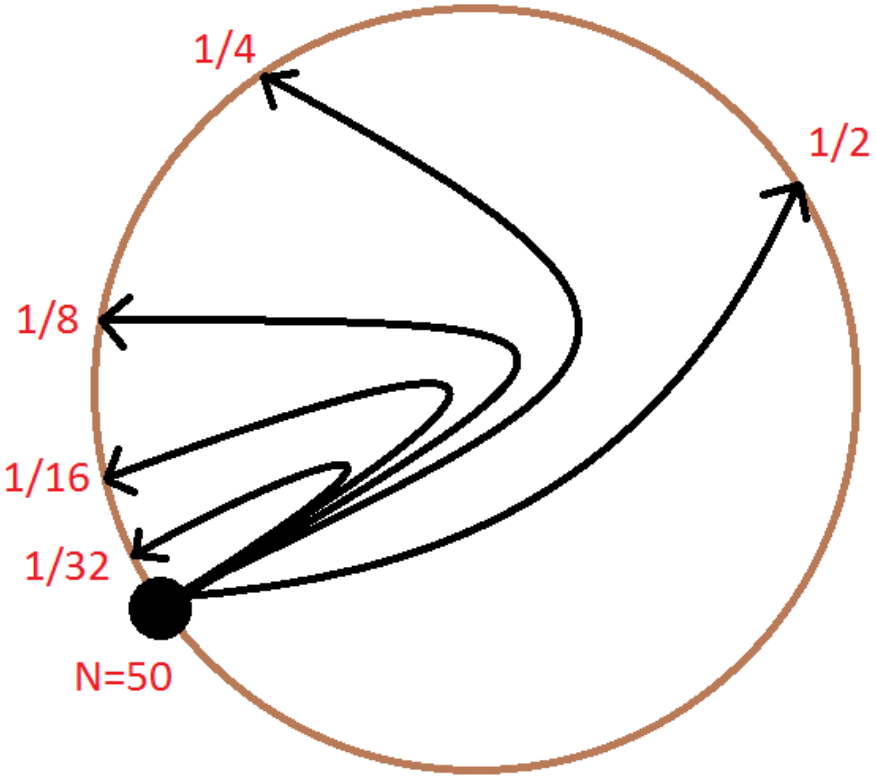


Rysunek 2.2: Schemat węzłów i danych w sieci opartej o protokół Chord

Należy zwrócić uwagę, że protokół Chord jest oparty na pierścieniu, dlatego w pytaniu o następnika trzeba uwzględnić fakt, że numerem ID następującym po najwyższym możliwym ID (maksimum z przestrzeni ID) jest numer 0. Na Rysunku 2.2 w takim przypadku następnikiem węzła od ID=60 będzie węzeł o ID=10.

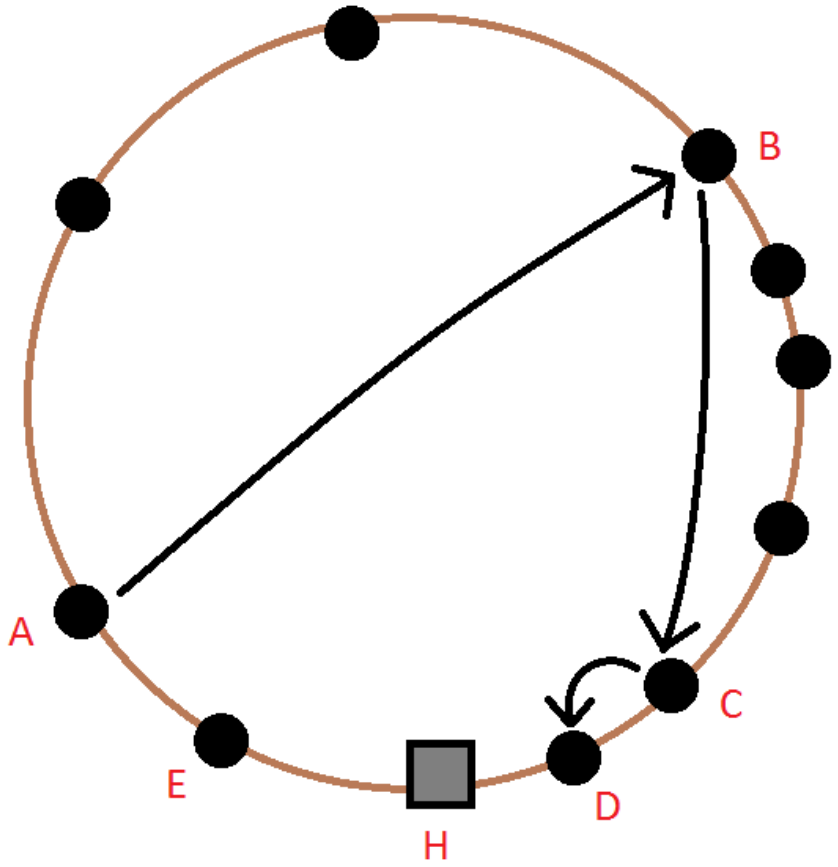
Chord definiuje również pojęcie poprzednika (ang. predecessor), który jest odpowiedzią na pytanie: „Następnikiem którego węzła jest węzeł *X* ?”, gdzie *X* jest argumentem pytania.

W celu umożliwienia komunikacji między dowolnymi węzłami każdy węzeł przechowuje zestaw linków (ang. fingers) pozwalających na bezpośrednie połączenie się z danym węzłem (np. para: adres IP oraz port TCP). Każdy węzeł posiada link do swojego następnika, następnika, oraz do *n-1* węzłów, dla $2^n=M$, gdzie *M* oznacza najwyższą wartość z przestrzeni ID. Sposób wyboru węzłów do których trzeba posiadać linki prezentuje *Rysunek 2.3*.



Rysunek 2.3: Linki węzła w sieci opartej o protokół Chord

Węzeł posiada linki które wskazują na następników ID o numerach $N+2^i$, gdzie N jest numerem węzła a i jest numerem linku. W szczególnych przypadkach (np. gdy w sieci jest tylko jeden węzeł) linki mogą wskazywać na jeden i ten sam węzeł. Taka struktura pozwala na uzyskanie logarytmicznej złożoności algorytmu do przeszukiwania DHT.



Rysunek 2.4: Poszukiwanie danej w sieci opartej o protokół Chord

Poszukiwanie danej w każdym kroku przynajmniej przepoławia drogę (droga jest rozumiana jako różnica numerów ID) do szukanej danej. Węzeł poszukujący *A* wylicza hasz danej *H*, a następnie sprawdza, czy ID danej *H* należy do przedziału [A, następnik(A)], jeśli tak to dana znajduje się u następnika (A), jeśli nie to *A* szuka w swoim zbiorze linków węzeł o największym ID mniejszym niż *H* i odpytuje się go o lokalizację danej H - ID węzła, który ją przechowuję. Algorytm następnie postępuje rekurencyjnie dla każdego kolejno odpytywanego węzła. *Rysunek 4* pokazuje schemat poszukiwań.

Dołączanie nowego węzła do sieci wymaga przekazania mu danych DHT, które powinien przechowywać (i udostępniać). Zaletą protokołu Chord jest fakt, że te dane będą pochodzić od jednego węzła i jest nim następnik nowodołączającego węzła.

Przypadek odłączania się węzłów od sieci jest nieco odmienny, ponieważ odłączenie może nastąpić wskutek zamknięcia aplikacji, ale również wskutek awarii urządzenia, sieci, gwałtownej utraty połączenia. Rozłączenie węzła powoduje utratę tych danych DHT, które przechowywał. Chociaż Chord nie definiuje zachowania węzła podczas odłączania, i prawdziwe jest stwierdzenie, że Chord nie gwarantuje utrzymywania danych DHT, a zadanie sprawdzania, czy dane DHT są dostępne jest przerzucone na wyższe warstwy aplikacji, to

jednak dobrą praktyką jest, aby węzeł świadomie opuszczający sieć przekazał przechowywane przez siebie dane sąsiadowi (następnikowi).

3 URZĄDZENIA MOBILNE

Przenośne urządzenia elektroniczne, które nie wymagają przewodowego połączenia z siecią (elektryczną, komputerową) nazywamy urządzeniami mobilnymi, są nimi np. telefon komórkowy, smartfon, tablet, notebook. Często są wyposażone w Internet mobilny, co stwarza ogromne możliwości ich użytkownikom. Przykłady urządzeń mobilnych: telefony komórkowe (Siemens S60), smartfony (iPhone, Samsung Galaxy S, HTC Wildfire), tablety (Asus Nexus), laptopy, netbooki (Eee PC), przenośne konsole do gier (Nintendo DS).

3.1 Mobilne systemy operacyjne

Mobilne systemy operacyjne zawdzięczają swoją popularność urządzeniom mobilnym, na których są instalowane przez producentów. Producenci części urządzeń (również systemów operacyjnych) umożliwiają użytkownikom tworzenie aplikacji przeznaczonych na nie. Aplikacje tworzone przez użytkowników pozwalają na ogromne zwiększenie funkcjonalności urządzenia oraz popularyzację platformy.

Popularne w ostatnich latach mobilne systemy operacyjne to: Android, iOS, BlackBerry, Symbian, Windows Phone. Wśród nich najpopularniejszym jest Android – obsługiwał na świecie blisko 52% urządzeń w czerwcu 2012 (źródło: Nielsen http://www.nielsen.com).

	Czerwiec 2012	Zakupione w okresie Kwiecień-Czerwiec 2012
Android	**51,8%**	**54,6%**
iOS	**34,3%**	**36,3%**
Blackberry	**8,1%**	**4,0%**
Pozostałe	**5,9%**	**5,0%**

Tabela 3.1: Światowy udział mobilnych systemów operacyjnych

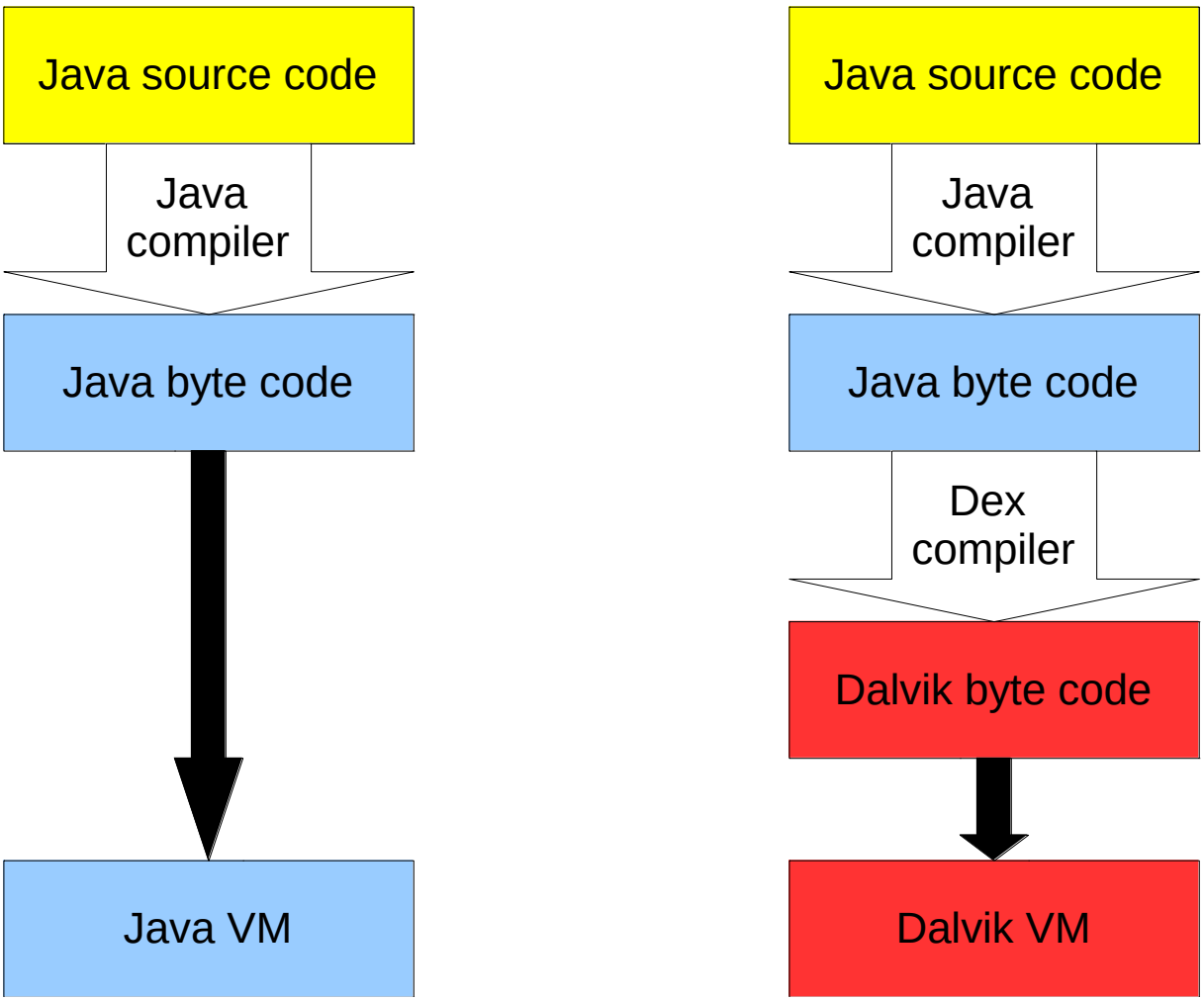
3.1.1 Android

Android (http://www.android.com) jest jednym z systemów operacyjnych przeznaczonych dla urządzeń mobilnych. Jego kod źródłowy jest otwarty i udostępniony na licencji Apache. System jest oparty na jądrze Linuxa (ang. Linux kernel). Do uruchamiania aplikacji używa maszyny wirtualnej Dalvik [2], która jest częścią systemu. Pliki wykonywalne maszyny Dalvik to *.dex, które mają swój odpowiednik plikach .class wirtualnej maszyny Javy. Pliki *.dex, w odróżnieniu od *.class, mogą przechowywać wiele klas.

Proces tworzenia aplikacji dla Androida zaczyna się od zaimplementowania aplikacji w języku Java, powstaje plik *.java, kompilator *javac* przekształca kod źródłowy do kodu bajtowego, a następnie kompilator *dx* konwertuje kod bajtowy Javy na kod bajtowy Dalvik.

Kompilator *dx* jest częścią AndroidSDK – pakietu narzędzi wspomagającego tworzenie aplikacji na tę platformę. Warto wspomnieć o narzędziu *adb*, które wspomaga komunikację komputera deweloperskiego (na którym implementujemy aplikację) z urządzeniem mobilnym będącym platformą docelową aplikacji (urządzenia mogą być połączone kablem USB). *Adb* umożliwia szybkie zainstalowanie, uruchomienie i testowanie aplikacji na rzeczywistym urządzeniu zaraz po kompilacji, oraz ułatwia debugowanie, ponieważ Android przekierowuje wyjście z konsoli aplikacji do adb, dzięki czemu programista może przeprowadzać analizę kodu na komputerze. Gdyby nie istniało *adb*, to proces testowania na urządzeniu mobilnym wydłużyłby się do przesłania aplikacji mobilnej w postaci pliku *.apk (odpowiednik pliku *.jar z Javy), instalacji aplikacji oraz przekierowania wyjścia z konsoli na jakiś inny kanał komunikacji w celu odczytania dodatkowych informacji o ewentualnych błędach wykonania.

Aplikacje dla systemu Android można implementować w językach Java, C# oraz w C++. Powyższy zestaw języków zależy od tego czy w chwili bieżącej istnieje kompilator kodu bajtowego generowanego przez kompilator danego języka na kod bajtowy maszyny Dalvik. Poniżej pokazano schemat porównujący procesy kompilacji aplikacji Java z aplikacją Dalvik.



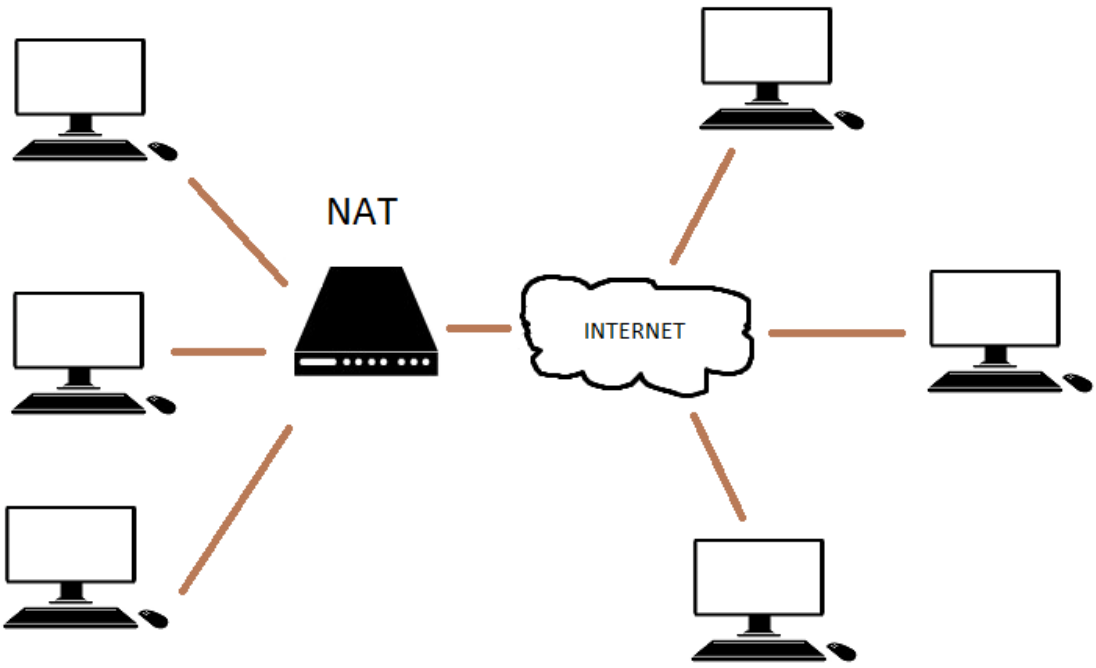
Rysunek 3.1: Proces kompilacji aplikacji dla systemu Android, źródło [3]

Wniosek ze schematu: dopóki w programie nie są wywoływane funkcje z AndroidAPI to kod źródłowy może być użyty bez żadnych konwersji w aplikacji Java.

Android pozwala na wykorzystanie w aplikacjach funkcjonalnego interfejsu użytkownika. Interfejs zawiera łatwe w użyciu komponenty takie jak EditText, Button, ImageView, ListView pozwalające znacznie rozszerzyć możliwości tworzonej aplikacji w tym zakresie.

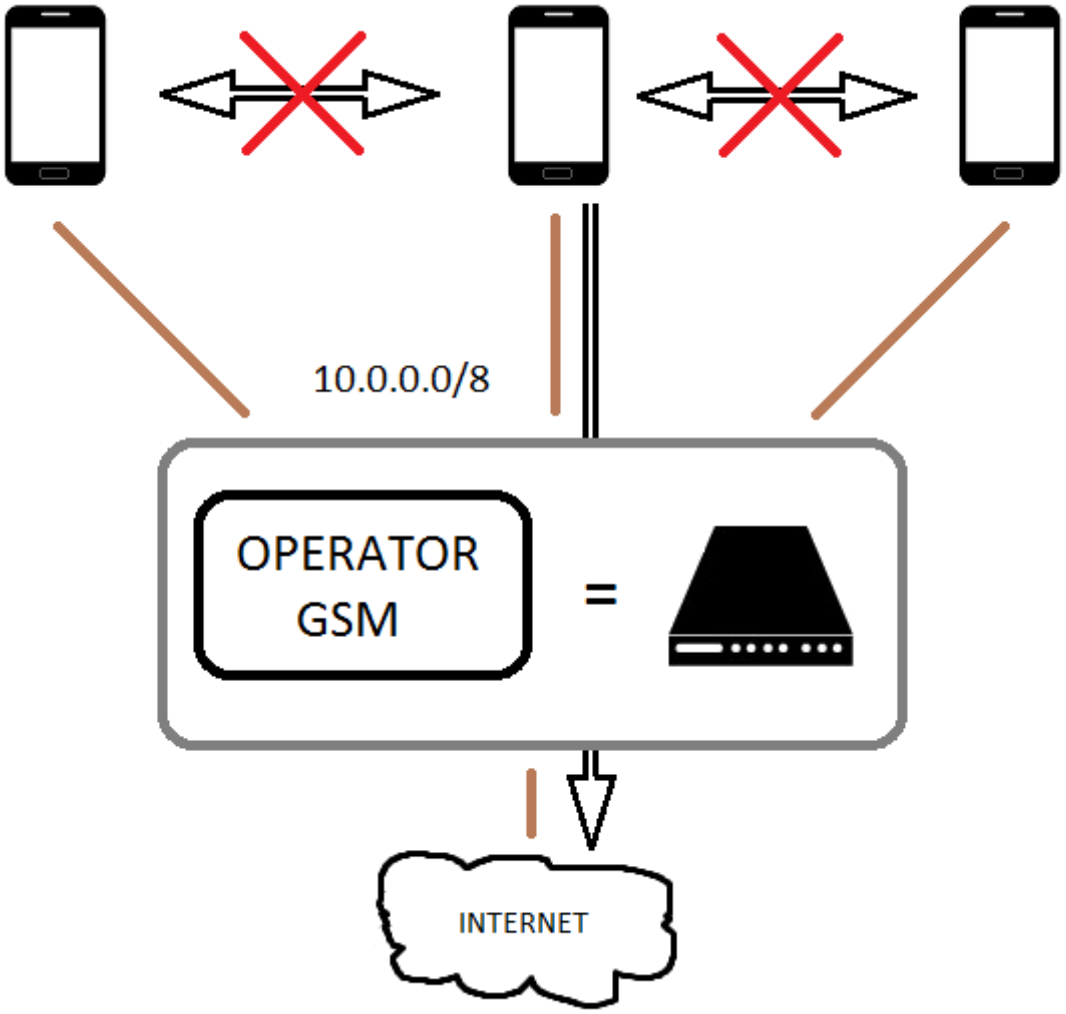
3.2 Internet mobilny

Dostawcy Internetu mobilnego w Polsce (operatorzy telefonii komórkowej) coraz rzadziej przydzielają klientom adresy IP z puli adresów publicznych, więc urządzeniu zostaje przydzielony adres IP z puli prywatnej a korzystanie z Internetu odbywa się poprzez maskaradę sieci (ang. network address translation). Wobec czego urządzenie mobilne nie jest widoczne z zewnątrz sieci i nie można nawiązać z nim połączenia (w szczególności połączenia TCP) w kierunku od Internetu do urządzenia.



Rysunek 3.2: Przykład topologii sieci z użyciem NAT

Oprócz problemu z adresami IP istnieje kolejny problem. Dostawcy Internetu mobilnego blokują połączenia przychodzące w kierunku do urządzenia mobilnego. Potwierdza to ze swojej strony dział pomocy technicznej P4 (operator sieci Play). Pozostali polscy operatorzy (Plus, T-mobile, Orange) zostali sprawdzeni doświadczalnie pod tym względem. Tę sytuację przedstawia Rysunek 3.3. Gdyby nie ten fakt to możliwe byłoby ustanowienie bezpośredniego połączenia pomiędzy urządzeniami korzystającymi z Internetu dostarczanego przez tego samego operatora GSM.



Rysunek 3.3: Blokada połączeń bezpośrednich wewnątrz sieci lokalnej operatora GSM

Powód blokowania połączeń przychodzących przez operatorów można tłumaczyć tym, że Internet mobilny jako usługa jest pod względem sumy przesłanych danych (w warstwie protokołu IP) limitowany. Aktualnie w ofertach pojawiają się limity od 100 MB do 1 GB miesięcznie. Gdyby połączenia przychodzące nie były blokowane, wówczas za taki transfer danych musiałby zapłacić (w ramach wykupionego limitu transferu) klient, a to z kolei mogłoby prowadzić do nadużyć biorąc pod uwagę fakt, że Internet jako usługa dla urządzeń stacjonarnych nie jest limitowana pod względem transferu danych.

4 KRYPTOGRAFIA

Kryptografia pozwala na ukrycie przesyłanych danych przed niepożądanym dostępem. Jest to szczególnie ważne w sieciach peer-to-peer, gdzie nie powinno mieć miejsca zaufanie do jakiegokolwiek węzła sieci, a zwłaszcza gdy zakładamy, że każdy węzeł ma równe uprawnienia. Kryptografia w komunikacji ma na celu zapewnienie poufności i spójności przesyłanych wiadomości (tekstu jawnego), oraz niezaprzeczalności. W obszarach kryptografii znajdują się omówione poniżej funkcje skrótu, klucze asymetryczne oraz klucze symetryczne.

4.1 Funkcja skrótu

Funkcje skrótu pozwalają wyliczyć krótkie i o stałym rozmiarze sygnatury (hasz, ang. hash) dla dowolnie długich danych, w celu weryfikacji spójności danych (wykrycie modyfikacji), a także w celu rozpraszania. Sygnatura dla tej samej danej będzie wyliczona dokładnie tak samo za każdym razem. Funkcje skrótu znalazły zastosowanie m. in. w protokołach sieciowych oraz rozproszonych tablicach haszujących.

Pożądane cechy funkcji skrótu:

- Odporność na kolizje – nie istnieją dwie takie same sygnatury wyliczone dla dwóch różnych danych
- Odporność na wyliczenie kolizji – dla pewnej danej A i jej sygnatury S brak praktycznej możliwości znalezienia innej danej B, która miałaby tę samą sygnaturę S.
- Jednokierunkowość – brak praktycznej możliwości odkodowania danej na podstawie jej sygnatury.
- Rozpraszanie – zmiana dowolnego jednego bitu wiadomości powinna powodować zmianę dokładnie połowę bitów sygnatury

Przykłady algorytmów do obliczania funkcji skrótu:

- CRC – Cyclic Redundancy Check
- MD5- Message-Digest Algorithm
- SHA (Secure Hash Algorithm): SHA-1, SHA-256, SHA-3
- WHIRLPOOL
- RIPEMD

Algorytm CRC nie powinien być używany do celów kryptograficznych, ponieważ nie spełnia pożądanych cech dobrej funkcji skrótu, jednak z powodzeniem jest używany do ochrony integralności danych w protokołach internetowych.

Algorytmy MD5 oraz SHA-1 obecnie już nie są kryptograficznie bezpieczne ponieważ opublikowano serię ataków na nie. Nie zmienia to faktu, że MD5 dalej nadaje się dobrze do rozpraszania danych.

4.2 Klucze asymetryczne

Kryptografia przy użyciu kluczy asymetrycznych pozwala na wykonanie szyfrowania oraz podpisu cyfrowego. W tym podejściu kryptograficznym istnieją 2 klucze – publiczny i prywatny sparowane ze sobą. Wymaganą własnością kluczy jest niemożność odtworzenia klucza prywatnego na podstawie publicznego. Klucz publiczny z reguły jest udostępniany wszystkim publicznie, lub wybranym osobom, natomiast klucz prywatny pozostaje w posiadaniu wyłącznie jego właściciela. Dobrą praktyką jest przechowywanie klucza prywatnego w postaci zaszyfrowanej.

W celu zaszyfrowania wiadomości jest ona szyfrowana kluczem publicznym odbiorcy. Jedynie osoba posiadająca klucz prywatny powiązany z publicznym, którym zaszyfrowano wiadomość, jest w stanie ją odkodować. W ten sposób skutecznie ograniczamy grono potencjalnych odbiorców do jednego wskazanego.

Drugim zastosowaniem kluczy asymetrycznych jest podpis cyfrowy. Polega on na wyliczeniu wartości funkcji skrótu dla podpisywanej wiadomości a następnie zaszyfrowanie go kluczem prywatnym. Odbiorca będzie mógł zweryfikować integralność danych oraz upewnić się, że wiadomość wysłała osoba posiadająca klucz prywatny, sparowany z publicznym. Warunkiem skutecznej weryfikacji jest uzyskanie przez odbiorcę klucza publicznego od nadawcy (można to utożsamiać z przekazaniem certyfikatu) najlepiej innym kanałem komunikacji.

4.2.1 RSA

RSA [4] jest przykładem algorytmu opartego na kryptografii asymetrycznej. W odróżnieniu od innych algorytmów tego typu (np. DSA [5] – Digital Signature Algorithm) może, przy użyciu jednego zestawu kluczy, być użyty do szyfrowania oraz podpisów cyfrowych. Algorytm opiera się na tym, że mnożenie jest łatwe a faktoryzacja trudna. Potencjalnym zagrożeniem kryptograficznego bezpieczeństwa RSA jest skonstruowanie komputera kwantowego, dla którego istnieje opracowano już algorytm faktoryzacji liczb [8] (o złożoności w czasie mniejszej niż wielomianowa).

4.3 Klucze symetryczne

Kryptografia z użyciem kluczy symetrycznym polega na użyciu tego samego klucza do zaszyfrowania jak i odszyfrowania wiadomości. W porównaniu z szyfrowaniem kluczami asymetrycznymi to podejście wymaga mniej czasu oraz obliczeń, co więcej długość zaszyfrowanej wiadomości jest na ogół taka sama jak długość wiadomości niezaszyfrowanej. Ponadto maksymalna długość wiadomości szyfrowana kluczem asymetrycznym zależy od długości klucza. Długość wiadomości szyfrowanej kluczem symetrycznym nie jest ograniczona przez algorytm.

Wadą kluczy symetrycznych jest konieczność dzielenia wspólnego klucza przez odbiorcę i nadawcę, dlatego w celu szyfrowania transmisji w praktyce dla każdej pary nadawca-odbiorca, nadawca ustala klucz. Nadawca szyfruje klucz algorytmem asymetrycznym przy użyciu klucza publicznego odbiorcy, a następnie przesyła wiadomość zaszyfrowaną kluczem symetrycznym. Odbiorca rozszyfrowuje klucz symetryczny swoim kluczem prywatnym i odbiera wiadomość zaszyfrowaną kluczem symetrycznym. W ten sposób otrzymujemy szybki i szyfrowany kanał komunikacji.

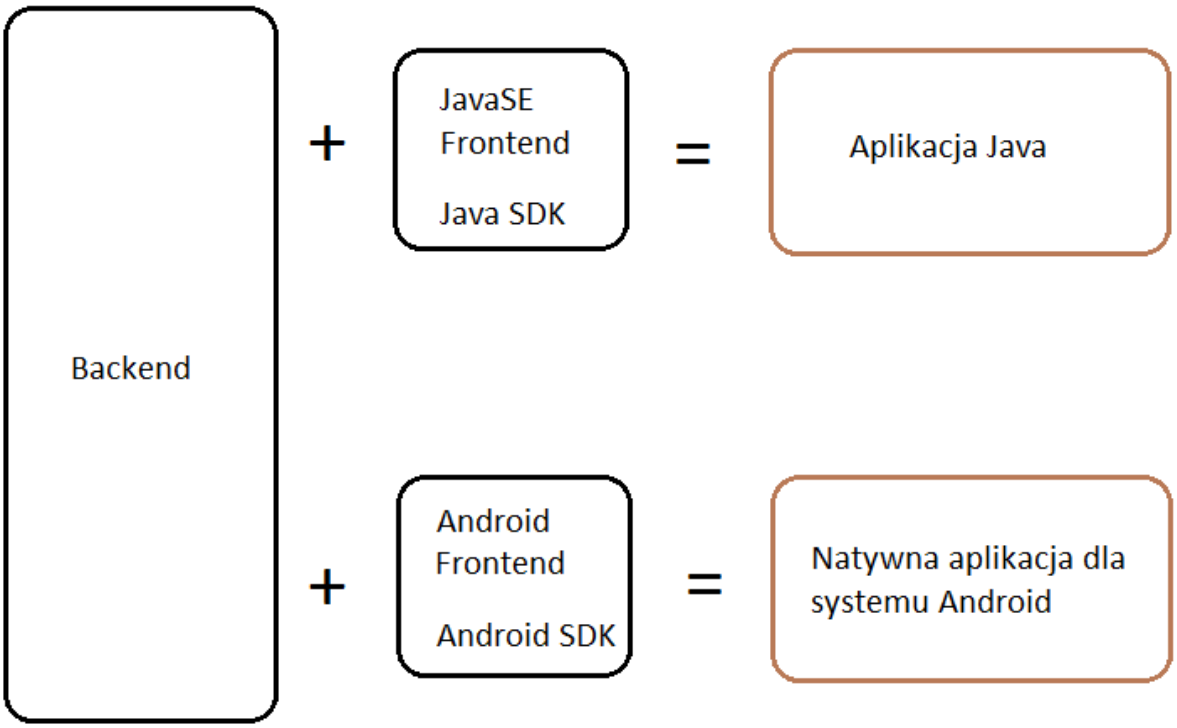
5 APLIKACJA SERWISU SPOŁECZNOŚCIOWEGO

5.1 Opis serwisu społecznościowego

W ramach serwisu społecznościowego powstała aplikacja dla platformy Android, która jest prototypem aplikacji mobilnej pełniącej funkcje serwisu społecznościowego opartego na architekturze peer-to-peer.

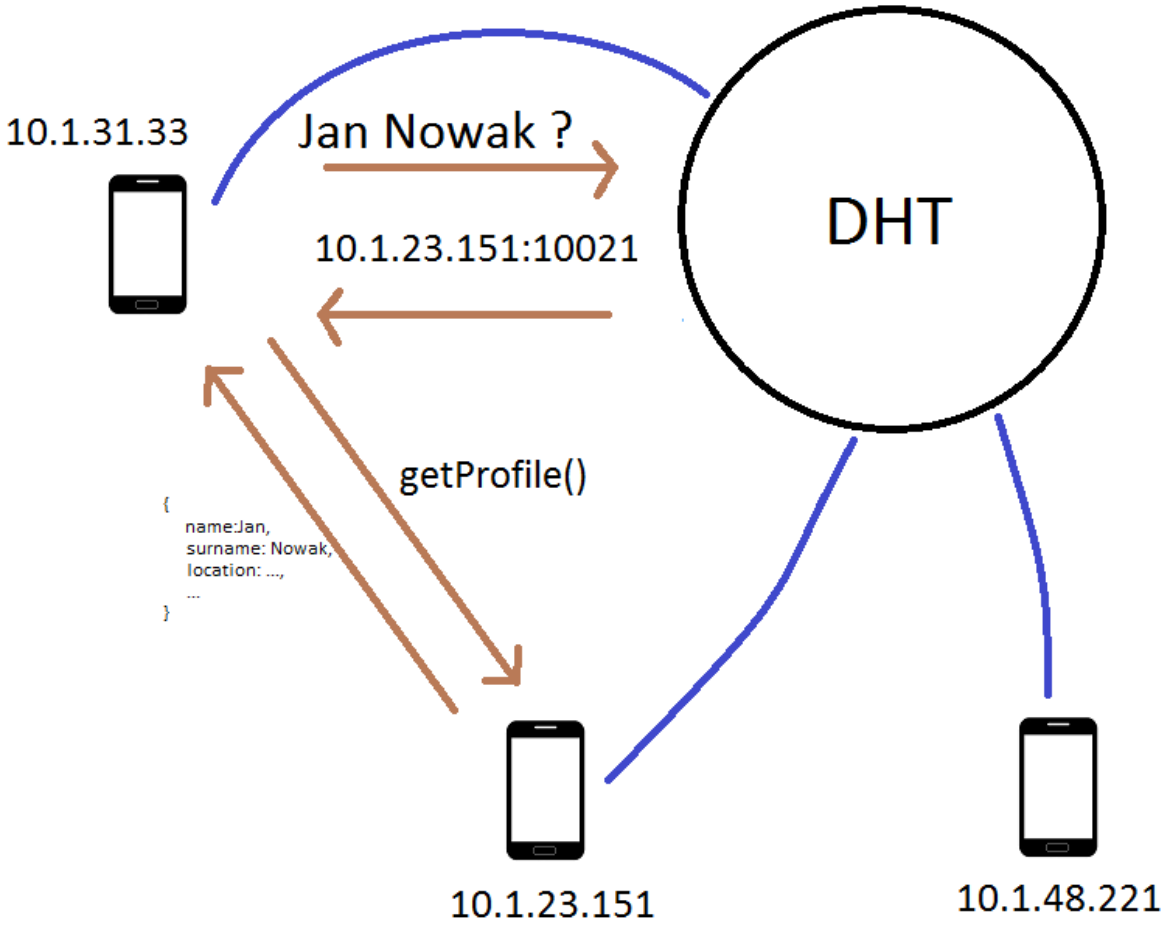
5.1.1 Historia zmian założeń

Początkowo aplikacja miała być napisana w języku Java MicroEdition (JavaME) ponieważ z tej technologii miały korzystać urządzenia mobilne. Celem było umożliwienie uruchomienia aplikacji na jak największej liczbie urządzeń. Okazało się, że Android – najpopularniejszy obecnie mobilny system operacyjny - nie wspiera natywnie aplikacji Java dla telefonów komórkowych (MIDlet), i ewentualnie można zainstalować specjalne oprogramowanie do ich uruchamiania, ale wymaga to wcześniej przeprowadzenia procesu rootowania – zdjęcia blokady uprawnień do konta root. Niestety proces rootowania na ogół prowadzi do utraty gwarancji producenta (sam proces jest jednak jest odwracalny i przywraca gwarancję). System iOS nie posiada implementacji Javy (z powodu polityki firmy), wobec czego uruchomienie na nim MIDletu jest niemożliwe. Nie chcąc tracić ogromnego rynku potencjalnych użytkowników aplikacji, wybór padł na napisanie natywnej aplikacji mobilnej dla systemu Android. Android jest obecnie najpopularniejszym systemem operacyjnym, który pozwala użytkownikom na pobieranie aplikacji z serwisu Android Market (play.google.com). Android Market jest przyjazny deweloperom aplikacji i w odróżnieniu od AppStore firmy Apple nie ma abonamentu na konto deweloperskie. Sprzedaż aplikacji przez Android Market jest również możliwa, właściciel platformy (firma Google) pobiera prowizję od sprzedaży. Android posiada jeszcze jedną zaletę: kod bajtowy dla systemu Android można uzyskać w wyniku kompilowania kodu napisanego w języku Java, wobec czego część kodu jest przenośna i stosunkowo małym nakładem pracy (implementacja samego frontendu) można będzie utworzyć wersję tej samej aplikacji w JavaSE np. na platformę PC.



Rysunek 5.1: Utworzenie dwóch aplikacji opartych na wspólnym kodzie

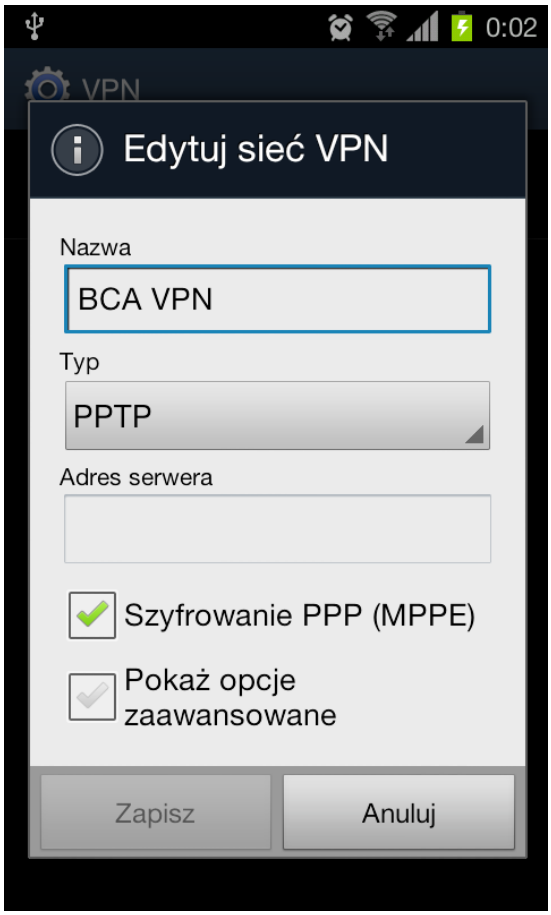
Kolejnym początkowym założeniem było użycie technologii DynDns (darmowa usługa DNS dla użytkowników ze zmiennym adresem IP), która jest dostępna [6] na platformę Android i miała rozwiązać problem zmiennych adresów IP przydzielanych urządzeniom mobilnym. Użytkownik miał przechowywać swoje dane na swoim urządzeniu i udostępniać je na żądanie, i jego znajomi znając jego adres przydzielony przez usługę DynDns będą mogli się z nim połączyć. Wadami użycia tej technologii były: krytyczny punkt centralny (serwer DynDns), inni użytkownicy DynDns (pozajmowane nazwy domenowe), konieczność instalacji na urządzeniu użytkowników dodatkowej aplikacji (aplikacja musiałaby być zainstalowana oddzielnie – ręcznie – przez użytkownika). Koncepcja użycia DHT (Rozproszonej tablicy haszującej) rozwiązała problem lokalizacji urządzenia udostępniającego dane (adres IP oraz port TCP miały być umieszczane w DHT pod sprecyzowanym kluczem w celu późniejszego odnalezienia przez osoby zainteresowane).



Rysunek 5.2: Koncepcja zastosowania rozproszonej tablicy haszującej – DHT
Linie bez strzałek oznaczają przynależność do DHT, linie ze strzałkami to zapytania i odpowiedzi.

W kwestii protokołu DHT wybór padł na Chord. Zaimplementowałem protokół Chord po swojemu, wprowadzając innowację dotyczącą generowania ID węzłów. Pojawił się kolejny problem. Operatorzy GSM w ramach usługi Internetu mobilnego przydzielali części klientów (w tym mi) adresy IP z puli prywatnej, wobec czego nie byli dostępni z zewnątrz sieci (w szczególności z Internetu). Pojawiła się koncepcja powstania mniejszej sieci serwisu społecznościowego ograniczonego do użytkowników sieci jednego operatora GSM. Jednak ten pomysł okazał się nierealizowalny, gdy pracownicy z pomocy technicznej firmy P4 (Play – operator sieci) oznajmili, że wszelkie połączenia TCP inicjujące połączenie w stronę do użytkownika Internetu mobilnego są blokowane przez operatora. Informacja została potwierdzona moimi próbami ustanowienia połączenia z użyciem Internetu mobilnego dostarczanego przez pozostałych operatorów GSM w Polsce. Koncepcja zrealizowania globalnej sieci serwisu społecznościowego peer-to-peer z użyciem Internetu mobilnego w takim przypadku okazuje się bardzo utrudniona. Rozwiązaniem może być użycie technologii wirtualnych sieci prywatnych VPN [7] (ang. Virtual Private Network), która niestety wymaga

ośrodka centralnego, którym jest serwer VPN. Warto dodać, że VPN jest natywnie wspierany przez Androida.



Rysunek 5.3: Konfiguracja VPN w systemie Android 4.0.3

VPN jednak nie jest jedyną opcją pozwalającą na sensowne korzystanie z aplikacji. Kolejnym sposobem jest tworzenie małych sieci serwisu społecznościowego z użyciem routera posiadającego funkcję WiFi. Znajomi, będąc połączeni z tym samym routerem korzystając z adresów IP z zakresu prywatnego mogą stworzyć taką sieć i wymieniać się mediami. Transmisja jest dużo szybsza, wadą jest niewielki zasięg (ograniczony mocą anteny WiFi).

5.1.2 Założenia

Głównym celem stworzenia serwisu społecznościowego była potrzeba ochrony danych osobistych oraz zdjęć i plików video użytkowników. Przypuszczalnie pierwszymi użytkownikami tego serwisu będą osoby, które obawiają się powierzać swoje dane trzecim stronom.

Architektura peer-to-peer pozwala na rozproszenie przechowywanych danych, w szczególności możliwe do zastosowania będzie rozwiązanie, w którym każdy użytkownik

przechowuje wyłącznie swoje dane i udostępnia je na żądanie. Wadą natomiast (z punktu widzenia serwisu) jest to, że dane użytkownika, który jest odłączony od sieci, nie są dostępne.

Serwis społecznościowy, wykorzystując zalety architektury peer-to-peer, miał spełniać następujące założenia:

- Dane użytkownika mogą być udostępnione wyłącznie przez niego samego.
- Udostępnienie pewnych danych dodatkowych (np. zdjęcia) mogą wymagać uprzedniej zgody użytkownika, który je udostępnia.
- Dane użytkownika, które są niepubliczne (np. te wymagające zgody), będą szyfrowane.
- Dane użytkownika będą przechowywane na jego urządzeniu mobilnym
- Użytkownik swoje dane będzie mógł w dowolnym momencie zmienić lub usunąć z efektem natychmiastowym.
- Będzie można jednoznacznie zidentyfikować znajomego, którego profil już wcześniej odwiedziliśmy.
- Będzie można zapisać (ang. cache) ostatnio oglądaną wersję profilu znajomego (za wyjątkiem jego mediów), która będzie dostępna do podglądu wówczas, gdy znajomy będzie offline.
- Połączenie internetowe użytkownika jest nielimitowane pod względem transferu danych.
- Poprawnie działać w sieciach lokalnych, w których brakuje dostępu do Internetu.
- Móc być uruchomiona w tle (nie na pierwszym planie), aby nie przeszkadzać w normalnym użytkowaniu urządzenia mobilnego.

5.1.3 Funkcje aplikacji

Pierwszą czynnością jest znalezienie przez użytkownika adresu IP oraz portu dowolnego węzła, który już jest połączony z interesującą użytkownika siecią peer-to-peer. Ponieważ niemożliwe jest stworzenie jednej globalnej sieci (Internet mobilny), nie będzie nic złego w tworzeniu małych sieci lokalnych wśród znajomych (np. w sieciach lokalnych za pośrednictwem WiFi). W przypadku, gdy nie istnieje jeszcze żadna sieć lub nie jest dostępna można utworzyć nową sieć łącząc się z samym sobą. Wówczas kolejni użytkownicy będą mogli się do niej dołączyć i przystąpić do korzystania z serwisu społecznościowego.

Użytkownik definiuje swój profil tzn. wypełnia dane, przynajmniej imię i nazwisko, a następnie aplikacja umieszcza w DHT rekord (zestaw danych) pozwalający na wyszukanie tego profilu w serwisie przez pozostałych użytkowników. Rekord umieszczany w DHT

zawiera parę indeks-wartość, gdzie indeksem jest ciąg znaków opisujący profil (tu imię, nazwisko lub imię i nazwisko) a wartością jest URL (tu adres IP oraz port TCP), pod którym znajduje się usługa udostępniania profilu.

Do momentu opublikowania profilu jest on niewidoczny dla pozostałych użytkowników. Znajomi użytkownika mogą znaleźć jego profil (adres IP oraz port TCP usługi udostępniania profilu) podając jego imię lub nazwisko, lub jednocześnie imię i nazwisko. Nie znając żadnej z tych danych można znaleźć użytkownika podając jego klucz publiczny jednoznacznie go identyfikujący (taki mechanizm jest użyty do lokalizowania profili zapisanych w aplikacji znajomych użytkownika). Wyszukiwanie profilu po niepełnych danych (np. wszystkie imiona zaczynające się od „A") jest niemożliwe. Użytkownik ma możliwość wyboru czy jego profil będzie publikowany również z opisem imienia i nazwiska, czy wyłącznie z kluczem publicznym. Każdy profil jest podpisywany cyfrowo kluczem prywatnym użytkownika. W momencie otrzymania danych profilowych przez inne osoby aplikacja weryfikuje podpis i informuje użytkownika oglądającego profil o tym czy podpis jest poprawny.

Aplikacja nie posiada opcji enumerowania wszystkich profili znajdujących się w sieci (technicznie jest to wykonalne ale niewydajne).

Aplikacja umożliwia przesyłanie wiadomości publicznych, które po wysłaniu pojawią się obok profilu adresata. Adresat ma możliwość usunięcia takiej wiadomości. Użytkownik może również wysłać publiczną wiadomość do siebie samego, co może być interpretowane jako odpowiednik funkcji „Co u Ciebie słychać?" ze znanych popularnych serwisów społecznościowych. Fakt pojawienia się nowej wiadomości publicznej jest sygnalizowany użytkownikowi za pomocą funkcji powiadomienia z systemu Android (powiadomienie pojawi się nawet gdy aplikacja będzie uruchomiona w tle).

Aplikacja umożliwia zapamiętanie profilu znajomego. Gdyby znajomy później był offline, wówczas przy próbie podejrzenia jego profilu zostanie pokazana ostatnio widziana wersja z pamięci (o czym przeglądający zostanie poinformowany).

Wyjście z aplikacji nie powoduje jej zamknięcia, moduły DHT oraz serwisu społecznościowego będą uruchomione w tle, jednak opcja zamknięcia aplikacji jest również dostępna dla użytkownika.

Funkcjonalności do zaimplementowania:

- Wiadomości prywatne

  Wiadomość prywatna różni się od wiadomości publicznej tym, że będzie dostępna do odczytu wyłącznie adresatowi oraz nadawcy. W celu ochrony treści wiadomości

i potwierdzenia jej pochodzenia treść wiadomości będzie podpisana cyfrowo przez nadawcę, oraz zaszyfrowana przy użyciu klucza publicznego odbiorcy.

- Przesyłanie mediów

  Funkcja przesyłania mediów (w ogólności danych binarnych) będzie użyta w celu przesłania zdjęcia profilowego użytkownika podczas odwiedzenia jego profilu przez gościa, oraz w celu przesłania pozostałych zdjęć i video udostępnianych w serwisie przez użytkownika. Wszystkie media będą szyfrowane przy użyciu klucza publicznego odbiorcy, oraz podpisane cyfrowo przez nadawcę.

5.2 Architektura aplikacji

Aplikacja została zaimplementowana w języku Java, w wersji 1.7. Język ten zapewnia przenośność kodu między różnymi platformami systemowymi, w szczególności jest kompilowalny do kodu pośredniego używanego przez system Android (Dalvik byte code).

Budowa aplikacji opiera się na zestawie modułów, które udostępniają zestawy funkcji, umożliwiające sterowanie oraz interakcję z modułem. Ponadto każdy moduł jest autonomiczny i nie pozwala na ingerencję z zewnątrz w wewnętrzne mechanizmy. Moduły w aplikacji z uwagi na wybór języka obiektowego są równoważne pakietom (ang. Java package). Zmiany dotyczące jednego modułu nie pociągają ze sobą zmian w pozostałych modułach, dla przykładu zmiana w module warstwy transportu danych nie wymaga zmian w module DHT.

Moduły, których jestem autorem

- budrys.transport.socket – Moduł odpowiedzialny za komunikację sieciową.
- budrys.chord – Moduł dostarczający funkcjonalność protokołu Chord (rozproszona tablica haszująca).
- budrys.social – Moduł serwisu społecznościowego, odpowiada za pobieranie i udostępnianie danych profilowych.
- budrys.crypt – Moduł udostępniający funkcjonalność haszowania, szyfrowania i podpisu cyfrowego.

Moduły, których nie jestem autorem, użyte w aplikacji

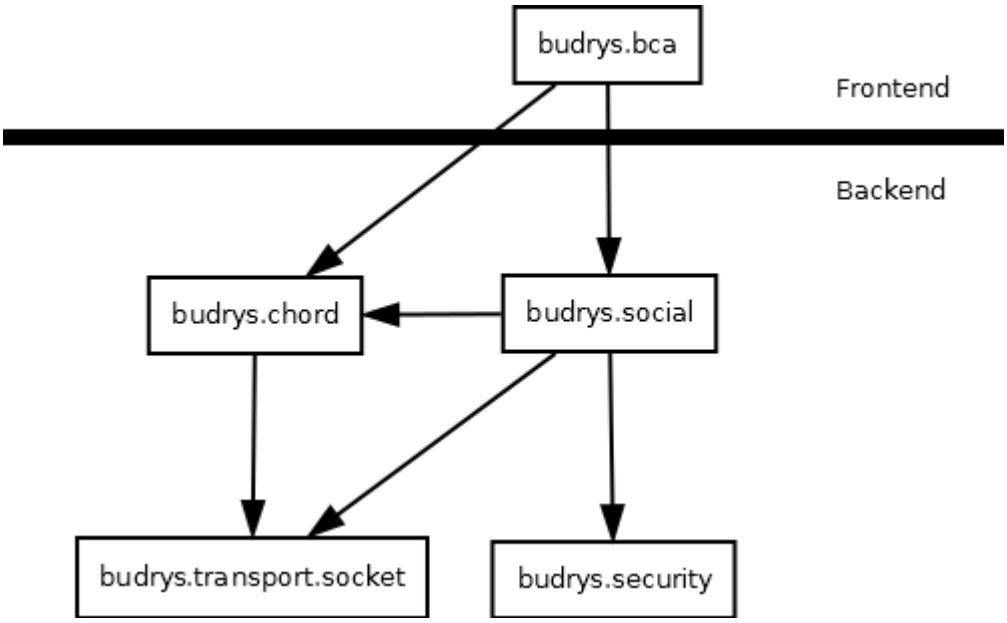
- org.json (http://www.json.org/license.html)

  Kodowanie i dekodowanie formatu JSON.
- org.apache.commons (Apache License)

  Implementuje funkcjonalność MultiValueMap.

- biz.source_code.base64Coder (GPL)

  Dostarcza funkcję kodowania do formatu base64 i funkcję dekodowania.



Rysunek 5.4: Schemat blokowy modułów aplikacji

Poniżej znajduje się szczegółowy opis modułów aplikacji.

5.2.1 Moduł transportu danych (budrys.transport.socket)

Stworzenie modułu wynikło z potrzeby istnienia żądań asynchronicznych. Stanowi on dodatkową warstwę abstrakcji między TCP a serwisem społecznościowym.

Składa się z klas:

- TransportPacket – Reprezentuje pojedynczy pakiet przesyłany przez sieć przez aplikację. Pozwala na zdefiniowanie dwóch pól tekstowych: polecenia oraz argumentu. O ile z logicznego punktu widzenia sensowne wydaje się istnienie jednego polecenia, to możliwość zdefiniowania jednego argumentu jest pewnym ograniczeniem, które zostało rozwiązane poprzez serializację obiektów złożonych z wielu argumentów do postaci tekstowej za pomocą formatu JSON.
- TransportServer – Obsługuje otwarcie portu TCP w celu nasłuchiwania oraz bezpośrednio obsługuje żądania klientów. Utworzenie instancji TransportServer wymaga zdefiniowania funkcji obsługi żądania (ang. callback) klienta. Udostępnia następujący interfejs:
  - startService
  - stopService

- TransportClient – Obsługuje proces łączenia się z instancją TransportServer i wysłanie do niej żądania oraz odebrania odpowiedzi. Podobnie jak w przypadku TransportServer utworzenie instancji TransportClient wymaga zdefiniowania funkcji obsługi żądania. Umożliwia wykonania żądania zarówno asynchronicznego jak i synchronicznego. Żądanie asynchroniczne będzie się różniło w użyciu od synchronicznego tym, że program nie zatrzyma się do momentu odpowiedzi oraz gdy odpowiedź nadejdzie, wówczas zostanie wywołana funkcja obsługi odpowiedzi (ang. callback).

5.2.2 Moduł DHT (budrys.chord)

Udostępnia funkcjonalność rozproszonej tablicy haszującej opartej na protokole Chord. Wykorzystuje funkcjonalność modułu transportu danych.

Składa się z wielu klas i interfejsów. Najważniejsze z nich to:

- IfcCallback – Interfejs, który definiuje metody wymagane przez obiekt służący do asynchronicznej obsługi odpowiedzi na żądanie. Wymaganymi metodami są *success* oraz *failure* (rozumiane jako *onSuccess, onFailure*). Program aplikacji tworzy obiekty implementujące ten interfejs przy użyciu klas anonimowych.
- ID – Klasa odpowiadająca za generowanie identyfikatorów węzłów dla protokołu Chord. Innowacją w porównaniu z innymi implementacjami Chord jest metoda generowania i używania ID, która zmniejsza możliwości manualnego manipulowania przy ID w celu poprawienia bezpieczeństwa DHT. Generowanie polega na wkomponowaniu w ID adresu IP węzła oraz portu TCP usługi. Instancja modułu Chord przechowuje informacje o pozostałych węzłach sieci peer-to-peer jako lista ID, w momencie chęci połączenia z węzłem zostaje wyłuskany z jego ID adres i port. Gdyby jakiś węzeł użył ID o spreparowanych ostatnich bajtach, wówczas połączenie się z nim byłoby niemożliwe. Strukturę wygenerowanego ID przedstawiono poniżej (opis a pod spodem numer bajtu):

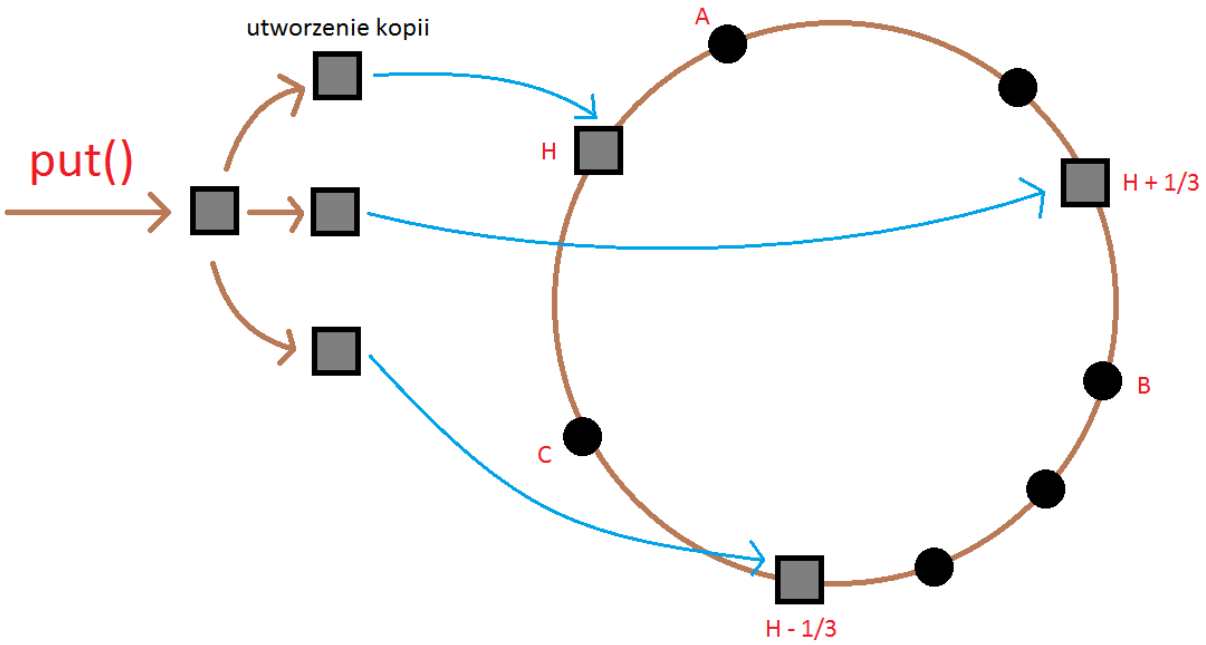
random	random	Adres IP				Port TCP	
1	2	3	4	5	6	7	8

Tabela 5.1: Konstrukcja numeru ID

- Chord – Klasa definiująca zachowanie węzła, implementuje funkcje protokołu Chord wymagane do zachowania spójności przy skalowaniu sieci. Peer – Klasa

udostępniająca funkcjonalność połączenia się z wybranym węzłem na podstawie jego ID oraz wysłania żądania związanego z protokołem Chord. Wybrane żądania (ich działanie jest takie jak w definicji protokołu Chord):

- findSuccessor – Znajdź ID węzła o najniższym ID większym od podanego.
- notify – Powiadom węzeł o tym, że być może jest naszym następnikiem.
- ping – Sprawdza, czy węzeł jest aktywny.
- ChordDHT – Klasa dziedzicząca po klasie Chord, rozszerza jej funkcjonalność o przechowywanie danych rozproszonej tablicy haszującej. Opakowuje funkcjonalność MultiValueMap, która pozwala na przechowywanie wielu wartości dla danego klucza, jednak ChordDHT ogranicza tę funkcjonalność w taki sposób, że wielokrotne umieszczenie jednakowej pary klucz-wartość jest ignorowane. Innowacją w porównaniu do innych implementacji Chord jest replikacja - każda dana jest przechowywana w trzech węzłach (może się zdarzyć że każdy wybrany do przechowywania danej węzeł będzie jednym i tym samym węzłem) w celu zapewnienia odtworzenia tej danej w przypadku awarii jednego z węzłów przechowujących daną. Implementacja posiada również mechanizm przekazywania sąsiadom danych przechowywanych przez węzeł w przypadku, gdy zamierza on opuścić sieć. Jako funkcja haszująca została użyta funkcja MD5, która choć jest już uważana za słabą kryptograficznie to w algorytmice spisuje się dobrze do rozpraszania indeksów w tablicach haszujących.
- PeerDHT – Klasa dziedzicząca po klasie Peer, rozszerza jej funkcjonalność o wysyłanie żądań dotyczących składowania i odbierania danych DHT.



Rysunek 5.5: Proces replikacji rekordów DHT

5.2.3 Moduł szyfrowania (budrys.security)

Składa się z klasy: Crypt, która udostępnia statyczne metody (funkcje) służące do:

- Wygenerowania pary kluczy RSA.
- Cyfrowego odpisywania tekstu oraz weryfikacji podpisu.
- Szyfrowania tekstu za pomocą efektywnego złożenia szyfrowania kluczem publicznym RSA oraz kluczem symetrycznym.
- Konwersji kluczy do postaci tekstowej i na odwrót.

5.2.4 Moduł serwisu społecznościowego (budrys.social)

Moduł odpowiada za udostępnianie i dostęp do danych profilowych jak również definiuje te dane. Moduł składa się z 5 klas.

- ProfileService – Uruchamia usługę udostępniania profilu. Decyduje o tym, czy żądający ma prawo dostępu do danych o ograniczonej widoczności czy jedynie do publicznych danych. Zawiera odnośniki do instancji ProfileModel oraz FriendsModel.
- RemoteProfile – Nawiązuje połączenie z wybranym węzłem i pobiera od niego dane profilowe.
- ProfileModel – Definiuje dane profilowe, umożliwia ich eksport do formatu JSON (oraz import) w celu przesłania przez sieć jako tekst lub w celu zapisania w pamięci twardej urządzenia. Wśród publicznych danych użytkownika znajduje się klucz

publiczny RSA używany do weryfikacji danych profilowych. Weryfikacja odbywa się podczas importu danych. Podpisanie elektroniczne danych odbywa się przy eksporcie i sam podpis również zostaje załączony. ProfileModel posiada odnośnik do instancji ProfileWallEntry, której dane są załączane do eksportu. Eksport można wykonać za pomocą trzech metod, które różnią się między sobą zakresem eksportowanych danych:

- exportAll – Eksport wszystkich danych, wraz z kluczem prywatnym, w celu zapisania na dysku podczas wyłączania aplikacji.
- exportRestricted – Eksport danych oznaczonych jako „o ograniczonym dostępie", w celu przesłania znajomemu.
- exportPublic – Eksport wyłącznie danych oznaczonych jako publiczne, w tym klucza publicznego RSA w celu przesłania dowolnemu użytkownikowi sieci.
- Dane profilu są eksportowane do formatu JSON, który jako tekst może być z łatwością zapisany na dysku lub przesłany przez sieć. Przykładowy profil wyeksportowany do postaci JSON (klucze RSA oraz sygnatura zostały skrócone, w celu prezentacji obiekt JSON w postaci tekstowej jest sformatowany, postać przesyłana przez sieć nie zawiera zbędnych spacji ani znaków nowej linii):


```
{
   "data":{
      "publicKey":"MIIBI...DAQAB",
      "privateKey":"MIIE...HqL0=",
      "address":"Warsaw, Poland",
      "name":"Michał",
      "surname":"Pawluczuk",
      "avatarPath":"/mnt/sdcard/avatar.png",
      "wall":[
         {
            "message":"Hello!",
            "name":"Rick Roll",
            "date":"10-12-2012 10:20"
         },
         {
            "message":"Testowa wiadomość!",
            "name":"Michał Pawluczuk",
            "date":"15-12-2012 12:50"
         }
      ]
   },
   "signature":"qUTa...hKFA=="
}
```

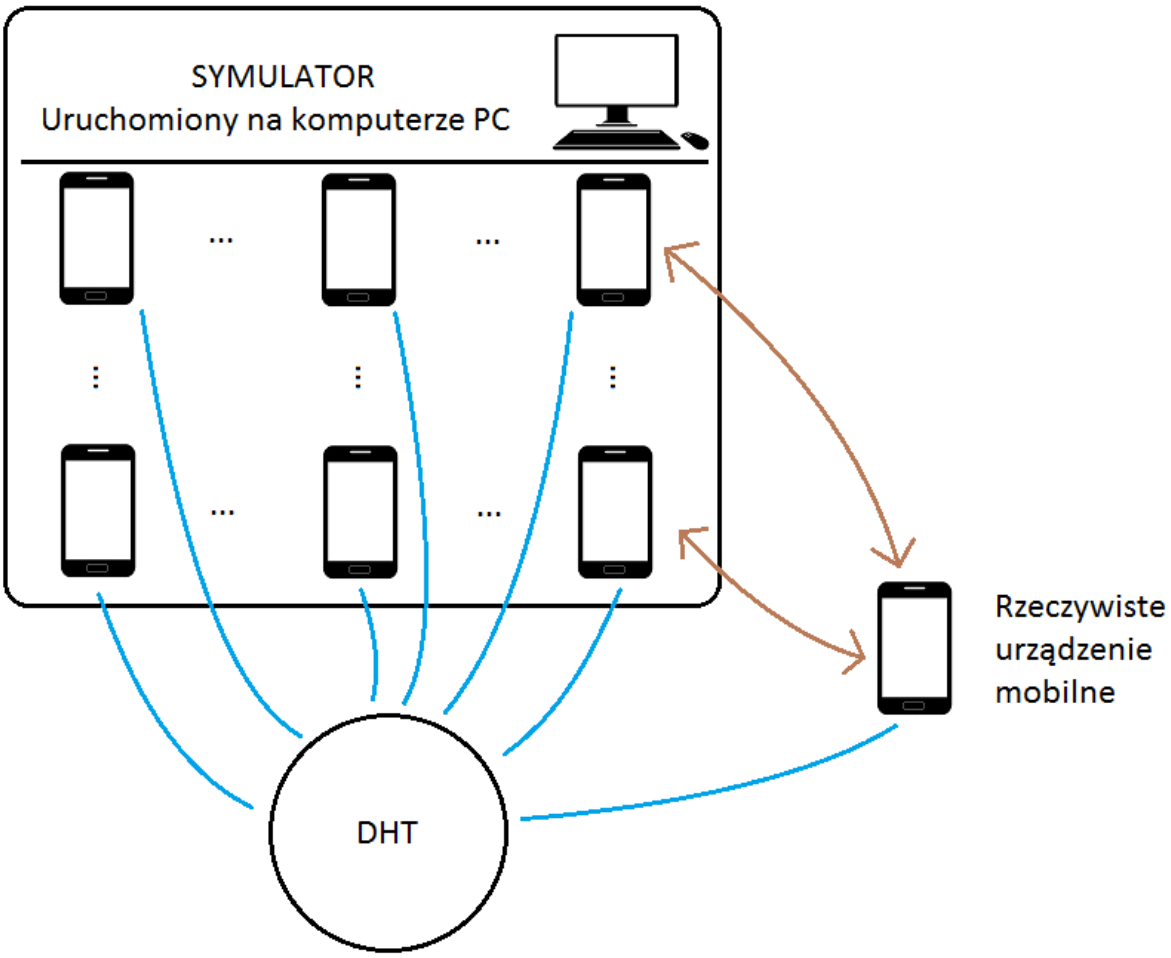
Do zaimportowania danych służy metoda *import*, w której odbywa się weryfikacja podpisu cyfrowego.

- ProfileWallEntry – Reprezentuje strukturę publicznych wiadomości przesłanych do użytkownika. Stanowi część ProfileModel. Każda wiadomość zawiera informację o dacie wysłania, nazwę nadawcy wraz z jego kluczem publicznym i podpisem cyfrowym całej struktury ProfileWallEntry.
- FriendsModel – Przechowuje informacje o tym, kto jest znajomym użytkownika. Znajomi są identyfikowani po ich kluczach publicznych. Umożliwia zapamiętanie w pamięci podręcznej (ang. cache) profilu znajomego w takim stanie jak został ostatnio widziany oraz późniejsze odtworzenie profilu z pamięci w przypadku gdy znajomy będzie offline. Pamięć podręczna jest aktualizowana w momencie gdy odwiedzimy profil znajomego, który akurat będzie online.

5.2.5 Symulator sieci

Powstanie symulatora wynikło z potrzeby przetestowania mojej implementacji protokołu Chord. Symulator i aplikacja mobilna dzielą ze sobą część kodu, ponieważ kod aplikacji mobilnej powstaje w języku Java. Ta część kodu to wszystkie moduły wymienione i opisane wcześniej. Symulator jest aplikacją Java i umożliwia uruchomienie na jednym komputerze wielu węzłów symulujących sieć peer-to-peer. Komunikacja węzłów symulowanych z innymi węzłami symulowanymi odbywa się przy użyciu protokołu TCP, czyli dokładnie tak jak z prawdziwymi węzłami. Z punktu widzenia rzeczywistego urządzenia mobilnego łączącego się z tą siecią wyglądało to tak, gdyby było połączone z prawdziwą siecią:

- Jest obecnych wiele węzłów w sieci
- Węzły dołączające do sieci / odłączające się
- Węzły awaryjnie odłączające się
- Symulowane opóźnienia w przesyłanych pakietach



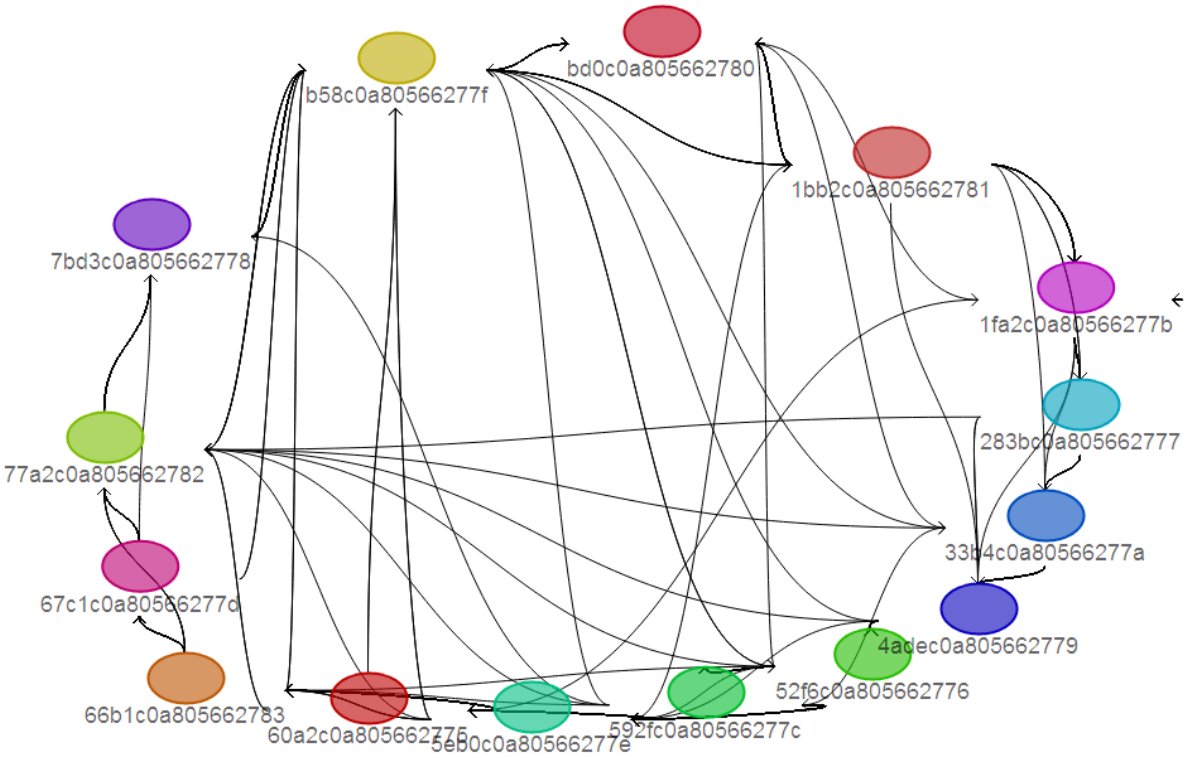
Rysunek 5.6: Rola symulatora
Linie bez strzałek oznaczają przynależność do DHT, linie ze strzałkami to połączenia bezpośrednie.

Jak pokazano na Rysunku 5.6 symulator może samodzielnie tworzyć sieć serwisu społecznościowego, może tworzyć część tej sieci wspólnie z rzeczywistymi urządzeniami mobilnymi. W ogólnym przypadku symulator nie musi być obecny w sieci serwisu.

Symulator posiada możliwość podejrzenia stanu każdego węzła przez referencję, co nie wymaga angażowania stosu TCP. Dane możliwe do podglądu to w szczególności:

- Poprzednik i następnik węzła
- Węzły dalsze (ang. fingers)
- Dane DHT przechowywane lokalnie

Program potrafi wyeksportować stany wszystkich symulowanych węzłów do formatu JSON, co pozwoliło wygenerować poniższy schemat pokazujący strukturę sieci peer-to-peer na której oparto serwis.



Rysunek 5.7: Graf sieci Chord na podstawie danych z symulatora

5.3 Koncepcje technologii

W tej części zostaną omówione koncepcje technologii możliwych do zastosowania w aplikacji serwisu społecznościowego. Zostały one podzielone na trzy części. Pierwszą z nich są zabezpieczenia aplikacji, które już zostały zaimplementowane. Kolejną częścią są podatności aplikacji na ataki wraz z opisami scenariuszów. W ostatniej części zostały omówione koncepcje zwiększenia - z zachowaniem prywatności - dostępności danych profilowych użytkowników.

5.3.1 Zabezpieczenia serwisu

Dane DHT są replikowane na dwóch dodatkowych węzłach w celu zwiększenia niezawodności sieci. W przypadku gdy jeden z węzłów przechowujących pewną daną rozłączy się awaryjnie, bez możliwości przekazania jej sąsiadowi, wówczas ta dana zostanie odtworzona na podstawie repliki znajdującej się na jednym z dwóch dodatkowych węzłów.

Szczególny przypadek utraty danej z DHT (gdy wszystkie węzły przechowujące tę daną opuszczą awaryjnie sieć) zostanie naprawiony przez Moduł serwisu społecznościowego poprzez ponowne umieszczenie w DHT danej pozwalającej na zlokalizowanie profilu użytkownika.

Dane przesyłane przez sieć są szyfrowane kluczem publicznym RSA o długości 2048 bitów, możliwe do rozszyfrowania wyłącznie przez odbiorcę, czyli użytkownika posiadającego odpowiedni klucz prywatny.

Dane profilowe użytkownika są zabezpieczone podpisem cyfrowym przed ewentualną modyfikacją na poziomie łącza danych (np. przez dostawców Internetu). Jednocześnie odbiorca ma pewność, że dane pochodzą od konkretnej osoby, pod warunkiem, że wcześniej poznał jej klucz publiczny (najlepiej przy pomocy innego medium, np. osobiście lub za pomocą wiadomości SMS, w przyszłych wersjach aplikacji pojawi się opcja wysłania z poziomu aplikacji wiadomości SMS zawierającej klucz publiczny - wizytówkę).

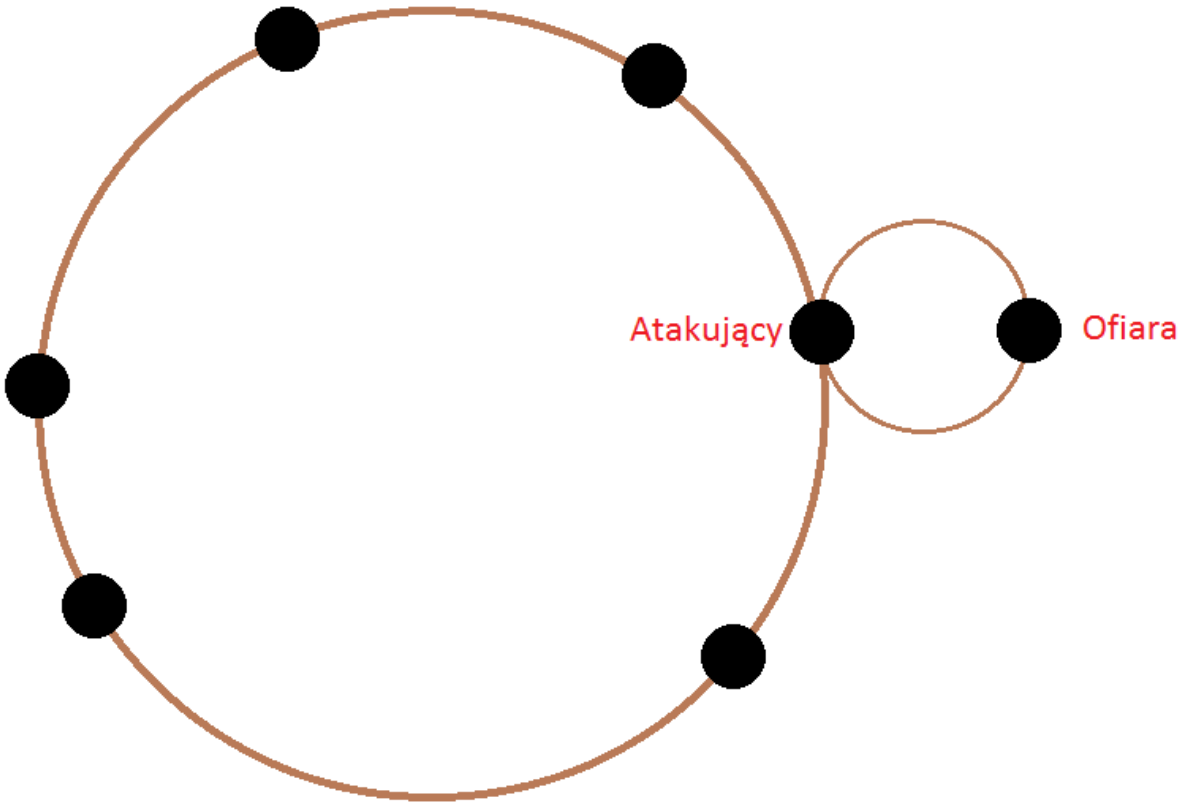
W oryginalnej wersji protokołu Chord każdy węzeł sieci losuje sobie numer ID, co jest zagrożeniem, ponieważ może sobie wybrać dowolne miejsce w pierścieniu i próbować izolować pozostałe węzły. W mojej implementacji manipulacja przy ID protokołu Chord jest mocno ograniczona, ponieważ 4 bajty pochodzą od adresu IP węzła, 2 bajty od portu TCP, na którym stoi usługa DHT, oraz 2 bajty losowane (dowolne). Całość tworzy 8 bajtowy ID, gdzie losowane bajty są przesunięte maksymalnie w stronę najbardziej znaczących bitów. W efekcie atakujący może wybrać sobie miejsce w dowolnej części pierścienia, na którym jest oparty protokół Chord, ale ze stosunkowo niewielką dokładnością. Dowolność w zakresie wyboru przez węzeł 2 bajtów zastosowano w celu zachowania kompromisu między ograniczeniem możliwości manipulowania numerami ID, a możliwością rozproszenia węzłów, które posiadają zbliżone do siebie adresy IP.

5.3.2 Scenariusze ataków na serwis społecznościowy

Serwis społecznościowy nie jest zabezpieczony przed wszystkimi zagrożeniami ze strony nieuczciwych użytkowników. Przed częścią zagrożeń trudno jest się zabezpieczyć, a na implementację pozostałych zabezpieczeń zabrakło czasu. Poniżej przedstawiono podatności serwisu wraz z koncepcjami ich zlikwidowania. Warto zwrócić uwagę, że część z przedstawionych podatności wymaga użycia przez atakującego zmodyfikowanej (nieoficjalnej wersji) aplikacji.

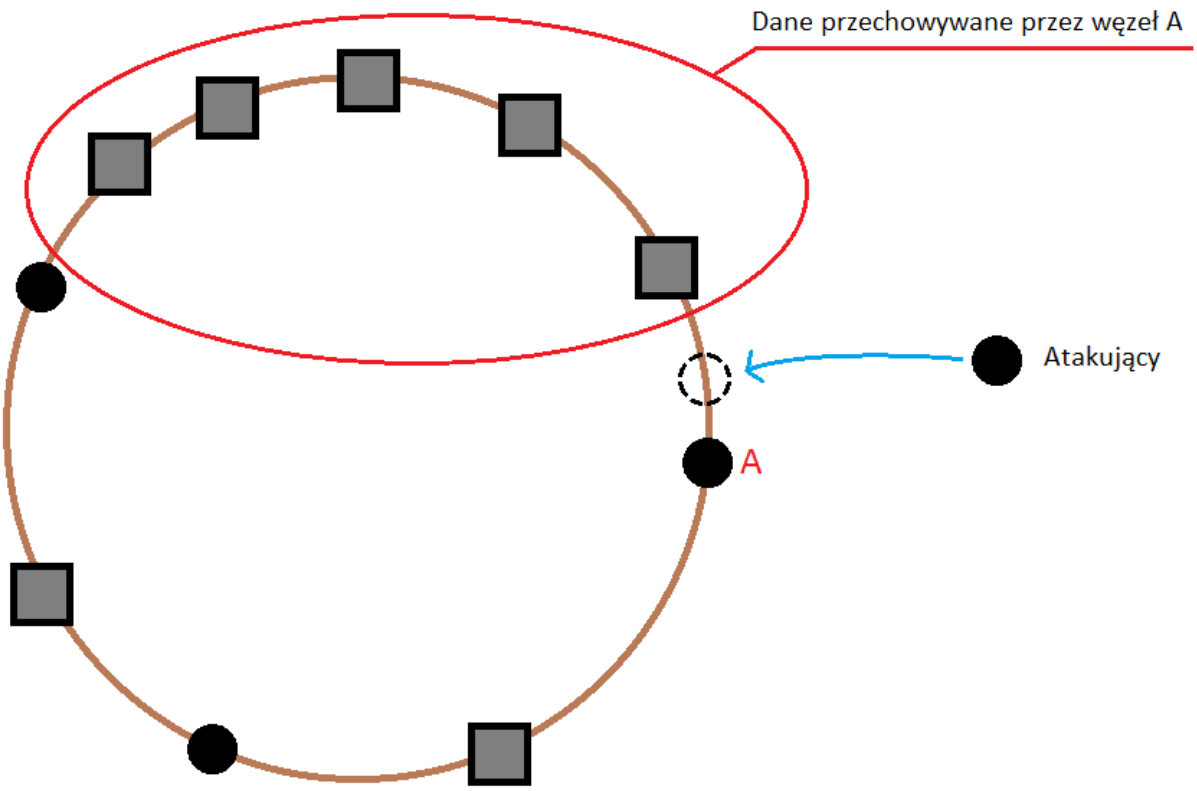
Izolacja nowych węzłów sieci peer-to-peer – atak możliwy do przeprowadzenia na poziomie DHT. Celem atakującego jest wyizolowanie nowo dołączających węzłów w celu uniemożliwienia im dołączenia do sieci. Warunkiem powodzenia jest dołączenie węzła ofiary do sieci za pośrednictwem węzła atakującego, którego odpytuje o następnika. Atakujący odpowiada, że następnikiem ofiary jest on sam, po czym wysyła informację, że jej poprzednikiem również jest on sam. Ofiara wnioskuje, że sieć jest kompletna i zawiera tylko

dwa węzły. Niestety ten atak wynika z pewnego minimalnego zaufania w sieciach peer-to-peer, które w takich podejściach jest wymagane. Aplikacja nie będzie w stanie stwierdzić przeprowadzenia takiego ataku. Jedynym wyjściem z tej sytuacji jest połączenie się ofiary z innym węzłem, który przypuszczalnie będzie w porządku.



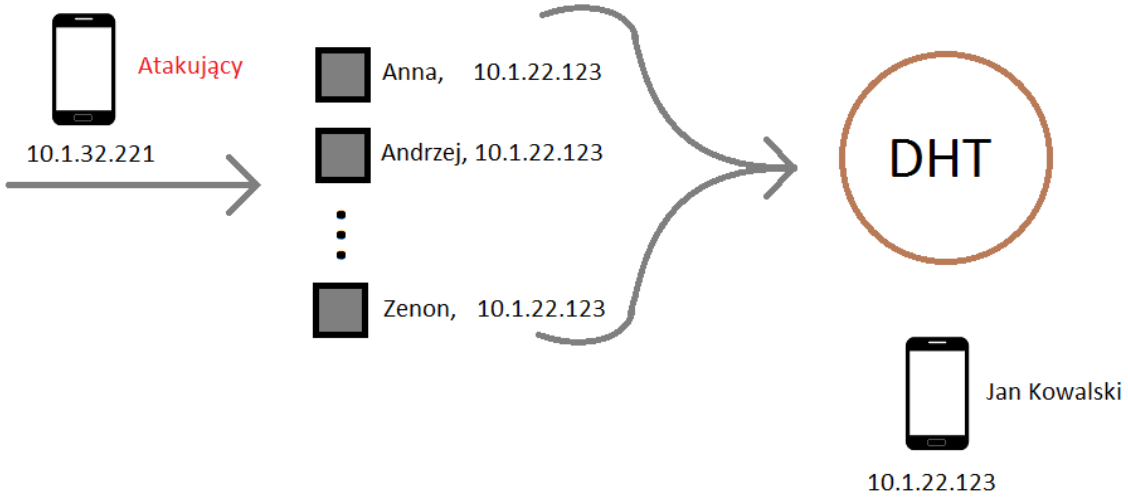
Rysunek 5.8: Izolacja nowych węzłów

Przejmowanie danych DHT – celem atakującego jest przejęcie danych DHT oraz nieudostępnianie ich nikomu. Przeprowadzenie ataku wymaga zmodyfikowania kodu aplikacji, aby na każde żądanie GET odpowiadała brakiem rekordów. W celu ograniczenia atakującemu możliwości wyboru danych o pożądanym haszu dowolność w generowaniu numerów ID została ograniczona do 2 bajtów. Ponadto każda dana DHT jest rozproszona po 3 węzłach i w przypadku nieznalezienia jej u jednego węzła jest szukana u pozostałych.



Rysunek 5.9: Przejmowanie danych DHT

Zaśmiecanie wyników wyszukiwania – atakujący umieszcza w DHT wiele rekordów o kluczach podobnych do rekordów z danymi o istniejących profilach w celu zaśmiecenia wyników wyszukiwania i utrudnienia pozostałym odnalezienia właściwego profilu. Zabezpieczenie polega na odpytaniu węzła na który wskazuje dana rekordu z DHT o publiczne dane profilowe i dopiero w przypadku uzyskania poprawnej odpowiedzi (zgodnej z protokołem aplikacji) rekord jest wyświetlany w wynikach wyszukiwania. W celu ominięcia takiego zabezpieczenia atakujący musiałby uruchomić usługi udostępniania profilu dla każdego śmieciowego rekordu.



Rysunek 5.10: Zaśmiecanie DHT fałszywymi danymi i spamowanie węzła

Spamowanie komendami – atakujący może nieprzerwanie wysyłać do ofiary komendy w celu zmniejszenia wydajności lub odłączenia ofiary od sieci. Szczególnie wrażliwą komendą jest *getSuccessor*, która wymaga od węzła przeszukiwania sieci peer-to-peer. Zabezpieczeniem przed tym jest zaimplementowanie tej komendy tak, aby żądanie zwracało odpowiedzi nieautorytatywne, czyli w przypadku stwierdzenia przez węzeł, że on sam nie może udzielić pełnej odpowiedzi, odpowiedzią będzie ID węzła, który być może takiej odpowiedzi udzieli. W takim podejściu każdy węzeł wykonuje zapytania na własną rękę i żaden inny węzeł nie zadaje pytań w jego imieniu, co redukuje skutki spamowania żądaniami.

Spamowanie węzła-ofiary komendami przez normalnych użytkowników – atakujący może umieścić w DHT wiele rekordów oznaczonych imionami i nazwiskami popularnych osób wskazujące na jedno urządzenie - urządzenie ofiary. W efekcie zwykli użytkownicy wyszukujący popularne imiona lub nazwiska będą za każdym wyszukaniem odpytywać ofiarę o udostępnienie publicznej części jej profilu. Tę sytuację przedstawia Rysunek 5.7. Metoda uodpornienia serwisu na takie działanie powinna sprawdzać, czy umieszczany rekord z danym imieniem i nazwiskiem zgadza się z nazwą w udostępnianym profilu, jednak wiązałoby się z dodatkowym obciążeniem docelowego węzła, szczególnie w przypadku redystrybucji rekordów DHT wskutek reorganizacji sieci przy dołączających lub odłączających węzłach.

Powyższy przypadek można rozszerzyć do scenariusza, w którym atakujący usiłuje podmienić rekord w DHT opisany kluczem publicznym pewnego użytkownika, w celu podszycia się pod niego. W efekcie zapytanie o profil znajomego z poziomu listy znajomych

w aplikacji dotarłoby nie do znajomego, a do atakującego. Oczywiście próba przesłania przez atakującego zmodyfikowanego profilu ofiary będzie zasygnalizowana odbiorcy niepomyślnym zweryfikowaniem podpisu cyfrowego profilu, ale odbiorca nie będzie w stanie zlokalizować profilu atakowanego znajomego. Opisywany atak może być pomyślnie przeprowadzony z użyciem koncepcji Replay Attack, która polega na zapisaniu przez atakującego podpisanego poprawnie cyfrowo profilu ofiary w momencie gdy udostępnił ona przypadkiem jakieś kompromitujące ją informacje, oraz udostępnianie tego profilu później w niezmienionej formie. Funkcja replikowania rekordów w DHT również może nie być wystarczającym zabezpieczeniem przed tym atakiem, ponieważ atakujący może podmienić również repliki. Skutecznym zabezpieczeniem byłoby odpytywanie węzła (pytającym jest węzeł, który będzie przechowywał tę daną), na którego ma wskazywać rekord DHT, o podpisanie cyfrowe pewnej losowej wiadomości wygenerowanej losowo przez pytającego i weryfikację podpisu kluczem publicznym z wstawianego do DHT rekordu. Wadą rozwiązania jest konieczność odpytywania węzła wskazywanego przez rekord przy każdym umieszczeniu go w DHT oraz przy każdej replikacji, lub redystrybucji rekordu.

Fałszowanie publicznych wiadomości – w tym przypadku atakującym jest właściciel profilu. Celem jest opublikowanie na własnym profilu wiadomości rzekomo pochodzącej od innego użytkownika (lub podmiana treści istniejącej wiadomości), opatrzonej imieniem i nazwiskiem rzekomego nadawcy. Rozwiązaniem jest podpisywanie cyfrowe publicznych wiadomości przez nadawców i weryfikacja tych podpisów przez odwiedzających profil.

Publikowanie prywatnych wiadomości – atakującym jest właściciel profilu, który otrzymał prywatną wiadomość. Celem atakującego jest upublicznienie prywatnej wiadomości (powodem może być szantaż nadawcy), która jest podpisana cyfrowo przez nadawcę i zaszyfrowana kluczem publicznym odbiorcy (atakującego). Utrudnieniem jest zamiana kolejności podpisywania cyfrowego z szyfrowaniem wiadomości, czyli najpierw następuje szyfrowanie wiadomości kluczem publicznym a dopiero zaszyfrowana wiadomość jest podpisywana. Atakujący w dalszym ciągu może taką wiadomość upublicznić, ale intuicyjnie są dostępne dwa warianty: albo publikuje rozszyfrowaną wiadomość bez podpisu cyfrowego (brak dowodu nadania), albo publikuje zaszyfrowaną wiadomość wraz z podpisem cyfrowym, ale musi dołączyć również swój klucz publiczny, co na ogół wiąże się z udostępnieniem wszystkim nieograniczonego dostępu do profilu atakującego (dowód nadania jest, ale skutki opublikowania klucza prywatnego mogą być bardzo niekorzystne dla atakującego). Niestety jest też trzeci wariant: atakujący publikuje zaszyfrowaną wiadomość wraz z podpisem cyfrowym, oraz rozkodowaną wiadomość wraz z kluczem publicznym, którym zaszyfrowano tę wiadomość. Użytkownicy chcąc zweryfikować sygnaturę wiadomości mogą wykonać

szyfrowanie wiadomości kluczem publicznym, a następnie porównać zaszyfrowane postacie wiadomości (tej zaszyfrowanej przez siebie z oryginalną - podpisaną).

5.3.3 Zwiększanie funkcjonalności i dostępności danych

Wymóg gotowości użytkownika do udostępniania profilu za każdym razem, gdy ktoś inny chce podejrzeć jego profil, jest dosyć rygorystyczny. W serwisach społecznościowych opartych o centralny serwer (lub grupę serwerów) dane użytkownika są dostępne przez cały czas od momentu opublikowania do usunięcia przez użytkownika, co stanowi bezpośrednią konkurencję dla serwisów opartych na architekturze peer-to-peer. Aby wyjść naprzeciw wymaganiom użytkowników powstała koncepcja zwiększenia dostępności danych profilowych.

Idea polega na umieszczaniu danych profilowych bezpośrednio w DHT, wówczas dane będą dostępne tak długo jak DHT będzie istniało. Danymi profilowymi są imię, nazwisko, miejscowość, i wiadomości publiczne. Należy zwrócić uwagę, że w przyszłości będą to również media (zdjęcia, video). Dane profilowe, będąc przechowywanymi w DHT, powinny być w postaci zaszyfrowanej, aby ograniczona grupa osób miała do nich dostęp. Rozwiązanie zapewnia kryptografia z użyciem kluczy asymetrycznych: szyfrowanie danych kluczem publicznym odbiorcy, który użytkownik udostępniający dane powinien znać przed ich opublikowaniem. Mamy do dyspozycji dwa warianty:

1. Dane są szyfrowane oddzielnie dla każdego odbiorcy używając kluczy publicznych każdego z osobna
2. Udostępniający generuje parę kluczy asymetrycznych, po czym szyfruje profil kluczem publicznym, a klucz prywatny przesyła odbiorcom.

Wadą pierwszego podejścia jest konieczność umieszczania danych profilowych w DHT w tylu kopiach, w ilu jest odbiorców – dla każdego po jednej. Wadą drugiego podejścia jest fakt dzielenia się kluczem prywatnym (wygenerowanym specjalnie dla danej; klucz prywatny, który służy do podpisywania profilu nigdy nikomu nie zostaje przekazany), co powoduje, że w przyszłości w razie potrzeby nie będzie można zawęzić grona odbiorców. Rozwiązaniem może być generowanie pary kluczy odrębnej dla każdej danej i dystrybuowanie klucza odszyfrowującego każdemu osobnie wedle życzenia udostępniającego.

Wadą całego podejścia do zwiększania dostępności danych profilowych jest fakt, że użytkownik w pewnym momencie traci kontrolę nad swoimi danymi (brak możliwości usunięcia raz umieszczonej danej). Dzieje się to w momencie umieszczenia danych w DHT. Z

tego powodu ta opcja byłaby dostępna na wyraźne życzenie użytkownika (domyślnie wyłączona).

6 ZAKOŃCZENIE

Z powodu blokady połączeń TCP przychodzących w Internecie mobilnym niemożliwe okazuje się utworzenie globalnej sieci w ramach serwisu. Aplikacja jednak dobrze spisuje się w przypadku małych sieci lokalnych opartych o router z funkcją WiFi.

Wykonane przez znajomych alfa testy aplikacji w zakresie tworzenia, publikowania i wyszukiwania profilu, oraz przesyłania wiadomości pozwoliły wyłapać błędy implementacji oraz pokazały, że aplikacja jest używalna. Grupa testująca liczyła 10 osób, testerzy dysponowali smartfonami z różnymi wersjami systemu Android w zakresie wersji od 2.1 do 4.0.3 oraz różnymi przekątnymi ekranu. Aplikacja korzysta ze standardowych interfejsów systemu Android, co pozwala na uruchomienie jej używając dowolnej wersji systemu począwszy od wersji 2.1 wzwyż. W szczególności nie jest wymagane przeprowadzenie procesu rootowania urządzenia użytkownika.

Po zaimplementowaniu mechanizmów szyfrowania oraz przesyłania mediów będzie można umieścić aplikację w internetowym sklepie Google Play (http://play.google.com), co pozwoli trafić do większego grona osób.

Aplikacja znajdzie zastosowanie wśród osób dbających o swoją prywatność, w szczególności tych, którzy obawiają się powierzania swoich prywatnych danych trzecim stronom.

W przypadku uzyskania pozytywnego echa od pierwszych użytkowników kolejnym krokiem prawdopodobnie będzie udostępnienie kodu aplikacji na otwartej licencji. Powstały serwis społecznościowy, z powodu zastosowania architektury peer-to-peer, niekoniecznie znajdzie zastosowanie komercyjne.

W przypadku postępu technologicznego związanego z globalnym wprowadzeniem protokołu internetowego IPv6 rozwiązałby się problem z prywatnymi adresami IP, ponieważ wówczas adresów publicznych starczyłoby dla każdego urządzenia. W połączeniu z przypadkiem gdyby operatorzy telefonii komórkowej zdecydowali się odblokować nawiązywanie połączeń przychodzących w stronę użytkowników Internetu mobilnego problem bezpośrednich połączeń między urządzeniami mobilnymi w skali globalnej zostałby rozwiązany. Wówczas kolejnym krokiem byłoby utworzenie dodatkowych punktów centralnych serwisu pozwalających na znalezienie adresów będących już w sieci peer-to-peer urządzeń w celu dołączenia się do sieci przez nowych użytkowników.

Aplikacja posiada potencjał pozwalający na realne jej wdrożenie. Z wersji alfa, choć jeszcze sporadycznie, sam już korzystam razem z moimi znajomymi.

7 LITERATURA

[1]. I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, H. Balakrishnan. http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf MIT Laboratory for Computer Science

[2]. D. Ehringer. http://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf 2010

[3]. M. Gargenta. Learning Android. O'Reilly Media. 2011 http://ofps.oreilly.com/titles/9781449390501/The_Stack.html

[4]. B. Kaliski. PKCS #1: RSA Encryption. http://www.ietf.org/rfc/rfc2313.txt

[5]. M. Blaze, J. Ioannidis, A. Keromytis. DSA and RSA Key and Signature Encoding for the KeyNote Trust Management System. http://tools.ietf.org/html/rfc2792

[6]. Dynamic DNS client for Android, http://l6n.org/android/dyndns.shtml

[7]. E. Rosen, Y. Rekhter. BGP/MPLS IP Virtual Private Networks (VPNs). http://www.ietf.org/rfc/rfc4364.txt

[8]. R. de Wolf. Quantum Computation and Shor's Factoring Algorithm. http://homepages.cwi.nl/~rdewolf/publ/qc/survey.pdf 1999

8 WYKAZ UŻYWANYCH SKRÓTÓW

DHT – Rozproszona Tablica Haszująca (ang. Distributed Hash Table)

GSM – Global System for Mobile Communications

ID – identyfikator; w kontekście protokołu Chord jest numerem, którym oznaczono węzeł

IP – Internet Protocol

P2P – Peer-to-peer

P4 – P4 Spółka z o.o. operator sieci Play

SDK – Software Development Kit

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

URL – Universal Resource Locator

VPN – Virtual Private Network

WiFi – synonim WLAN – Wireless Local Area Network

9 ZAŁĄCZNIKI

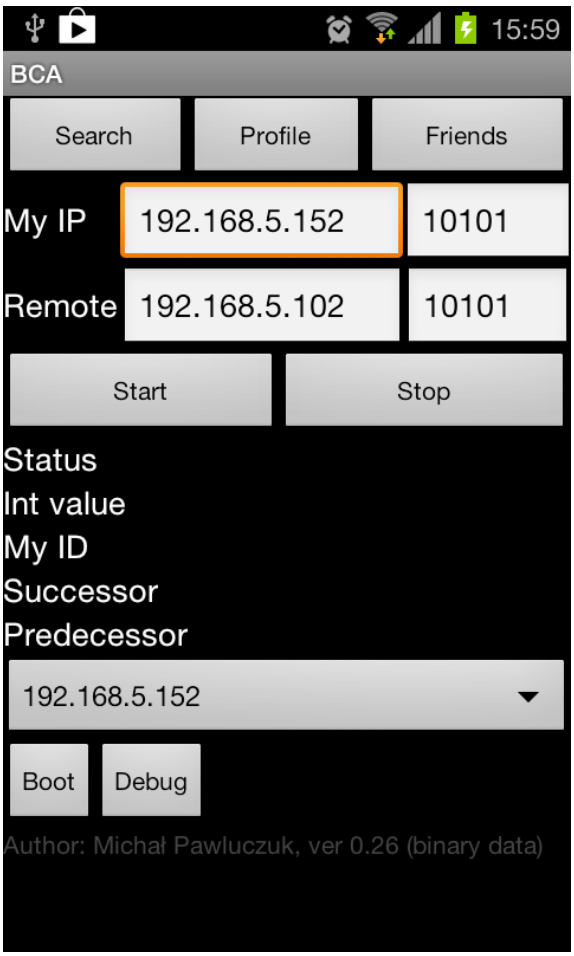
9.1 Instrukcja użytkownika

9.1.1 Instalacja aplikacji

W celu zainstalowania aplikacji na urządzeniu z systemem Android należy przesłać do niego plik BCA.apk, który znajduje się na załączonej płycie CD. Następnym krokiem jest uruchomienie przesłanego pliku w celu zainstalowania. System Android poprosi w imieniu instalowanej aplikacji o przydzielenie jej uprawnień do modyfikacji danych w pamięci trwałej urządzenia oraz o dostęp do komunikacji sieciowej. W przypadku nieudzielenia zgody aplikacja nie zainstaluje się. W przypadku akceptacji aplikacja zostanie zainstalowana w systemie.

9.1.2 Uruchomienie i połączenie się z serwisem

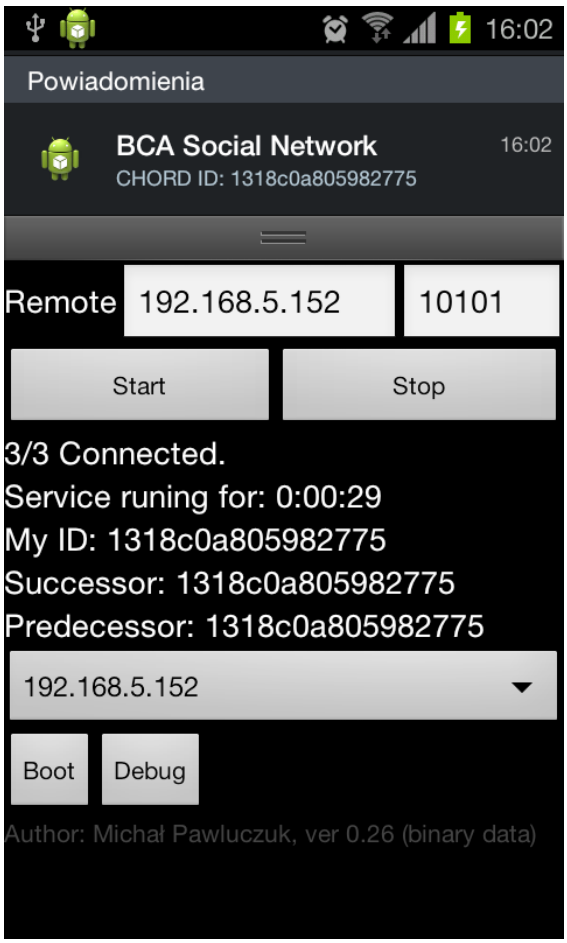
Aby uruchomić aplikację należy wcisnąć jej ikonkę na liście zainstalowanych aplikacji. Po uruchomieniu pokaże nam się ekran główny aplikacji:



Rysunek 8.1: Ekran główny aplikacji

W celu połączenia się z serwisem społecznościowym musimy w jakiś sposób znaleźć adres IP oraz port TCP urządzenia, które już z serwisem jest połączona. Zakładając, że jesteśmy pierwszą osobą w tym serwisie powinniśmy go utworzyć, ponieważ jeszcze nie istnieje. W tym celu wciskamy przycisk *Boot*. W efekcie w polu *Remote*, gdzie jest adres urządzenia, z którym chcemy się połączyć, zostanie wpisany nasz adres IP. Po prawej stronie pól z adresami IP znajdują się pola tekstowe określające numer portu TCP, pod którym inne węzły będą mogły z nami nawiązać połączenie. W dolnej części ekranu nad przyciskiem *Boot* znajduje się lista rozwijalna zawierająca adresy IP aktywnych interfejsów sieciowych naszego urządzenia. Wybór adresu IP z listy powoduje skopiowanie go do pola *MyIP*.

Ponieważ wybraliśmy opcję *Boot*, dlatego będziemy łączyć się z samym sobą w celu utworzenia nowej sieci serwisu społecznościowego. Po naciśnięciu przycisku *Start* zostanie uruchomiona w tle usługa serwisu społecznościowego oraz nawiązane połączenie:



Rysunek 8.2: Połączenie z serwisem

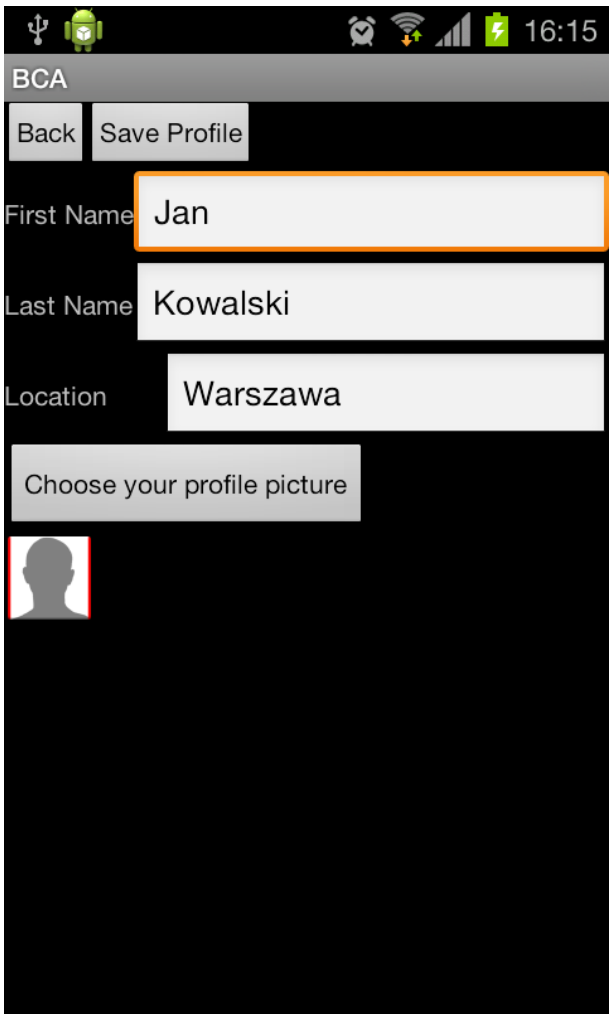
Po ustanowieniu połączenia z siecią serwisu zostanie pokazane powiadomienie w systemowym ekranie powiadomień (na górze ekranu) zawierające nasz numer ID. Ponadto również numery ID sąsiadów zostaną pokazane w polach *Successor* oraz *Predecessor*.

9.1.3 Podgląd i edycja profilu użytkownika

Mając ustanowione połączenie możemy udać się do ekranu profilu.

Rysunek 8.3: Podgląd profilu

Widzimy imię i nazwisko, które przed skonfigurowaniem nie będą ustawione. W celu ustawienia naszych danych wciskamy przycisk *Edit*. Pokaże nam się następujący ekran:

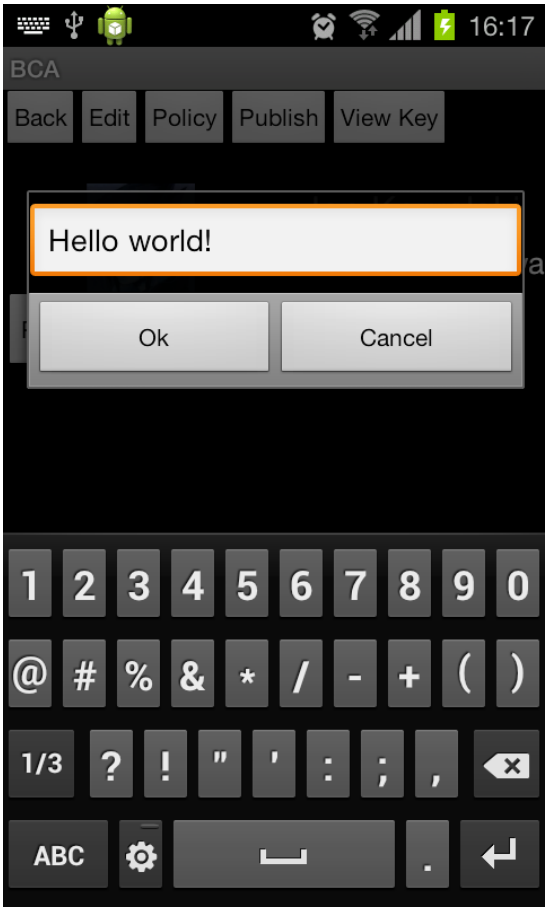


Rysunek 8.4: Edycja profilu

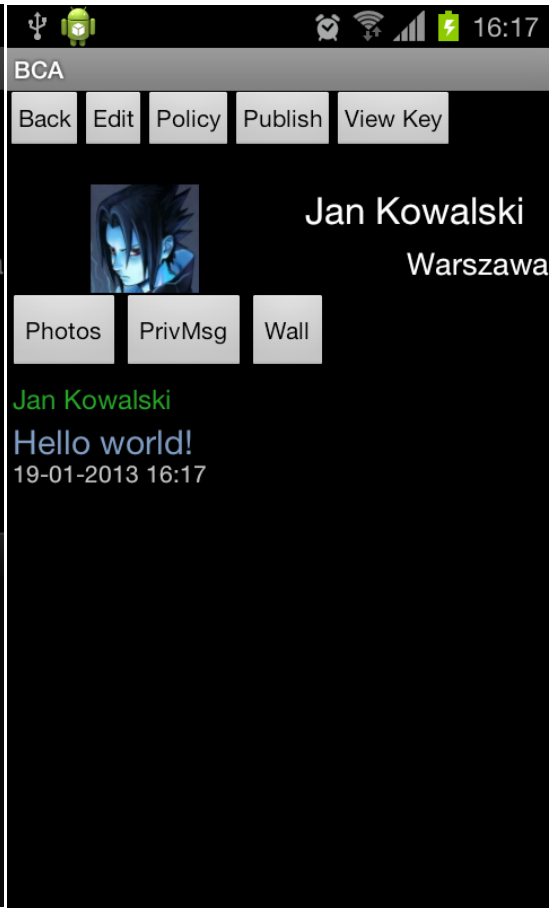
W ekranie edycji profilu możemy wpisać nasze imię i nazwisko oraz miejscowość. Możemy również wybrać zdjęcie profilowe (avatar) wciskając przycisk *Choose your profile Picture*. Po zakończonej edycji wciskamy przycisk *Save profile*, a następnie przycisk *Back*, który przeniesie nas z powrotem do ekranu podglądu profilu.

Wciskając przycisk *Publish* (Rysunek 8.3) opublikujemy w sieci informację o naszym profilu, która pozwoli pozostałym użytkownikom zlokalizować nasze urządzenie, które udostępnia profil o danym imieniu i nazwisku. Możemy podejrzeć nasz kryptograficzny klucz publiczny wciskając przycisk *View Key*.

Za pomocą przycisku *Wall* możemy opublikować na naszym profilu wiadomość publiczną (Rysunek 8.5), która będzie widoczna dla każdego odwiedzającego nasz profil (Rysunek 8.6).



Rysunek 8.5: Publikowanie wiadomości

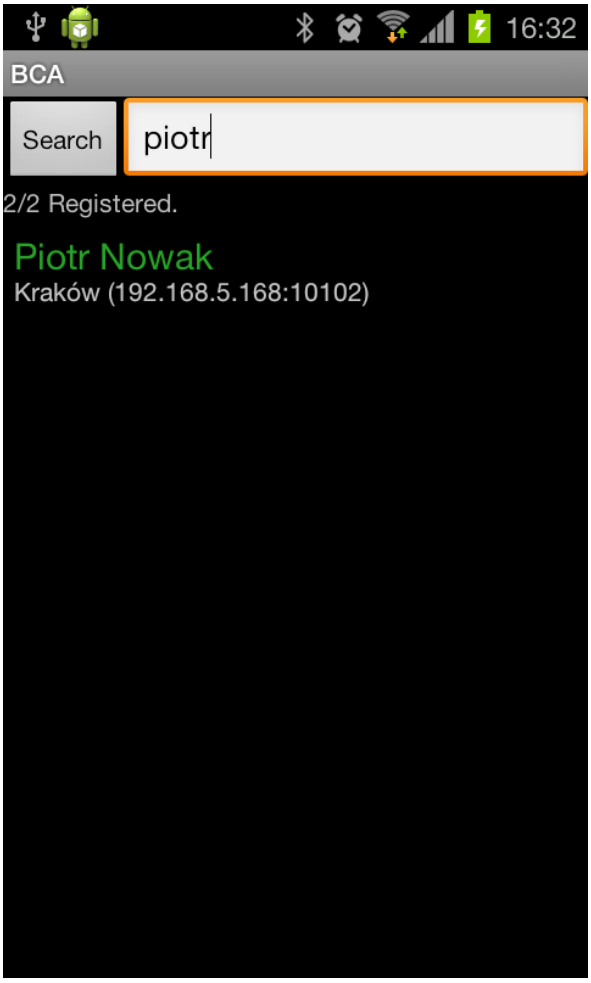


Rysunek 8.6: Publiczna wiadomość

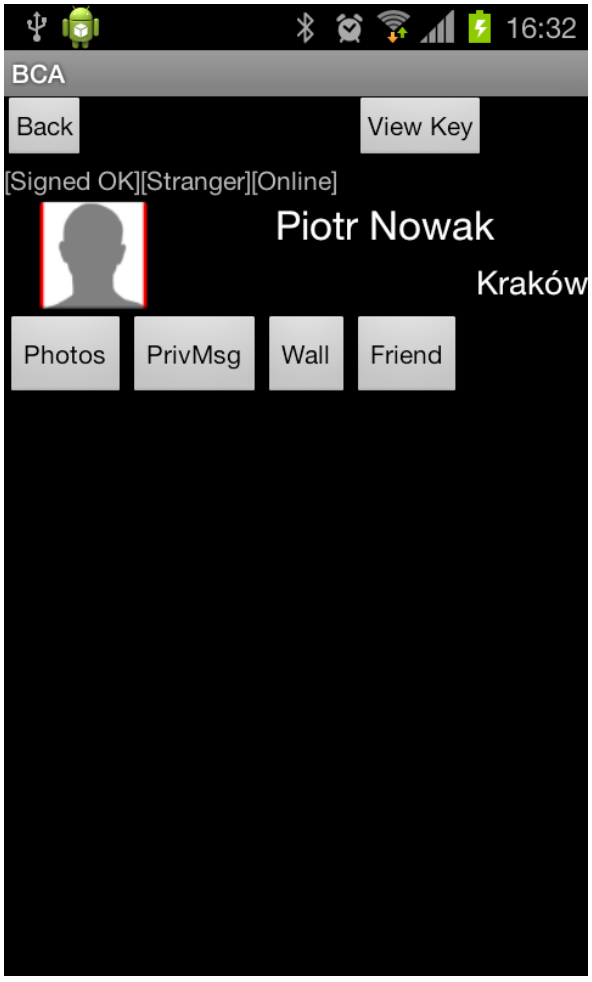
Po wciśnięciu i przytrzymaniu wiadomości na liście pokaże nam się okno dialogowe pozwalające usunąć publiczną wiadomość.

9.1.4 Wyszukiwanie profili użytkowników

Wciskając przycisk *Search* (Rysunek 8.1) otworzymy ekran wyszukiwania (Rysunek 8.7). Po wpisaniu pełnego imienia lub pełnego nazwiska lub imienia i nazwiska oraz wciśnięciu przycisku *Search* (będąc na ekranu z Rysunku 8.7) ukaże się nam lista znalezionych profili.



Rysunek 8.7: Wyszukiwanie profili użytkowników



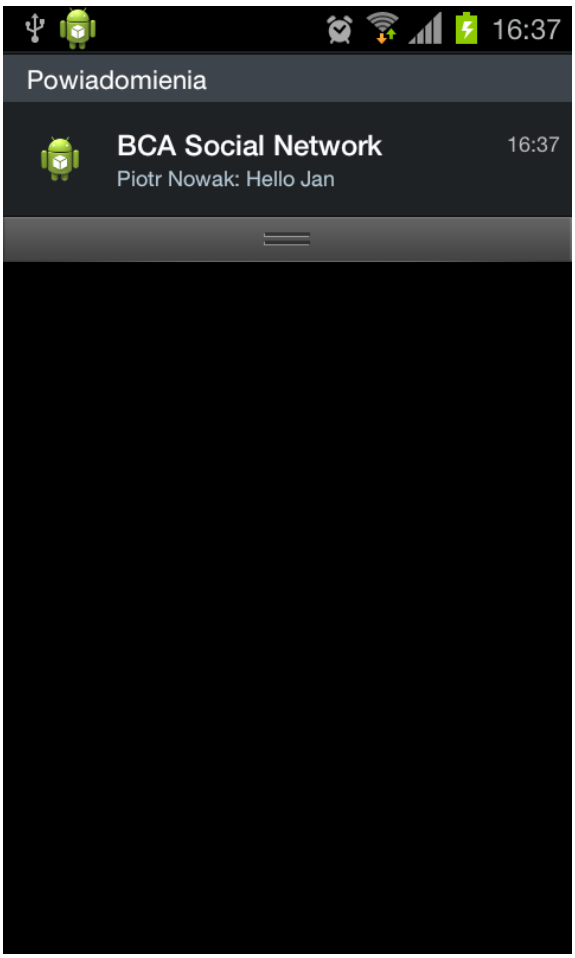
Rysunek 8.8: Podgląd cudzego profilu

Po wciśnięciu pozycji z listy aplikacja przeniesie nas do ekranu podglądu wybranego profilu (Rysunek 8.8). W odróżnieniu od ekranu podglądu własnego profilu tutaj pojawiły się pod przyciskiem *Back* dodatkowe informacje:

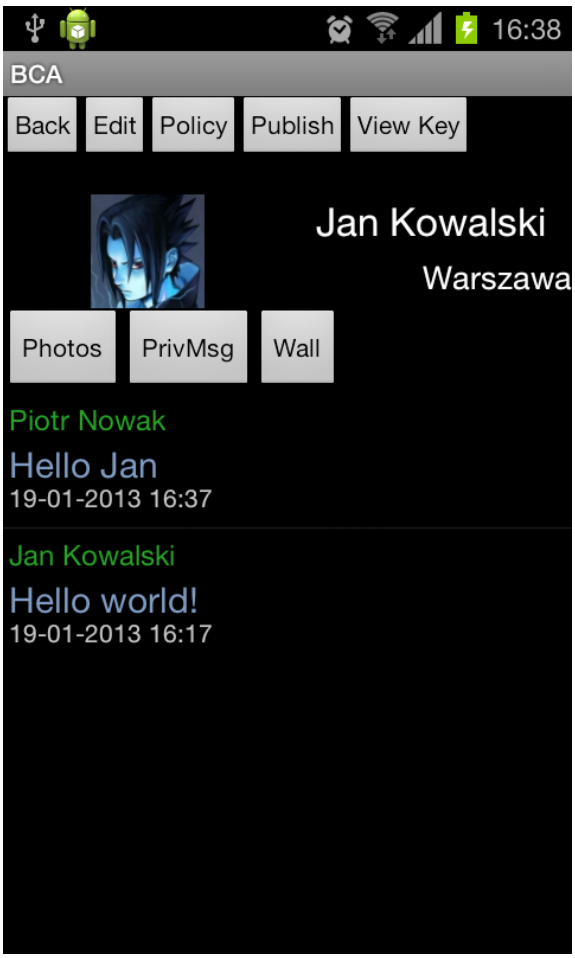
- profil jest poprawnie podpisany cyfrowo (*Signed OK*)
- osoba, której profil przeglądamy nie jest na liście naszych znajomych (*Stranger*)
- osoba, której profil przeglądamy jest podłączona do sieci serwisu (*Online*)

9.1.5 Wysyłanie wiadomości

Będąc na ekranie podglądu cudzego profilu (Rysunek 8.8) możemy opublikować wiadomość na danym profilu za pomocą przycisku *Wall*. Wiadomość doda się do listy publicznych wiadomości na docelowym profilu, oraz u odbiorcy pojawi się komunikat o otrzymanej wiadomości z nazwiskiem nadawcy, co pokazano na Rysunku 8.9.



Rysunek 8.9: Komunikat o otrzymanej wiadomości

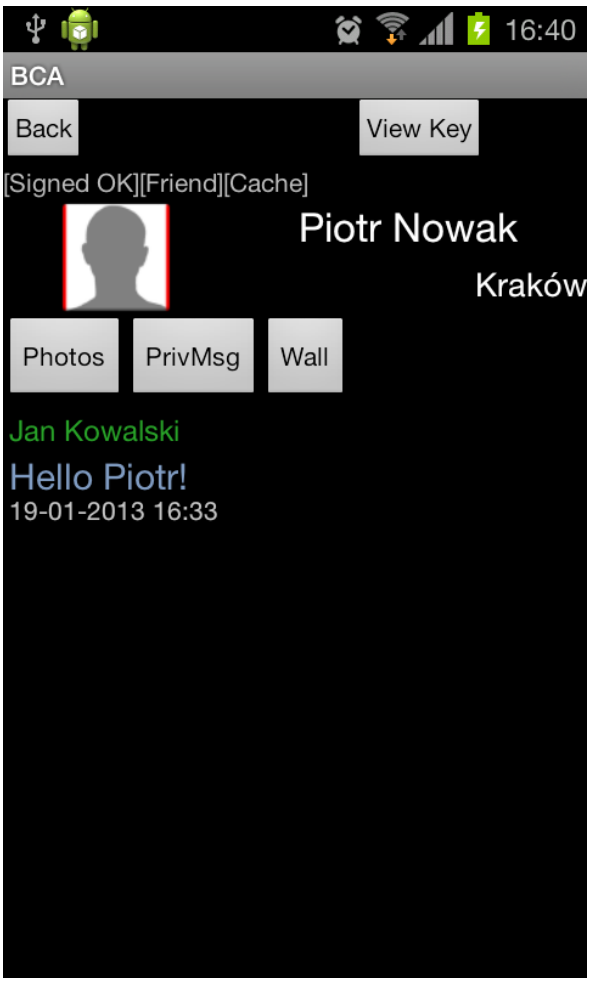


Rysunek 8.10: Otrzymana wiadomość

Odbiorca może odczytać treść wiadomości z powiadomienia lub z okna podglądu swojego profilu (Rysunek 8.10). Wiadomość jest opatrzona imieniem i nazwiskiem nadawcy, datą i godziną wysłania.

9.1.6 Lista znajomych

Przycisk *Friends* znajdujący się na ekranie głównym (Rysunek 8.1) przenosi nas do ekranu z listą znajomych. Wciskając pozycję z tej listy aplikacja przeniesie nas do ekranu podglądu wskazanego profilu. Gdy użytkownik udostępniający dany profil będzie online, wówczas zostanie poproszenie o przesłanie aktualnej wersji profilu. W przypadku gdyby znajomy był offline będzie załadowana wersja jego profilu z pamięci podręcznej aplikacji odzwierciedlająca ostatnio widzianą przez użytkownika wersję profilu, co zostanie zasygnalizowane komunikatem *Cache* na podglądzie profilu (Rysunek 8.11)



Rysunek 8.11: Profil załadowany z pamięci podręcznej

9.1.7 Ustawienia prywatności

Z okna podglądu profilu (Rysunek 8.3) możemy przejść do okna ustawień prywatności (Rysunek 8.12). Są dostępne dwie opcje, których dokładny opis znajdziemy wciskając przyciski *Explain* znajdujące się po lewej stronie opcji.

- *Publish profile on startup* – włączenie tej opcji spowoduje, że przy starcie aplikacji nasz profil będzie publikowany automatycznie
- *Publish also by names* – włączenie tej opcji pozwoli aplikacji publikować przy starcie nasz profil przy użyciu naszego imienia i nazwiska; wyłączenie opcji pozwoli publikować nasz profil wyłącznie przy użyciu naszego klucza publicznego w celu odnalezienia profilu z okna listy znajomych (*Friends*)

Rysunek 8.12: Ustawienia prywatności

9.1.8 Wyłączenie aplikacji i wyłączenie serwisu

Wyjść z aplikacji można wciskając przycisk wstecz (fizyczny przycisk urządzeń opartych o system Android). Warto zwrócić uwagę, że zamykając aplikację w ten sposób pozostawiamy uruchomioną w tle usługę serwisu społecznościowego. Aby zatrzymać tę usługę wciskamy przycisk *Stop*, znajdujący się na ekranie głównym (Rysunek 8.1).

9.2 Zawartość CD

- Kod źródłowy aplikacji
- Kod źródłowy symulatora
- Aplikacja dla platformy Android
- Tekst pracy w formacie PDF