Nikolaos Tsambikakis

Verkon käytön monitorointi ja hallinnointi mobiililaitteille

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

22.11.2015

Metropolia

Tekijä Otsikko Sivumäärä Aika	Nikolaos Tsambikakis Verkon käytön monitorointi ja hallinnointi mobiililaitteille 26 sivua 22.11.2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja	Lehtori Peter Hjort

Insinöörityössä hahmotettiin pintapuolisesti, kuinka paljon tietoverkkoliikennettä muodostuu mobiililaitteella käyttäjältä piilossa eri sovellusten toimesta, ja ohjelmoitiin yksinkertainen sovellus monitoroimaan ja hallitsemaan tätä liikennettä. Alustana käytettiin Android-käyttöjärjestelmää niin tuotetulle sovellukselle kuin analysoidulle verkkoliikenteelle.

Verkkoliikennettä tutkittiin tallentamalla mobiililaitteella siirrettyjä paketteja ja analysoimalla niitä mobiilisovelluksella ja tietokoneohjelmalla. Pakettien tarkastelu osoitti piilossa tapahtuvan verkkoliikenteen olevan suhteellisen yleistä ja näin ollen resursseja vievää. Mobiililaitteelle ohjelmoitiin myös sovellus, jonka ominaisuuksiin kuului verkkoyhteyksien yksinkertainen estäminen ja salliminen, siirrettyjen datamäärien seuraaminen reaaliajassa ja käyttäjältä piilossa suoritettavien prosessien listaaminen.

Tuloksena saatiin yksinkertainen toimiva Android-sovellus, jolla kyetään hallitsemaan verkkoyhteyksiä ja selvemmin valvomaan verkkoliikennettä.

Avainsanat	Android, monitorointi, hallinnointi, mobiililaite, sovellus

Author Title Number of Pages Date	Nikolaos Tsambikakis Monitoring and controlling network usage for mobile devices 26 pages 22 November 2015
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	Software Engineering
Instructor	Peter Hjort, Senior Lecturer

The purpose of this final year project was to study how much network traffic is generated hidden from the user by different applications. In addition, simple software was programmed to monitor and control this traffic. The Android operating system was used as a base both for the software produced and the network traffic analyzed.

Network traffic was examined by saving packets transferred through a mobile device and the packets were analyzed with a mobile and a computer application. Inspection of packets demonstrated hidden network traffic to be relatively common and therefore resource consuming. An application with the ability to simply allow and deny network connections, monitor the amounts of data transferred in real time and list running processes hidden from the user was also programmed for the mobile device.

The result was a simple working Android application with the capability to control network connections and to monitor network traffic more clearly.

Keywords	Android, monitoring, controlling, mobile device, application

Sisällys

Lyhenteet

1 Johdanto 1

2 Verkkoliikenne mobiililaitteella 2

2.1 Yleistä ja ongelmat 2
2.2 Saatavilla olevat työkalut 2

3 Vertailuna verkkoliikenne tietokoneissa 4

3.1 Yleistä 4
3.2 Verkkoliikenteen hallinnointi 4
3.3 Verkkoliikenteen monitorointi 6

4 Taustalla tapahtuva verkkoliikenne käytännössä 8

5 Android alustana 11

5.1 Ohjelmistojen rakenne 11
5.2 Tekninen rakenne 13

6 Mobiilisovelluksen rakenne 15

7 Mobiilisovelluksen toteutus ja kehitys 20

7.1 Kehitysympäristö 20
7.2 Toteutus 20
7.3 Kehitys 23

8 Yhteenveto 25

Lähteet 26

Lyhenteet

ADT	*Android Development Tools*. Eclipse IDE:lle tarkoitettu liitännäinen, jossa on Android-ohjelmointia helpottavia ominaisuuksia.
IDE	*Integrated Development Environment*. Ohjelmistokehitykseen tarkoitettu sovellus, joka sisältää useita ohjelmointia helpottavia työkaluja.
IDS	*Intrusion Detection System.* Järjestelmä, joka havaitsee tietoverkkoteitse tapahtuvan hyökkäyksen kohdekoneelle.
IPS	*Intrusion Prevention System.* Järjestelmä, joka estää tietoverkkoteitse tapahtuvan hyökkäyksen kohdekoneelle.
Java	Oraclen tarjoama ja laajalti käytetty laitteistoriippumaton oliopohjainen ohjelmointikieli.
SDK	*Software Development Kit*. Paketti, joka sisältää tarvittavat työkalut tietylle alustalle tehtävään ohjelmistokehitykseen.
UID	*User Identifier*. Unixin kaltaisten käyttöjärjestelmien käyttäjätunnistukseen käytetty arvo.

1 Johdanto

Erilaisten mobiililaitteiden suosio on viime vuosien aikana kasvanut huomattavasti. Teknologinen kehitys on supistanut älypuhelinten ja tietokoneiden eroa, mikä on monissa kotitalouksissa johtanut tavanomaisen pöytätietokoneen korvaamiseen tabletilla. Jatkuvasti kehittyvät suorituskyvyt ja entistä kattavammat verkkoyhteydet ovat avanneet ovet monipuolisemmille mobiilisovelluksille. Kehityksen seurauksena osa verkkoliikenteestä on siirtynyt käyttäjältä huomaamattomiksi taustaprosesseiksi.

Tiedostamattomalla verkkoliikenteellä saattaa olla haitallisia seurauksia. Taustalla tapahtuvat tiedonsiirrot kuluttavat akkua ja voivat aiheuttaa yllättäviä kustannuksia. Myöskään tietoturvan kannalta piilossa toimivat yhteydet eivät ole hyvä asia.

Insinöörityön tavoitteena on pureutua tähän ongelmaan luomalla yksinkertainen sovellus, jonka avulla käyttäjä kykenisi tarkkailemaan ja hallinnoimaan erityisesti laitteensa taustaprosesseista syntyvää verkkoliikennettä.

2 Verkkoliikenne mobiililaitteella

2.1 Yleistä ja ongelmat

Mobiililaite on laite, jonka käyttö perustuu langattomuuteen. Käyttäjän ei siis tarvitse olla sidoksissa tiettyyn paikkaan käyttääkseen laitteen ominaisuuksia. Yleisimpiä mobiililaitteita ovat älypuhelimet ja tabletit.

Luonnollisesti laitevalmistajia on olemassa useita, ja laitteiden hintatasot ovat hyvinkin vaihtelevia. Kuitenkin käyttöjärjestelmien osalta ovat paikkansa vakiinnuttaneet kolmeksi markkinaosuuksiltaan suurimmaksi avoimeen lähdekoodiin pohjautuva Googlen kehittämä Android, Applen kehittämä IOS ja Microsoftin valmistama Windows Phone. Kullekin käyttöjärjestelmälle on runsaasti sovelluksia, ja lähestulkoon poikkeuksetta sovellukset hyödyntävät tietoverkkoyhteyksiä jollain tavalla. Yleistä on esimerkiksi näyttää käyttäjälle mainoksia sovelluksen käytön yhteydessä. Vaikka itse sovelluksen toiminta ei olisi verkkokeskeistä, on siis kuitenkin mahdollista, että sovelluksen käyttö muodostaa verkkoliikennettä. Joissakin tapauksissa verkkoliikenne muodostuu niin sanotuista taustaprosesseista, jolloin käyttäjä ei edes välttämättä havaitse verkon käyttöä.

Epätoivotusta dataliikenteestä voi koitua useitakin haittoja, joista ilmiselvin on taloudellinen. Vaikka yhä useampi omistaakin matkapuhelinliittymän, jossa on rajaton tiedonsiirto, rajattomuus harvemmin pätee matkustettaessa ulkomaille. Ylimääräisten kustannuksien lisäksi ylimääräinen verkkoliikenne kuluttaa myös akkua, ja näytön ohella mobiiliverkko on mobiililaitteiden suurimpia virrankuluttajia. Lisäksi taustalla tietoverkkoa käyttävä prosessi voi olla tietoturvauhka tai jo toiminnallinen haittaohjelma.

2.2 Saatavilla olevat työkalut

Mobiililiikennettä on siis hyvin tärkeää pystyä hallitsemaan ja monitoroimaan mahdollisimman tarkasti. Dataliikenteen täydellisen sulkemisen sijaan käytännöllisempää on rajoittaa yksittäisten sovellusten pääsyä internetiin. Nykyään muun muassa Androidille suunnattu Googlen sovelluskauppa tarjoaa useitakin sovelluksia verkkoliikenteen tarkasteluun. Muutama vuosi sitten tilanne oli huonompi. Hyvinä esimerkkisovelluksina voidaan mainita Network Log, Network Connections ja Network monitor. Niiden omi-

naisuudet eroavat hieman toisistaan, mutta kaikilla pystytään seuraamaan reaaliajassa laitteen eri sovellusten aiheuttamaa verkkoliikennettä. Tämän lisäksi Googlen sovelluskaupassa on muutama pakettikohtaiseen verkkoliikenteen analysointiin tarkoitettu sovellus, joista voidaan mainita esimerkkinä tPacketCapture- ja Packet Capture-sovellukset. Eroja sovellusten välillä on tuetuissa Android-versioissa ja siinä, onko sovellus kokonaan ilmainen. Huomion arvoista on, ettei mikään näistä esimerkeistä kykene varsinaisesti verkon hallintaan. Yhteyksien estämiseen tarvitaan erillinen palomuurisovellus. Verkon valvontaan ja hallintaan tarkoitettujen sovellusten ominaisuudet yhdistäviä ratkaisuja ei siis juuri ole.

3 Vertailuna verkkoliikenne tietokoneissa

3.1 Yleistä

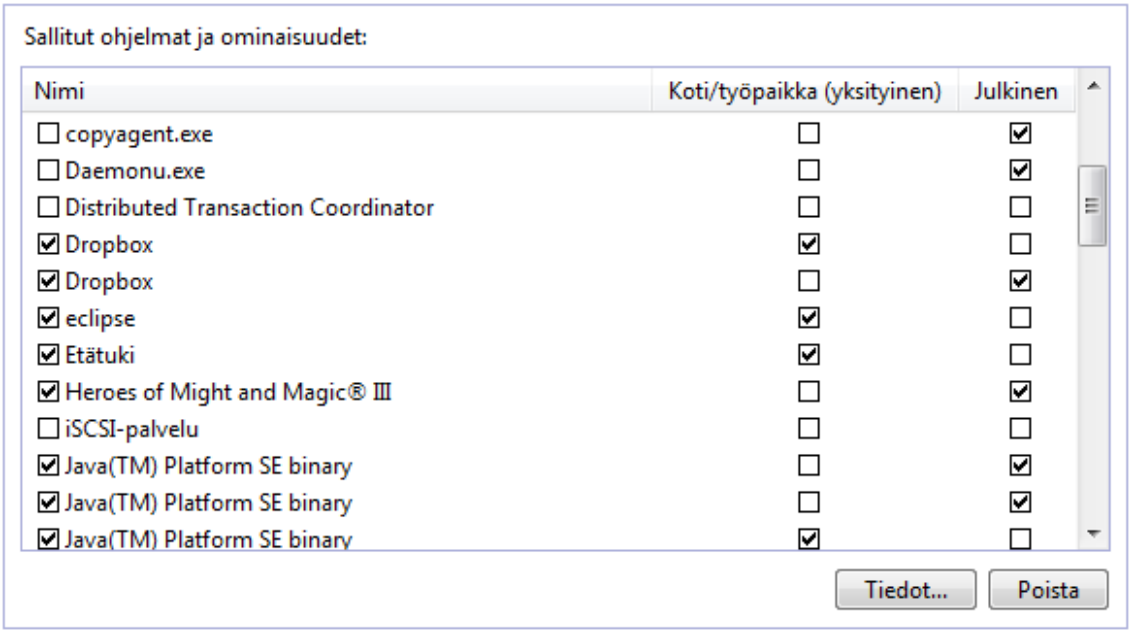
Niin sanottujen tavallisten tietokoneiden historia käsittää nykyään jo useita kymmeniä vuosia. On selvää, että näiden laitteiden osalta on vuosien kuluessa kehittynyt hyväksi katsottuja standardeja ja normeja. Mobiililaitteiden verkkoliikenteen monipuolisuus on täysin verrattavissa nykyajan PC-maailmaan, mutta käytäntöjen osalta ollaan mobiililaitteiden kohdalla vielä jäljessä. Molempien laitekantojen tarpeet verkon monitoroinnille ja hallinnoinnille ovat kuitenkin yhtäläiset. Pääpiirteittäin kyse on tietoturvasta ja toimivuuden ylläpitämisestä. Lähestymistavat kuitenkin näiden kahden seikan hoitamiseen eri alustoilla eroavat huomattavasti. Merkittävimpänä määrittäjänä ohjelmistojen ominaisuuksille ja käytettävyydelle voidaan pitää laitekannan käyttäjien yleistä taitotasoa. Esimerkiksi pöytätietokoneiden käyttäjät saatetaan mieltää keskimäärin kokeneemmiksi verrattuna älypuhelimien käyttäjiin. Tämä asettaa tiettyjä vaatimuksia ohjelmistolle.

Tätä nykyä voidaan pitää jo standardina, että tavallisesta tietokoneessa on jonkinlainen palomuuri. Useat käyttöjärjestelmät sisältävät palomuurin vakiona, ja lukuisat eri valmistajat tarjoavat niitä joko ilmaiseksi tai maksua vastaan. Palomuurien ominaisuudet vaihtelevat yksinkertaisesta verkkoyhteyksien hallinnoinnista aina edistyneempään sovelluskohtaiseen verkkoyhteyksien monitorointiin. Palomuurien lisäksi on tarjolla myös analyyttisempaan monitorointiin tarkoitettuja sovelluksia verkkoyhteyksien aikana siirtyvien pakettien yksityiskohtaiseen tarkasteluun. Erityisesti tarkemman monitoroinnin mahdollistavat sovellukset eivät ole vielä vakinaistaneet paikkaansa mobiilimaailmassa, vaikka tarvetta tämänkaltaisillekin ratkaisuille nykyään esiintyy.

3.2 Verkkoliikenteen hallinnointi

Yleisimmillään verkkoliikennettä voidaan siis kontrolloida palomuurein. Palomuurien perusteena on joko yhteyksien salliminen tai estäminen. Yksinkertaisimmillaan tämä tarkoittaa vaikkapa vain tietyn ohjelman verkkokäyttäytymisen hallinnointia. Usein kuitenkin voidaan palomuureista määrittää myös joko tiettyjen osoitteiden, porttien tai protokollien suhteen suoritettavia toimintoja. Esimerkiksi tiettyyn suuntaan kulkeva liikenne

voidaan estää tiettyyn porttiin, kun käytössä on tietty ohjelma. Lisäksi palomuurit sisältävät nykyään usein hyökkäyksen estoon (IPS) ja hyökkäyksen havaitsemiseen (IDS) käytettäviä ominaisuuksia. Verrattuna PC-maailmaan palomuurit voivat olla myös erillisiä fyysisiä yksiköitä laskentatehollisista syistä, kun taas mobiililaitteiden kohdalla tämä ei luonnollisestikaan ole käytännöllistä. Kuvassa 1 on esimerkki yksinkertaisesta palomuurista.



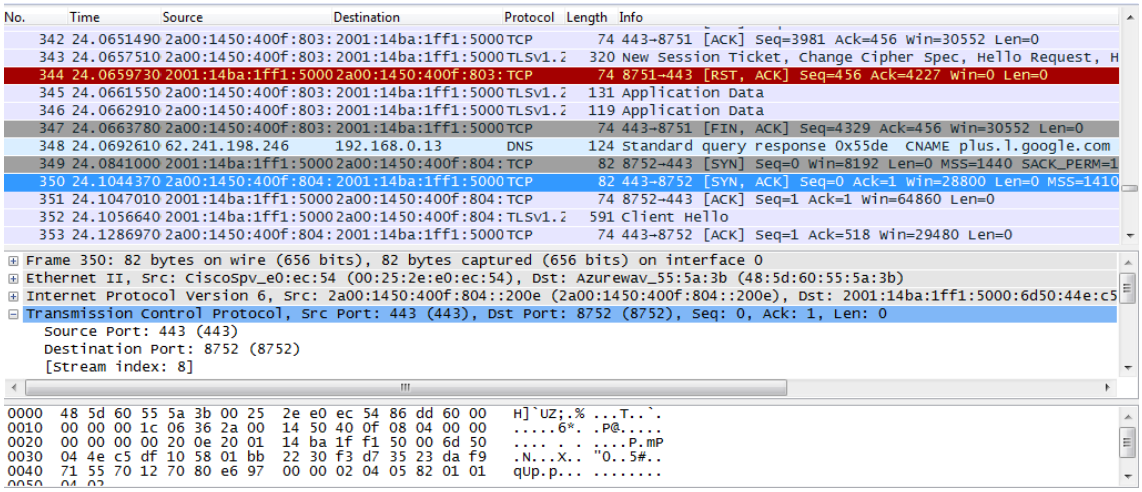
Kuva 1. Windows-käyttöjärjestelmän oman palomuurin asetusikkuna eri ohjelmien verkkoliikenteen sallimiseen tai estämiseen.

IPS ja IDS

Verkkoliikenteen hallinnointiin ja monitorointiin on myös automatisoituja ratkaisuja. IPS (Intrusion Prevention System) ja IDS (Intrusion Detection System) on molemmat kehitetty automaattiseen verkkoliikenteen analysointiin. Datapaketteja tarkistamalla nämä järjestelmät kykenevät etsimään siirretystä datasta hyökkäykseen viittaavia kaavoja. Hyökkäyksen havaitessaan kumpikin järjestelmä ilmoittaa tapahtumasta käyttäjälle. Toisin kuin IDS, IPS yrittää ilmoittamisen lisäksi myös estää meneillään olevan hyökkäyksen. Tämä tarkoittaa, että IPS kykenee reaaliajassa muuttamaan verkkoliikenteelle määriteltyjä sääntöjä. Esimerkiksi aiemmin avoinna ollut portti saatetaan hyökkäyksen estämiseksi sulkea.

3.3 Verkkoliikenteen monitorointi

Edistyneempien palomuurien ohella tietokoneiden tietoverkkoliikennettä voidaan seurata varta vasten tarkoitukseen kehitetyillä ohjelmilla. Ohjelmien luonne vaihtelee verkon yleisen liikenteen ja kuormituksen valvonnasta tarkempaan datapakettikohtaiseen analyysiin. Mobiilikäyttäjälle näistä tärkeimpänä voidaan pitää yleistä verkkoliikenteen valvontaa, mutta yhtä lailla mahdollisuus datapakettikohtaiseen tarkasteluun on tärkeää. Yleinen verkkoliikenteen monitorointi käsittää tiedon avoinna olevista yhteyksistä ja niihin sidoksissa olevista ohjelmista ja porteista. Tämän lisäksi voidaan näyttää informaatiota yhteyden aikana siirretyistä datamääristä ja nopeuksista. Kun halutaan tarkempaa tietoa yhteyden sisällöstä, tutkinnan kohteeksi otetaan siirretyt datapaketit. Lyhyidenkin datayhteyksien aikana saattaa datapaketteja liikkua koneiden välillä useita satoja. Näitä datapaketteja voidaan avata ja niiden sisältämät kerrokset lajitella käyttäjälle ymmärrettävämpään muotoon. Esimerkiksi koneiden välillä lähetetystä TCP-paketista saadaan esitettyä yksityiskohtaisemmat tiedot vaikkapa siirtokerrokseen liittyen. Ohjelmistoja tietoliikenneyhteyksien monitorointiin on useita, mutta yksi käytetyimmistä lienee Wireshark (kuva 2).



Kuva 2. Wireshark-ohjelman pääikkuna. Luettelossa näkyvät lisätiedot eräästä TCP-paketista.

Wireshark

Vuonna 1998 alkunsa saanut Wireshark-monitorointiohjelma sisältää kattavat työkalut nykyaikaiseen verkkoliikenteen analysointiin. Ohjelma mahdollistaa datapakettien listaamisen niin kaapeliverkoista kuin langattomasti. Kummassakin tapauksessa vas-

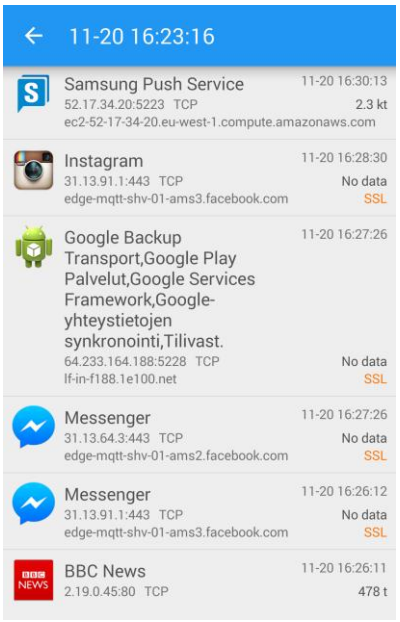
taanotettuja datapaketteja voidaan suodattaa ja niiden sisältöä tutkia tarkemmin. Sisältö jakautuu kyseessä olevan protokollan mukaisesti kerroksiin, joista yksityiskohdat ovat laajennettavissa näkymään selkokielisenä ja bittitasolla. Wiresharkissa on myös tuki erilaisten salausten purkuun ja listattujen datapakettien tallentamiseen. Mahdollisuus tallentaa osoittautuu erityisesti silloin käytännölliseksi, kun laite, josta informaatiota halutaan, ei tue Wiresharkia. [1.]

Wireshark ei ole saatavilla Android-alustalle ainakaan virallisesti, mutta vastaavanlaisia ominaisuuksiltaan karsitumpia vaihtoehtoja on. Nämä sovellukset itsessään eivät mahdollista poimittujen pakettien kovinkaan syvällistä tutkimista, mutta pystyvät tallentamaan siirretyn datan Wiresharkin tukemaan muotoon. Näin ollen datapakettivirrat voidaan tallentaa mobiililaitteella ja siirtää jälkikäteen analysoitavaksi erilliselle tietokoneelle.

4 Taustalla tapahtuva verkkoliikenne käytännössä

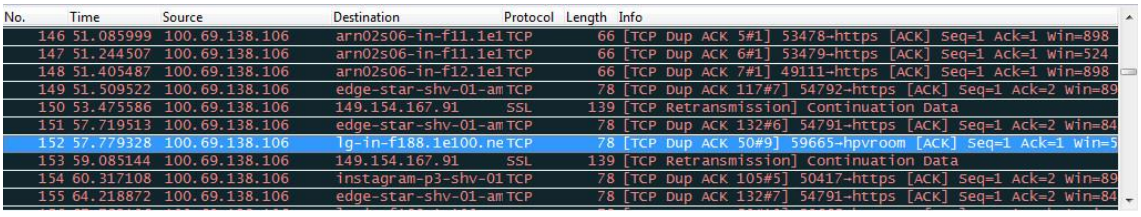
Taustalla tapahtuvan verkkoliikenteen muodostaman ongelman havainnollistamiseksi insinöörityössä päätettiin testata, kuinka paljon todellisuudessa taustalla tapahtuvaa tiedonsiirtoa esiintyy ja minkätyyppisistä yhteyksistä on kyse. Android-pohjaiseen puhelimeen asennettiin Googlen sovelluskaupasta tPacketCapture- ja Packet Capture-sovellukset. Muitakin tapoja seurata datapakettiliikennettä on, mutta nämä sovellukset eivät vaadi pääkäyttäjän oikeuksia, mikä tässä tapauksessa oli välttämätöntä. Tarve korotetuille oikeuksille on pystytty poistamaan kummastakin sovelluksesta reitittämällä kaikki puhelimen verkkoyhteydet virtuaalisen tunnelin läpi sovelluksen käytön ajaksi. Näin ollen sovellus saa oikeudet tarkastella kaikkea tunnelin läpi kulkevaa liikennettä.

Testi suoritettiin tallentamalla verkkoliikennettä kymmenen minuuttia puhelimen ollessa kotinäytöllä ilman vuorovaikutusta sallien kuitenkin näytön normaalin virran aikakatkaisun. Toimenpide suoritettiin molemmilla monitorointisovelluksilla erikseen, sillä teknisten rajoitusten vuoksi tunnelointia pystyy hyödyntämään vain yksi sovellus kerrallaan. Packet Capture -sovelluksella saaduista tuloksista huomattiin, että yhteyksiä oli muodostunut testin aikana 18. Osa näistä yhteyksistä oli samojen sovellusten muodostamia ja osa liittyi käyttöjärjestelmän toimintaan. Tästä huolimatta joukosta erottui selvästi muutama yhteys, joiden hyöty käyttäjälle on kyseenalainen. Englantilaisen uutistoimisto BBC:n ja sosiaalisen median palvelua tarjoavan Instagramin sovellukset ottivat yhteyttä testin aikana yllättävän usein tarjoamatta minkäännäköistä ilmoitusta käyttäjälle (kuva 3). Toki osa sovellusten toiminnallisuudesta perustuu käyttäjältä huomaamattomiin prosesseihin, mutta yhteydenottojen lyhyttä aikaväliä voidaan tässä tapauksessa pitää hieman kohtuuttomana.



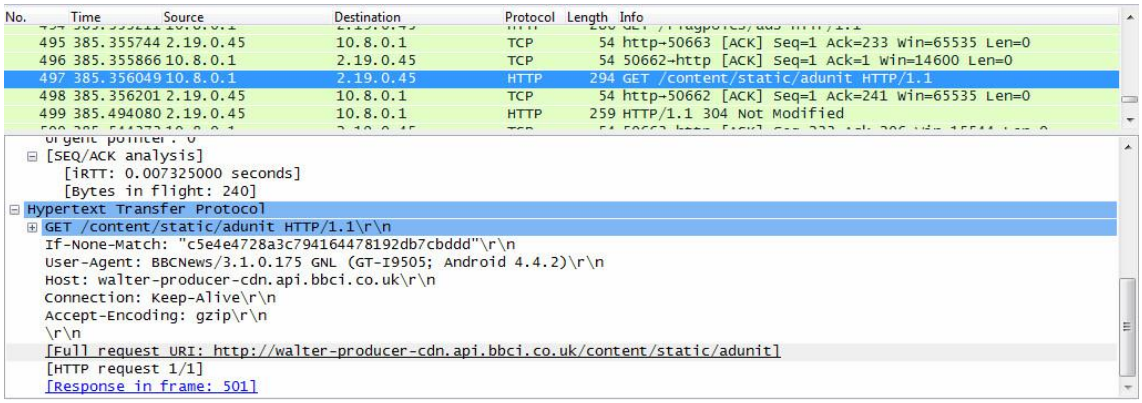
Kuva 3. Osa Packet Capture -sovelluksella tallennetuista yhteyksistä.

Testin toinen sovellus eli tPacketCapture ei sisällä minkääntyyppistä tulosten tarkasteluominaisuutta, vaan sovellus perustuu vain verkkoliikenteen tallentamiseen. Ohjelmalla voidaan kuitenkin tästä huolimatta tallentaa Wiresharkin tukemaan ”.pcap”-tiedostomuotoon, mikä mahdollistaa pakettien yksityiskohtaisemman tarkastelun tietokoneella. Tarkempi analyysi vahvisti jo aiemmin todettua seikkaa, että yhteyksiä on testin aikaväliin verrattuna yllättävän paljon. Tosin Wiresharkin avulla huomattiin, että osa aiemmalla sovelluksella kaapatuista paketeista saattaa olla uudelleenlähetyksiä aiemmin epäonnistuneille yhteyksille eikä niinkään yksilöllisiä kokonaan uusia yhteyksiä. Tähän viittaisi suuri määrä Wiresharkin merkitsemiä duplikaatti-ACK-paketteja. Tavallisen yhteydenmuodostuksen aikana yhteyden hyväksyviä pelkkiä Acknowledgement-paketteja eli ACK-paketteja siirtyy vain yksi. Testin aikana osissa yhteyksiä ACK-paketteja siirtyi jopa yhdeksän (kuva 4). Kaikki mobiilisovelluksille rekisteröidyt yhteydet eivät siis välttämättä ole sovellukselle suunniteltuja ominaisuuksia.



Kuva 4. Luettelo useasta duplikaatti-ACK-paketista.

Wiresharkista saadun informaation perusteella pystyttiin tekemään myös tarkempia johtopäätöksiä joidenkin yhteyksien luonteesta. Osa yhteyksistä sisälsi HTTP-protokollan mukaisia datapaketteja, joiden metodina oli GET. GET-metodi mahdollistaa tietojen hakemisen HTTP-palvelimelta sille annettujen parametrien mukaisesti. Tässä tapauksessa parametreista pystyttiin osittain päättelemään, mitä varten yhteydet olivat. Esimerkiksi erään datapaketin parametrina oli ”/content/static/adunit”, mikä viittaa siihen, että kyseisestä osoitteesta haetaan sisältöä sovellukseen (kuva 5). Osoitteen avaaminen internetselaimella toi listan eri uutisten aihealueista ja niiden tunnisteet. Sovellus siis mahdollisesti päivittää taustalla tietyin väliajoin uutislistaansa.



Kuva 5. Esimerkki GET-metodin sisältävästä HTTP-paketista.

Testissä käytetyssä puhelimessa ei ollut juurikaan asennettuja sovelluksia, mutta sovellusmäärien kasvaessa on todennäköistä, että myös taustalla tapahtuvan verkkoliikenteen määrä kasvaa. Turhat yhteydet saattavat muodostaa siis yllättävänkin suuren rasitteen mobiililaitteelle.

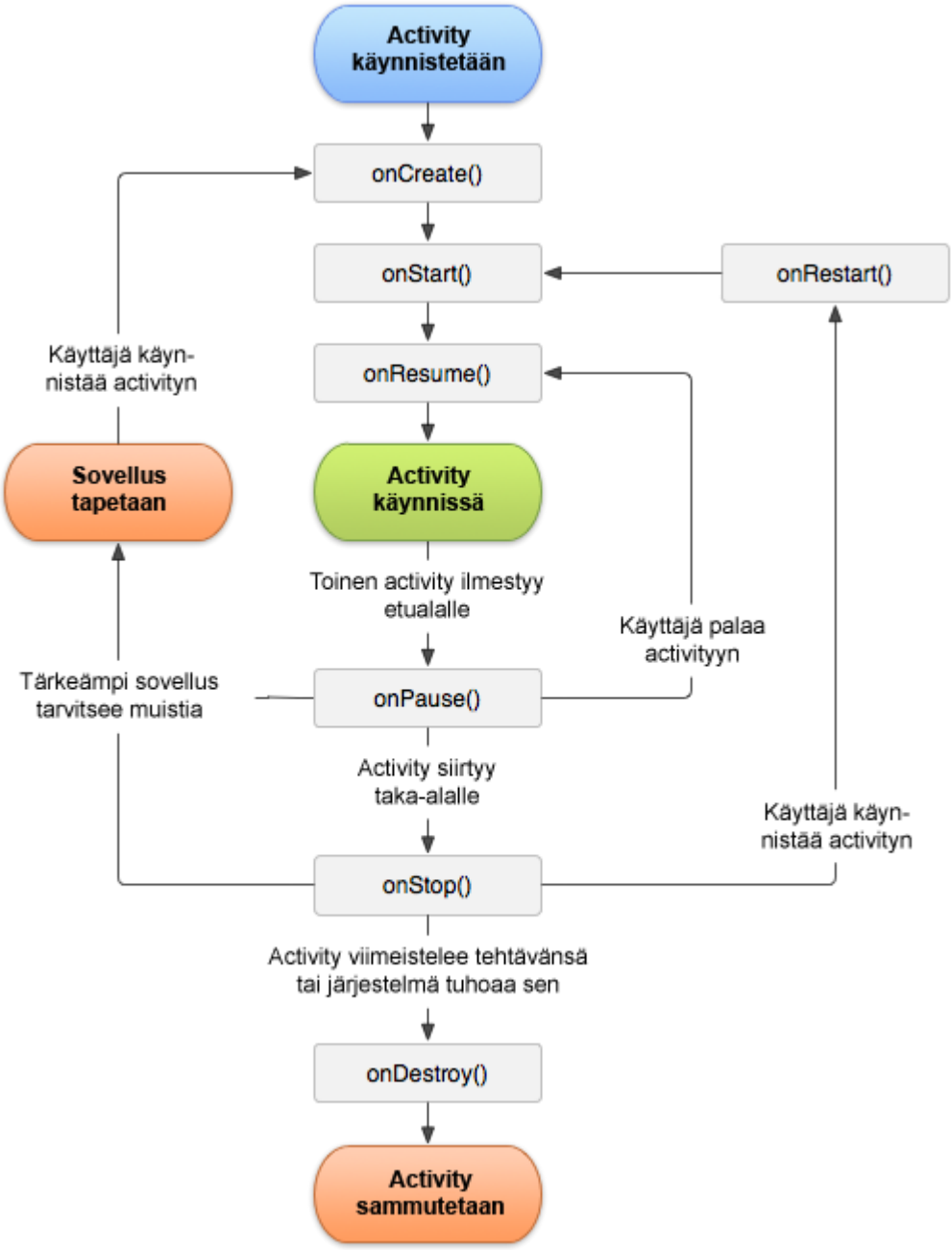
5 Android alustana

Android on Linux-ydintä käyttävä, vuonna 2007 julkaistu avoimeen lähdekoodiin perustuva käyttöjärjestelmä mobiililaitteille. Se on hyvin laajalti käytetty, ja kesällä 2013 Androidin markkinaosuuden arvioitiinkin olevan noin 70 %. Linux-ytimen lisäksi Android koostuu muun muassa eri järjestelmäkirjastoista ja Java-sovellusten suorittamiseen tarkoitetusta Dalvik-virtuaalikoneesta. [2.]

5.1 Ohjelmistojen rakenne

Android-sovelluksia suoritetaan Dalvik-virtuaalikoneella. Dalvik vastaa Javan omaa virtuaalikonetta, mutta on suunniteltu suorituskyvyltään rajoitetummille alustoille. Koska Dalvik on Java-virtuaalikone, kirjoitetaan myös Android-sovellukset useimmiten Java-ohjelmointikielellä. Androidin käyttämä Java kuitenkin eroaa hieman Oraclen Javasta tarjolla olevien kirjastojen vuoksi. Niiden syntaksi on tästä huolimatta identtinen.

Eroja on Android-maailmassa myös tavassa käsitellä sovellusten elämänkaarta ja prosesseja. Vastoin pöytätietokoneista opittua ajattelutapaa sovelluksia ei juuri koskaan erikseen suljeta. Sovelluksista poistuttaessa ne pysäytetään ja siirretään taka-alalle odottamaan. Myöhemmin samoihin sovelluksiin voidaan palata tuomalla ne etualalle, jolloin suoritus jatkuu määritetystä kohdasta (kuva 6). Mikäli laitteen resurssit eivät kuitenkaan riitä tärkeämpien tehtävien suorittamiseen, saatetaan nämä taka-alalla olevat prosessit poistaa.



Kuva 6. Android-sovelluksen Activity-komponentin elämänkaari [3].

Sovellukset voidaan rakentaa neljästä erityyppisestä komponentista: Activityistä, Serviceistä, ContentProviderista ja BroadcastReceiveristä. Näistä Activity on yleisimmin käytetty, ja niitä voi yhdessä sovelluksessa olla useita. Jokainen Activity hoitaa yhden tehtävän, ja esimerkiksi sähköpostisovelluksessa sähköpostien listaus ja uuden viestin kirjoittaminen voivat olla omia Activityjä. Service puolestaan toimii tavanomaisen palvelun tavoin tarjoten mahdollisuuden suorittaa pitkäaikaisia tehtäviä tausta-ajona. Service ei tarjoa käyttöliittymää, mutta vuorovaikuttaminen on mahdollista toisen komponentin

avulla. Musiikin taustasoitto on hyvä esimerkki Servicen toiminnasta. ContentProvider ja BroadcastReceiver ovat yksinkertaisissa sovelluksissa ehkä hieman harvemmin käytettyjä. ContentProvider tarjoaa nimensä mukaisesti mahdollisuuden eri sovelluksille jakaa tietoja keskenään. Android-järjestelmästä pystytään esimerkiksi kysymään vaikkapa käyttäjän yhteystietoja järjestelmän oman ContentProviderin avulla, jos toimenpiteeseen on saatu riittävät oikeudet. BroadcastReceiver sen sijaan mahdollistaa järjestelmän laajuisten viestien vastaanottamisen. Näitä viestejä voivat olla vaikka näytön sammuminen tai akun varauksen muutos. [5.]

Turvallisuuden lisäämiseksi Android-sovellukset suoritetaan aina niin sanotuissa hiekkalaatikoissa. Tämä estää sovellusten suoran pääsyn järjestelmän kriittisiin tiedostoihin, mikä suojaa järjestelmää mahdollisilta haitallisilta operaatioilta. Ennen Androidin 4.3-versiota hiekkalaatikot toteutettiin antamalla jokaiselle sovellukselle asennuksen aikana yksilöllinen Linux UID, jonka mukaan sovelluksen käyttöoikeudet määräytyivät. Uusimmissa versioissa Androidin ytimeen on liitetty tarkempien käyttöoikeuksien hallinnan mahdollistava SELinux. [6.]

Vaikka käyttöoikeuksien rajaaminen lisää turvallisuutta, se myös joissain tapauksissa rajoittaa toiminnallisuutta. Androidin alla piilevän Linux-järjestelmän kaikkia ominaisuuksia ei voida hyödyntää ilman pääkäyttäjän oikeuksia. Pääkäyttäjän oikeudet on mahdollista saada suorittamalla toimenpide, jota kutsutaan *roottaamiseksi*. Tällöin järjestelmässä esiintyvää ohjelmointivirhettä hyväksikäyttäen saadaan rajatut oikeudet korotettua pysyvästi pääkäyttäjän oikeuksiksi.

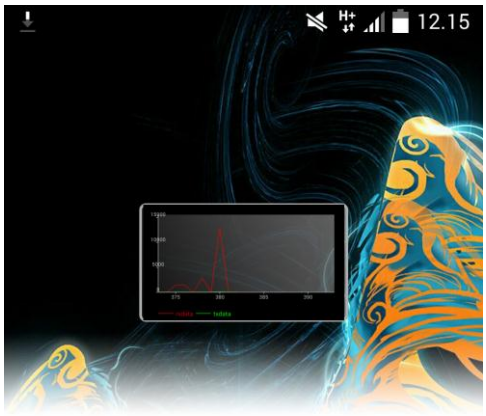
5.2 Tekninen rakenne

Laitteiden vaihtelevan hintatason vuoksi myös niiden tarjoamat tekniset ominaisuudet eroavat toisistaan. Tästä huolimatta Android käyttöjärjestelmänä tukee useita eri teknologioita. Tuki on muun muassa viestintäteknologioista toisen sukupolven GSM:lle, kolmannen sukupolven UMTS:lle ja paikallisempaan tiedonsiirtoon tarkoitetuille Bluetoothille ja Wi-Fi:lle. Lisälaitteistojen osalta Android tukee esimerkiksi suurta määrää eri mittareita ja antureita. [2.] Mobiililaitteiden ohella Android on laajentanut tuettujen laitteidensa valikoimaa nykyisin myös televisioihin, kelloihin ja autoihin.

Android-käyttöjärjestelmä toimii 32-bittisillä ja 64-bittisillä suorittimilla. Tuettuja prosessoriarkkitehtuureja ovat ARM:n, Intelin ja MIPS:n suorittimet [4]. Näistä ARM:n prosessorit ovat ehkä yleisimmin käytetyt. Android-laitteiden suorittimet ovat yleensä moniytimisiä ja ovat tällä hetkellä laskentatehollisesti 1–2 GHz:n luokkaa. Keskusmuistia laitteet voivat sisältää mallista riippuen 1–4 Gt. Mobiililaitteissa on usein myös erillinen grafiikkapiiri.

6 Mobiilisovelluksen rakenne

Insinöörityönä ohjelmoitu sovellus koostuu kahdesta osasta: pienoisohjelmasta eli *widgetistä* (kuva 7) ja pääohjelmasta. Pienoisohjelma on laitteen kotinäytölle sijoitettava kuvake, joka tässä tapauksessa näyttää kuvaajana reaaliajassa laitteen kokonaistiedonsiirron sekunnin välein. Kuvaaja koostuu kahdesta käyrästä, joista punainen kuvaa vastaanotettua dataa ja vihreä lähetettyä.

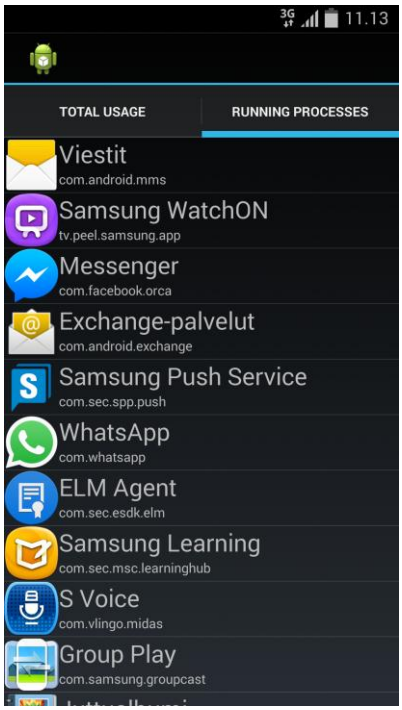


Kuva 7. Widget eli pienoisohjelma kotinäytöllä.

Pääohjelman käyttöliittymä suunniteltiin yksikertaiseksi, mutta tarjoamaan mahdollisimman paljon relevanttia informaatiota. Sovelluksen avatessaan käyttäjä näkee ensimmäiseksi laitteensa kokonaistiedonsiirron selvänä kuvaajana. Kuvaajan alla näkyvät numeerisina arvoina sekunnin välein tapahtuva tiedonsiirto sekä laitteen käynnistyksen jälkeen siirretty data yhteensä. Molemmat arvot on esitetty tavuina ja värikoodattu vastaamaan lähetettyä ja vastaanotettua dataa (kuva 8).

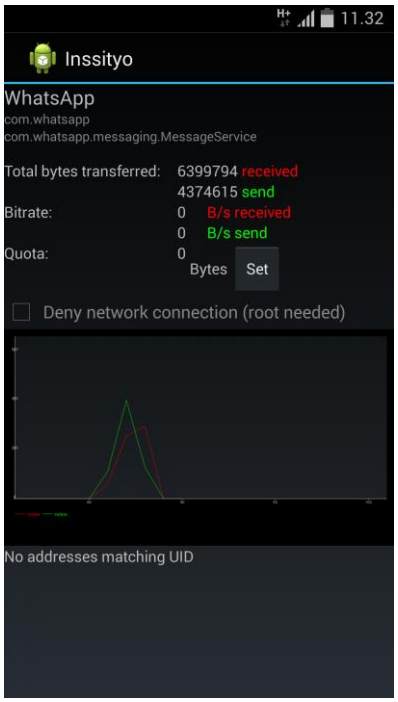
Kuva 8. Ohjelmoidun sovelluksen alkunäyttö.

Alkunäytön ylälaidasta voidaan siirtyä käynnissä olevien prosessien listaukseen (kuva 9). Lista näyttää kyseessä olevan sovelluksen nimen, kuvakkeen ja paketin, johon sovellus kuuluu.



Kuva 9. Lista prosesseista.

Listan kukin kohta johtaa prosessin tiedot yksityiskohtaisemmin esittävään näkymään. Tämän näkymän ylälaidassa ovat nimen ja paketin lisäksi myös pakettiin kuuluvat palvelut, mikäli niitä on. Niiden alta selviää prosessin siirtämä data yhteensä sekä sekunnin välein tapahtuva tiedonsiirto tavuina. Kullekin prosessille voidaan asettaa raja vastaanotetun datan määrälle. Rajan ylittyessä sovellus ilmoittaa asiasta laitteen yläpalkkiin ilmestyvällä *notificationilla* eli huomautuksella. Mikäli laitteella on pääkäyttäjän oikeudet, voi käyttäjä halutessaan estää myös kokonaan jonkin prosessin tiedonsiirron rastittamalla käyttökiintiön alla olevan ruudun. Yksityiskohtaisemman näkymän alalaidassa on lisäksi kuvaaja havainnollistamaan kyseessä olevan prosessin verkkoaktiivisuutta sekä lista prosessin UID:hen yhdistetyistä verkko-osoitteista.

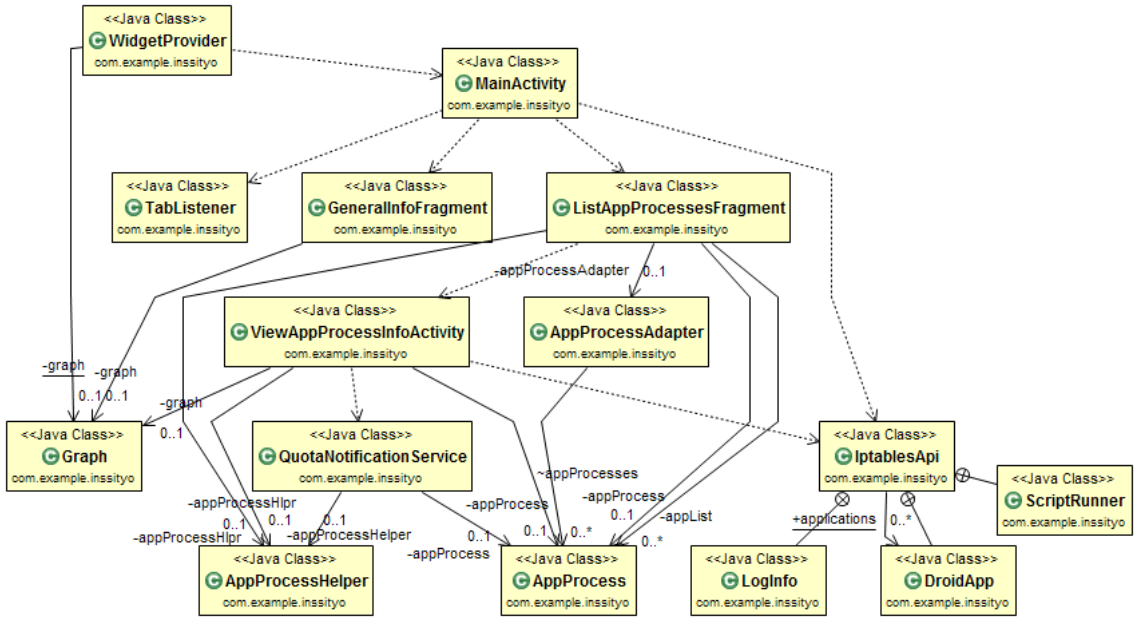


Kuva 10. Prosessin yksityiskohtaisempi näkymä.

Laitteen valikkonapin painaminen sovelluksessa tuo esiin kaksi vaihtoehtoa. Vaihtoehtojen avulla voidaan nollata joko estettyjen prosessien lista tai prosesseille asetetut käyttökiintiöt.

Koodin rakenne

Insinöörityönä toteutetussa ohjelmassa on Java-luokkia yhteensä 15 ja ulkoasua määritteleviä XML-tiedostoja seitsemän (kuva 11). Näistä seitsemästä kaksi määrittelee widgetin ja käyttökiintiön asettavan dialogin ulkoasun ja loput luokkien ulkoasun. Projektissa on myös muutama eri arvoja määrittelevä XML-tiedosto, joskin enemmistö esimerkiksi käyttäjälle näkyvistä teksteistä on kirjoitettu suoraan koodiin. Koska sovelluksen käyttöliittymässä ei esiinny kuvatiedostoja, ei myöskään eri näyttöjen resoluutioeroja ole otettu huomioon.



Kuva 11. Ohjelmoidun sovelluksen luokkakaavio.

7 Mobiilisovelluksen toteutus ja kehitys

7.1 Kehitysympäristö

Ohjelmoidun mobiilisovelluksen työstäminen Android-alustalle vaatii virallisen ohjelmointikirjaston eli SDK:n ja jonkin ohjelmointiympäristön eli IDE:n. Mobiilisovellus toteutettiin käyttäen ilmaista Eclipse-ohjelmointiympäristöä, johon on tarjolla liitännäisenä viralliset Android-sovelluskehitykseen tarkoitetut työkalut (ADT). Liitännäinen mahdollistaa muun muassa visuaalisen käyttöliittymäsuunnittelun ja sovelluksen testaamisen emulaattorin avulla. Tästä huolimatta mobiilisovelluksen testaaminen tehtiin käyttäen kahta Samsungin valmistamaa puhelinta: Galaxy S 1:tä ja Galaxy S 4:ää. Näistä testipuhelimista S 1:ssä on pääkäyttäjän oikeudet ja Androidin versiona 2.3.3, kun taas S 4:ssä on vakio-oikeudet ja versiona 4.4.2.

Jotta kehitystyö onnistuisi vaivattomasti usealta koneelta, päädyttiin projektin aikana käyttämään versionhallintaa. Näin pidettiin samalla huolta myös varmuuskopioinnista. Versionhallintaohjelmistoksi valittiin avoimen lähdekoodin Git, ja palveluntarjoajana käytettiin ilmaista BitBucketia [7]. Sovelluksen koodi kommitoitiin säännöllisesti aina uuden ominaisuuden valmistuttua, ja muutokset kirjattiin kehityspäiväkirjana käytettyyn Blogger-palvelun tarjoamaan blogiin.

7.2 Toteutus

Sovelluksien listaus

Ensisijaisesti lähdettiin tutkimaan, kuinka eri sovellukset voidaan listata ottaen huomioon niiden elämänkaaren mukaiset tilat. Päädyttiin käyttämään Androidin ohjelmointikirjaston tarjoamaa metodia käynnissä olevien palveluiden listaamiseen. Sovelluksen kehityksen edetessä tämä todettiin kuitenkin riittämättömäksi ja listaus päätettiin toteuttaa käyttäen vastaavaa sovellusprosessit palauttavaa metodia. Näin saatiin listattua kaikki verkkoliikennettä aiheuttavat sovelluskomponentit ja pystyttiin suodattamaan lista elämänkaaren mukaista tilaa määrittelevien Importance-tasojen perusteella. Merkityksellisiä Importance-tasoja on yhteensä kuusi:

- etualalla oleva prosessi (IMPORTANCE_FOREGROUND)

- havaittavissa oleva prosessi (IMPORTANCE_PERCEPTIBLE)
- ruudulla näkyvä prosessi (IMPORTANCE_VISIBLE)
- palvelu (IMPORTANCE_SERVICE)
- taustaprosessi (IMPORTANCE_BACKGROUND)
- tyhjä prosessi (IMPORTANCE_EMPTY).

Sovelluslistaan päätettiin sisällyttää tasot palveluun asti. [5; 8.]

Prosessin yksityiskohtaisempi näkymä

Yksityiskohtaisempia tietoja prosesseista pystyttiin keräämään käyttäen hyväksi Androidin ohjelmointikirjaston metodeja ja hyödyntämällä Linux-käyttöjärjestelmässä vakiona esiintyviä ominaisuuksia. Prosessin UID-arvoa vastaan saatiin käyttöjärjestelmän ”/proc/net/tcp”-tiedostosta luettua prosessiin kytköksissä olevia lokaaleita verkko-osoitteita. Osoitteet esiintyvät alkuperäisessä tiedostossa kuitenkin heksadesimaali-muodossa, joten sovellukseen oli ohjelmoitava käännösalgoritmi, jonka avulla arvot saatiin merkkijonoiksi. Ohjelmointikirjaston valmiilla metodeilla saadut arvot voitiin puolestaan käyttää sellaisinaan. Siirretyn tavumäärän lisäksi metodeista saatiin muuttujat säännöllisesti päivittyvälle siirtonopeuden kuvaajalle. Varsinainen kuvaaja toteutettiin käyttäen AChartEngine-nimistä Androidille tarkoitettua avoimen lähdekoodin diagrammi kirjastoa.

Avoimen lähdekoodin projektiin perustuu myös prosessin verkkoliikenteen kokonaan estävän ominaisuuden koodi. Drdoidwall-nimellä kulkeva projekti on Androidille suunniteltu käyttöliittymä Linux-käyttöjärjestelmästä vakiona löytyvään *Iptables*-palomuuriin. Mobiilisovellukseen muokattiin projektista käyttöön root-käyttöoikeuksien tarkistus ja itse Iptables-tiedoston käsittelyyn tarkoitetut poisto- ja lisäysmetodit. Tiedot estetyistä prosesseista tallennettiin käyttäen ohjelmointikirjaston SharedPreferences-luokkaa. Samaista luokkaa käytettiin läpi sovelluksen myös muiden samankaltaisten pitkäikäisten tietojen tallentamiseen.

Sovelluksen käyttökiintiöiden valvonta toteutettiin palvelun avulla. Palvelu käynnistyy, kun jollekin prosessille annetaan käyttökatto, ja sammuu, kun viimeisin käyttökatto on joko saavutettu tai poistettu manuaalisesti. Käyttökiintiöitä voi siis olla samaan aikaan

useille eri sovelluksille, mutta palveluita käynnistetään tästä huolimatta aina vain yksi. Prosessin käyttökiintiön voi poistaa manuaalisesti syöttämällä sen arvoksi nollan.

Widget eli pienoisohjelma

Android-ympäristössä asioiden esittäminen laitteen näytöllä tapahtuu View-luokan välityksellä. Luokka tarjoaa useita alaluokkia eri informaation näyttämiseen ja mahdollisuuden rakentaa mukautettu näkymä yksilöllistä tarvetta varten. Pienoisohjelmat tukevat kuitenkin vain osaa valmiista alaluokista, eivätkä hyväksy lainkaan mukautettuja näkymiä. Kuvaajan piirtämiseen käytetty AChartEngine tulostaa vakiona kuvaajat mukautettuna näkymänä. Tämän vuoksi kuvaajaa ei voida suoraan ohjata pienoisohjelmaan, vaan se tulee ensin tallentaa kuvaksi, minkä jälkeen se voidaan ladata kuvanäkymän kautta itse ohjelmaan.

Pienoisohjelman kuvaajan päivityksen haluttiin tapahtuvan sekunnin välein, kuten itse ohjelmassakin. Android-ohjelmointikirjasto sallii kuitenkin minimissään vain 30 sekunnin välein tapahtuvat päivitykset pienoisohjelmiin, joten päivitys täytyi suorittaa käyttäen jotain vaihtoehtoista menetelmää [9]. Vaihtoehtoja löytyi lopulta kaksi, joista ensimmäinen, pienoisohjelman päivitys palvelun avulla, todettiin liian raskaaksi. Päivitys päätettiin toteuttaa siis käyttäen toista vaihtoehtoa, joka hyödyntää ajastimen tavoin toimivaa AlarmManager-luokkaa. Tällöin AlarmManager lähettää sekunnin välein käskyn päivittää manuaalisesti pienoisohjelma, mikäli laite ei ole lepotilassa. Päivitysten aikana pienoisohjelma käyttäytyy kuin BroadcastReceiver-komponentti, joten sen elinkaari poikkeaa esimerkiksi tavallisen Activityn elinkaaresta. Lyhyemmän elinkaaren seurauksena muuttujat tuli tallentaa jokaisen päivityksen välissä. Muuttujien tallennukseen käytettiin jälleen jo aiemmin mainittua SharedPreferences-luokkaa.

Päänäkymä

Ohjelmoitu sovellus suunnattiin alkujaan Androidin 2.3.3-versiolle, joka on ominaisuuksiltaan huomattavasti uudempiin nähden karsitumpi. Käyttöliittymää suunniteltaessa haluttiin kuitenkin käyttää uudempien versioiden ominaisuuksia, kuten esimerkiksi Fragment-luokkaa. Activity-komponentin kaltaiset Fragmentit mahdollistavat muun muassa päänäkymän ylälaidassa esiintyvän kaksinappisen ActionBar-valikon. Jotta valikko pystyttiin toteuttamaan, jouduttiin suurin osa sovelluksen Activityistä kääntämään Fragmenteiksi ja sovelluksen kirjastoon piti tuoda erillinen AppCompat-tukikirjasto.

Androidin virallinen AppCompat-tukikirjasto mahdollistaa versiostaan riippuen uudempien Android-käyttöjärjestelmien eri ominaisuuksien käytön vanhemmilla Androidin versioilla. Tässä tapauksessa riitti, että käytettiin kirjaston versiota 7.

7.3 Kehitys

Sovellusprojektin kehityskaari oli suhteellisen pitkä verrattuna kenties tavalliseen insinöörityönä kehitettyyn sovellukseen. Tämä muodosti haasteen, mitä IT-alalle ominainen teknologioiden nopeatempoinen kehitys vain korosti. Ensimmäisen version julkaisuun ei koskaan kaupallisella tasolla tulisi mennä yhtä kauan kuin tähän projektiin meni. Näin ollen sovellusprojekti auttoikin konkreettisesti hahmottamaan nopeatempoisen kehitystyöskentelyn tärkeyden.

Syitä hitaaseen projektin etenemiseen oli useita, mutta yksi painavimmista oli kokemuksen puute. Sovelluksen ohjelmointi toi vastaan teknisiä haasteita, joiden selvittämiseen saattoi usein kulua päiviä. Näiden ongelmien ratkomisessa muodostuivat internetin tietolähteet tärkeiksi. Erilaiset keskustelupalstat ja ohjeet onnistuivat yleensä avaamaan ongelmia vähintään sen verran, että lopullinen ratkaisu oli saavutettavissa. Kirjoja ei koko projektin aikana käytetty, eikä tämä juurikaan vaikuttanut kehitystyöskentelyyn. Pitää kuitenkin muistaa, että tärkein sovelluksen ohjelmoinnin kannalta, Androidin virallinen dokumentaatio, on kokonaisena verkossa.

Vaikka yleisesti dokumentoinnissa onnistuttiin läpi projektin varsin hyvin, kuitenkin suunnitelmallisuus hieman ontui. Suunnittelutyötä tehtiin, mutta asioita ei järjestelmällisesti kirjattu tai niitä ei pohdittu tarpeeksi laajasti. Sovelluskehityksen alussa olisi ollut järkevää toteuttaa kaavioita sovelluksesta ja hahmottaa käyttöliittymää vaikkapa piirtämällä. Sen sijaan nyt tarvittava suunnittelu toteutettiin osittain lennosta ohjelmoinnin aikana. Eikä kyse ole välttämättä siitä, että lopullinen sovellus olisi muodostunut paremmaksi suunnitelmallisuuden myötä, vaan siitä, että työprosessi olisi edennyt sujuvammin. Suunnitelmallisuuden puutteista huolimatta esimerkiksi koodin kommentointia harjoitettiin aktiivisesti.

Sovelluksen testaaminen jäi pitkälti vain pakolliseen vähittäismäärään. Jokainen ominaisuus pyrittiin käymään läpi poikkeavilla ja normaaleilla arvoilla, minkä seurauksena monesti löydettiin ohjelmointivirheitä. On kuitenkin mahdollista, ettei kaikkia yhdistelmiä

kyetty käymään läpi. Minkäänlaisia testaustyökaluja ei sovelluskehityksen aikana käytetty. Myöskään täyttä varmuutta ei ole siitä, miten sovellus käyttäytyy muilla kuin testatuilla mobiililaitteilla, mikä johtuu Android-laitteiden laitekannan eroavaisuudesta.

8 Yhteenveto

Insinöörityön tarkoituksena oli selvittää mobiililaitteen verkkoliikenteen luonnetta ja luoda sovellus sen monitorointiin ja hallinnointiin. Projektin aihepiiri tarjosi mahdollisuudet laajaan toteutukseen, mutta insinöörityöhön laskettujen resurssien puitteissa pystyttiin ominaisuuksien osalta saavuttamaan vain pakolliset. Näin ollen sovellus jäi kokonaiskäytettävyydeltään vain tyydyttävälle tasolle, eikä se tuo juurikaan uutta nykypäivänä esiintyvien vastaavanlaisten ratkaisujen tarjontaan. Kehityksen alussa tilanne olisi ollut sovelluksen käytännöllisyyden kannalta suopeampi.

Muutamia toimintoja muuttamalla ja ominaisuuksia lisäämällä sovellusta on kuitenkin mahdollista parantaa. Prosessien yksityiskohtaisemmassa näkymässä verkko-osoitteita näyttävä tila toimii vaihtelevasti ja on suunniteltu näyttämään vain paikalliset osoitteet. Etäosoitteiden mukaan ottaminen ja toimivuuden parantaminen toisi ominaisuudelle lisäarvoa. Myös muunlaisen verkkoyhteyksiin liittyvän tiedon, kuten tietoliikenneprotokollan, näyttäminen olisi käyttäjälle hyödyllistä. Näytettävän informaation lisäksi yksityiskohtaisemmassa näkymässä olisi etuna monipuolisemmat mahdollisuudet estää sovellusten verkkoyhteyksiä. Tähän voitaisiin liittää myös sovellusten käyttökiintiöiden valvonta, jolloin käyttökiintiön saavutettuaan valvottavien sovellusten verkkoyhteydet estettäisiin pelkän ilmoituksen sijaan. Kaiken kaikkiaan mobiilisovellus voisi olla lisäksi käyttöliittymältään paremmin sommiteltu.

Pienoisohjelman osalta parannettavaa olisi sen tavassa käyttäytyä näytön sammuttua. Nyt pienoisohjelma päivittää itseään sekunnin välein, kunnes laite siirtyy lepotilaan. Lepotilaan ei kuitenkaan välttämättä päästä kymmenienkään minuuttien jälkeen. Ylimääräinen päivittäminen voitaisiin mahdollisesti estää kaappaamalla näytön tilasta kertovia broadcast-viestejä, mutta tätä ei enempää selvitetty. Pienoisohjelmassa voisi olla myös vaihtoehtoja eri informaation esittämiseen.

Lähteet

1 Wireshark About. Verkkodokumentti. Wireshark Foundation. <https://www.wireshark.org/#aboutWS>. Luettu 17.11.2015.

2 Android. Verkkodokumentti. Wikimedia Foundation. <https://fi.wikipedia.org/w/index.php?title=Android&oldid=14583239>. Luettu 21.1.2015.

3 Activity. Verkkodokumentti. Google. <http://developer.android.com/reference/android/app/Activity.html>. Luettu 23.1.2015.

4 ABI Management. Verkkodokumentti. Google. <http://developer.android.com/ndk/guides/abis.html>. Luettu 22.11.2015.

5 Processes and Application Life Cycle. Verkkodokumentti. Google. <http://developer.android.com/guide/topics/processes/process-lifecycle.html>. Luettu 23.1.2015.

6 Security-Enhanced Linux in Android. Verkkodokumentti. Google. <https://source.android.com/devices/tech/security/selinux/index.html>. Luettu 27.1.2015.

7 About – Git. Verkkodokumentti. Software Freedom Conservancy. <https://git-scm.com/about>. Luettu 18.11.2015.

8 ActivityManager.RunningAppProcessInfo. Verkkodokumentti. Google. <http://developer.android.com/reference/android/app/ActivityManager.RunningAppProcessInfo.html>. Luettu 4.2.2015.

9 AppWidgetProviderInfo. Verkkodokumentti. Google. <http://developer.android.com/reference/android/appwidget/AppWidgetProviderInfo.html#updatePeriodMillis>. Luettu 11.2.2015.