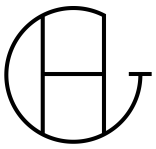


HoGent

Faculteit Bedrijf en Organisatie

Hoe beperken ontwikkelaars de schijfruimte ingenomen door mobiele applicaties voor Android Wear?

Thomas Ledoux

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Jens Buysse
Co-promotor:
Joris Missiaen

Instelling: —

Academiejaar: 2016-2017

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Hoe beperken ontwikkelaars de schijfruimte ingenomen door mobiele applicaties voor Android Wear?

Thomas Ledoux

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Jens Buysse
Co-promotor:
Joris Missiaen

Instelling: —

Academiejaar: 2016-2017

Tweede examenperiode

Samenvatting

De voorbije jaren hebben wearable devices meer en meer aan belang gewonnen in het dagelijks leven. Volgens onderzoek van **IDC** werd het aantal verkochte wearable devices wereldwijd geschat op 20.1 miljoen, waarvan 22.9 procent Android Wear als operating system gebruikt. IDC voorspelt tegen 2020 dat er 54.6 miljoen wearable devices verkocht zullen worden, waarvan 41.8 procent Android Wear als operating system zal gebruiken. Elk noemenswaardig bedrijf heeft tegenwoordig een app laten ontwikkelen die ook op wearable devices kan gebruikt worden om zoveel mogelijk van deze mobiele gebruikers te bereiken. Hiermee gaat echter gepaard dat gebruikers meer en meer apps willen installeren op hun wearables, terwijl deze slechts over een beperkte schijfruimte beschikken. Daarom moeten de ontwikkelaars van deze mobiele applicaties ervoor zorgen dat de grootte van hun applicaties zoveel mogelijk beperkt wordt, zodat mobiele toestellen niet zonder vrije schijfruimte komen te staan. Een bijkomend probleem voor de bedrijven is dat gebruikers een beperktere set van apps zullen installeren, en er dus een grotere kans is dat sommige apps niet meer geïnstalleerd worden en er dus minder omzet wordt behaald. De bedrijven die mobiele apps laten ontwikkelen, zullen hier rekening mee moeten houden en bijvoorbeeld meer diensten in de cloud moeten aanbieden, om zo de lokaal gebruikte schijfruimte te beperken. Om dit na te gaan, zal onderzoek gedaan worden naar de applicaties van enkele van de populairste en meestgebruikte apps, zoals Spotify, Tinder en WhatsApp. In het onderzoek zal de focus gelegd worden op de gebruikte technieken bij ontwikkeling voor Android Wear. Ook zal er bij softwarebedrijf Rialto een proof-of-concept toegepast worden door bij de ontwikkeling van een kopie van de iOS-app in Android Wear. In het onderzoek zal vooral gefocust worden op de technieken die gebruikt worden in de code achter de applicaties voor bijvoorbeeld compressie van bepaalde componenten. Zo zal beschreven worden welke technieken het meest gebruikt worden en welke invloed zij hebben op de gebruikte schijfruimte. Als resultaat van dit onderzoek verwachten we te vinden dat er reeds veel technieken gebruikt worden

om de gebruikte schrijfruimte door mobiele applicaties te verkleinen. Vervolgens zal een vergelijking opgemaakt worden van deze verschillende technieken. In de conclusie verwachten we te vinden dat hoewel er al goede technieken gebruikt worden, er zeker nog verbetering mogelijk is, mede doordat de shift naar wearable devices nog niet heel lang aan de gang is. Ook in de toekomst zal dit onderzoek dus zeker belangrijk blijven, aangezien het er niet op lijkt dat de groei van wearable devices snel zal stoppen.

Voorwoord

Inhoudsopgave

1. Inleiding

1.1 Probleemstelling en Onderzoeksvragen

In deze bachelorproef worden verschillende technieken beschreven en vergeleken om de grootte van apps voor Android Wear te beperken. Er werd voor Android Wear gekozen omdat dit een open-source platform is. De belangrijkste pijlers van open-source zijn dat open-source software vrij kan gedistribueerd worden en dat de source code beschikbaar wordt gesteld volgens **Open Source Initiative** . Dit vergemakkelijkt het onderzoek doordat er in de broncode kan gekeken worden welke technieken precies gebruikt worden, en op welke manier deze geïmplementeerd worden. Android Wear toestellen beschikken vaak maar over een beperkte opslagruimte, waardoor elke megabyte vrije ruimte telt. De eindgebruikers van de apps hebben hier dus het meest directe effect van, zij zullen kunnen profiteren van de vrijgekomen ruimte door de compressie van de apps. Maar ook voor bedrijven en hun app-ontwikkelaars is dit belangrijk, wanneer hun Android Wear app niet te veel schrijfruimte inneemt op het toestel van de eindgebruikers, zullen zij minder snel geneigd zijn deze app te verwijderen. Wanneer ze de app wel zouden verwijderen ontloopt het bedrijf belangrijke inkomsten uit in-app purchases en eventuele reclame-inkomsten. Het vergelijken van de verschillende technieken zal gebeuren op verschillende vlakken : Zijn er veel media-bestanden aanwezig in de applicatie? Is er veel code aanwezig in de applicatie? Zijn er meerdere APK's voorzien?

1.2 Opzet van deze bachelorproef

Om dit onderzoek te ondersteunen zal een proof-of-concept app gecreëerd worden voor Android Wear. Op deze app zullen dan de verschillende technieken toegepast worden, en

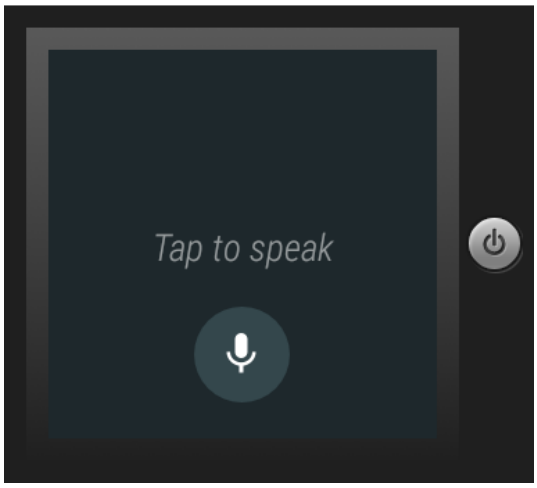
zal het effect van deze technieken vergeleken en besproken worden. Er wordt ook een literatuurstudie uitgevoerd waarin zal besproken worden wat Android Wear precies inhoudt, en waar de verschillende compressietechnieken besproken worden. Er zal ook gekeken worden of er al dan niet een verband is tussen de compressie van de apps en de prestatie ervan op vlak van geheugengebruik. Nadien zullen de resultaten van het onderzoek besproken worden. Tot slot volgt een samenvattende conclusie van het onderzoek.

2. Methodologie

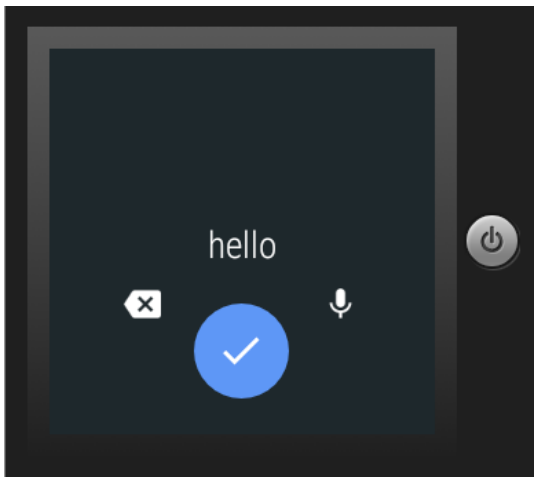
2.1 Creatie proof-of-concept app

Om het onderzoek te onderbouwen, werd een Android Wear proof-of-concept app gecreëerd. Dit is een algemene applicatie, waarin verschillende elementen toegevoegd worden zoals media-bestanden, veel code.. Het doel van de creatie van deze app is om de verschillende compressietechnieken hierop toe te passen, en de resultaten grafisch voor te stellen. De applicatie draait op Android Wear 2.0, deze versie werd op 09/02/2017 uitgebracht door **Google** . Doordat voor deze versie gekozen werd, kan de app als standalone app gebruikt worden. Dit houdt in dat de app op een smartwatch kan gebruikt worden onafhankelijk van een smartphone. De gebruikers kunnen meer taken op hun smartwatch uitvoeren zonder toegang tot een Android of iOS smartphone. De app zal als functie hebben om gemakkelijk notities te maken via spraakherkenningstechnologie. Hierdoor kunnen de gebruikers door te spreken in de microfoon van de smartwatch een nieuwe notitie maken.

Figuur 2.1: *Scherm voor spraakherkenning*



Figuur 2.2: *Resultaat van spraakherkenning*



2.2 Verschillende technieken toepassen op proof-of-concept app

2.3 Prestaties en grootte app testen na toepassen technieken

2.4 Testen van app door framework Brian Pinsard

3. Literatuurstudie

3.1 Wat is Android Wear?

Android Wear is Google's platform voor wearables. Een wearable worden beschreven door **Dictionary** beschreven als een computer of geavanceerd elektronisch toestel dat geïncorporeerd is in een accessoire gedragen op het lichaam of in een kledingstuk. **Techradar** heeft enkele van de belangrijkste kenmerken van Android Wear opgelijst. Android Wear is gebouwd voor kleinere toestellen en met hands-free gebruik in het achterhoofd. Het maakt de toegang tot enkele van uw smartphone's makkelijkste functionaliteiten zo makkelijk als naar beneden kijken naar uw pols. Het is vooral bedoeld om de notificaties van uw smartphone makkelijk te weergeven zodat niet elke keer gezocht moet worden naar de smartphone in de broekzak. Gebaseerd op uw Google zoekopdrachten zullen real-time scores van uw favoriete sportteam, verkeerdscondities of afspraken in uw afgenda weergeven worden. Als het Android Wear toestel NFC (Near Field Communication) ingebouwd heeft, kan er in verschillende landen (momenteel Australië, Hong Kong, Ierland, Japan, Nieuw-Zeeland, Polen, Singapore, Verenigd Koninkrijk en de Verenigde Staten volgens de support van **Androidpay**) ook met de wearable betaald worden via Android Pay. Ook spraakherkenning kon niet ontbreken in de Android Software. Door deze technologie kan je de wearable's spraakherkenning activeren door "OKay, Google"te zeggen of door de aanknop ingedrukt te houden. Google Assistant zal u hierop verderhelpen. Android Wear is compatibel met smartphones die draaien op Android 4.3 of hoger en iOS 9 of hoger. Eén van de belangrijkste app categorieën op Android Wear zijn de gezondheidsapps. Hiervoor wordt standaard gebruik gemaakt van Google Fit. Met Google Fit kan je kiezen tussen verschillende workouts tijdens het sporten, of bijvoorbeeld een dagelijks doel instellen voor het aantal stappen dat u wil nemen.

Figuur 3.1: *Enkele voorbeelden van smartwatches gebruikmakend van Android Wear*

3.2 Wat is een APK en uit wat bestaat dit?

APK staat voor Android Application Package. Volgens **PCMag** is een APK een applicatie bestand klaar voor installatie op een Android toestel. Het gecompresseerde APK bestand, een ZIP archief in JAR-formaat, wordt gedistribueerd naar Android gebruikers voor de installatie op hun smartphone, tablet of wearable. Een APK bestand bestaat normaliter uit volgende elementen :

1. **classes.dex**
   Bevat gecompileerde applicatiecode, getransformeerd naar een Dex bytecode. Het is mogelijk dat er meerdere DEX bestanden in de APK staan als er multidex gebruikt wordt om de 65536 limiet te overkomen. Vanaf Android 5.0, met de introductie van ART runtime, worden deze bestanden gecompileerd naar OAT bestanden door de compiler bij de installatie en worden deze op het toestel zijn data partitie geplaatst.

2. **res/**
   Deze folder bevat de meeste XML bestanden en drawables (bijvoorbeeld PNG of JPEG bestanden) in mappen met verschillende kwalificaties, zoals -mdpi en -hdpi voor schermdichtheden, -sw600dp of -large voor schermgroottes, en -en, -de, -pl voor talen. De XML bestanden worden automatisch gecompileerd naar een compactere binaire representatie, dus deze kunnen niet met een teksteditor geopend worden vanuit de APK.

3. **resources.arsc**
   Sommige resources en identifiers worden gecompileerd naar dit bestand. Het wordt normaal ongecomprimeerd opgeslagen in de APK voor snellere toegang tijdens runtime. Dit bestand manueel comprimeren kan een makkelijke oplossing lijken, maar dit is geen goed idee door 2 redenen. Eén, Play Store comprimeert alle data voor transfer automatisch en twee, dit bestand comprimeren in de APK verspilt systeem resources (RAM) en performantie (vooral de opstarttijd van de app).

4. **AndroidManifest.xml**
   Gelijklopend met andere XML resources wordt de applicatie Manifest getransformeerd naar een binair formaat tijdens de compilatie. De Play Store gebruikt bepaalde informatie in de AndroidManifest om te bepalen of een APK kan geïnstalleerd worden op een bepaald toestel, zo zijn er controles op toegelaten schermdichtheden of schermgroottes, beschikbare hardware en kenmerken (bijvoorbeeld aanwezigheid van een touchscreen). Deze Manifest inhoud kan geïnspecteerd worden na compilatie, gebruik hiervoor de aapt tool van de Android SDK:

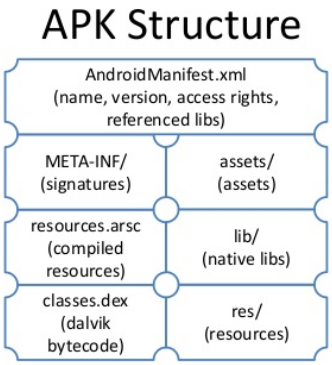
```
$ aapt dump badging your_app.apk
```

5. **libs/**
   Alle native libraries (*.so bestanden) zullen in submappen geplaatst worden genaamd naar de ABI (CPU architectuur, bv. x86, `x86_64`, armeabi-v7a) die ze als doel hebben onder de libs/ map. Normaal worden deze uit de APK gekopieerd naar de datapartitie tijdens het installeren van de applicatie. Echter, sinds de APK zelf nooit aangepast wordt zolang deze op het toestel van de gebruiker staat, neemt dit dubbel zoveel plaats in voor elke native library.

6. **assets/**
   Deze map zal gebruikt worden voor alle bestandonderdelen die niet als Android-type resources gebruikt zullen worden. Meestal zullen dit lettertypebestanden of game data zijn, zoals levels of textures, samen met andere applicatiedata die direct geopend moet kunnen worden als een file stream.

7. **META-INF/**
   Deze map is aanwezig in ondertekende APKs en bevat een lijst van alle bestanden in de APK met hun handtekening. Momenteel werkt het ondertekenen van bestanden in Android door het verifiëren van de handtekening tegen de ongecomprimeerde bestandinhoud van het archief, één voor één. Dit heeft enkele interessante gevolgen. Doordat elke onderdeel van een ZIP bestand apart opgeslagen wordt, betekent dit dat je individuele bestanden hun compressieniveau kan aanpassen zonder ze opnieuw te ondertekenen. De verificatie van de handtekening zal echter falen wanneer een bestand verwijderd wordt uit het archief nadat het ondertekend werd.

Figuur 3.2: *De structuur van de APK*



3.3 Verschillende compressietechnieken

Technieken gebruikt bij Cozmos, app-ontwikkelaar van Android (Wear) apps

Bij Cozmos is de belangrijkste tool die gebruikt wordt om de APK zo klein mogelijk te houden Proguard. Proguard zorgt er niet enkel voor dat je code moeilijker leesbaar wordt bij reverse engineering maar het verwijdert ook al je ongebruikte code en resources. Bij de grotere klanten waar ook security van groot belang is maken we gebruik van Dexguard. Dexguard is de commerciële variant van Proguard en gaat een stap verder met zijn compressietechnieken maar biedt daarnaast ook tal van andere features aan zoals bescherming van de APK of SDK (Software Development Kit) tegen het klonen ervan, piraterij en extractie van sleutelwaarden uit de APK door gebruik te maken van verschillende encryptie- en obfuscatietechnieken. Anderzijds beschermt Dexguard ook bescherming tegen Dynamic Analysis, zodat er tijdens dat de app actief is geen aanpassingen kunnen gebeuren door omgevings- en certificaatcontroles.

Eén van de andere manieren om de grootte van de APK te beperken die gebruikt wordt bij Cozmos is kijken naar de manier waarop images en andere resources gebruikt worden. Er kan al ruimte bespaard worden door een juiste keuze te maken tussen jpg/png en verdere compressie toe te passen. Soms is ook geen volledige image nodig maar kan er geopteerd worden voor een 9-patch image of om een drawable in XML te definieren. Bij animaties kan je dan weer het aantal frames gaan beperken.

Bij het gebruik van libraries kan het handig zijn om enkel de modules in het project te importeren die ook daadwerkelijk nodig zijn. Ook worden verschillende libraries tegenover elkaar afgewogen en wordt gekeken welke de laagste method count hebben, het best met het geheugen omgaan, het minste plaats innemen.. Stel dat men een library aan het project zou willen toevoegen om images in te laden dan zou de afweging kunnen gemaakt worden tussen Glide en Picasso. Picasso is 3,5 keer kleiner dan Glide maar anderzijds gebruikt Glide een pak minder geheugen om een image weer te geven. Daar moet dus een afweging

gemaakt worden waar de focus ligt voor de app, meer vrije opslagruimte of meer vrij geheugen.

Een andere optie is om niet-essentiële zaken achterwege te laten uit je APK. Afbeeldingen of data die maar voor een beperkte groep gebruikers van belang zijn kunnen beter in realtime gedownload worden in plaats van deze al in de APK te voorzien.

Verder kan er door middel van code zuiniger omgesprongen worden met opslagruimte. Zo dienen enums vermeden te worden, kan je (eenvoudige) afbeeldingen renderen..

Uit de richtlijnen van **googlereduceapksize** voor het verkleinen van de grootte van de APK werden volgende technieken gekozen om te onderzoeken:

Verwijder ongebruikte resources

De ingebouwde lint tool in Android Studio, een statische code analyzer, detecteert resources in de res/ map die in de code niet gerefereerd worden. Android Studio zal hiervan een melding maken, maar zal deze niet automatisch verwijderen. Libraries die u toevoegt kunnen ongebruikte resources meebrengen. Gradle kan deze automatisch verwijderen als "shrinkResourcesïn de build.gradle wordt toegevoegd.

Verklein resource gebruik van libraries

Wanneer u een Android app ontwikkelt, worden vaak externe librbaries gebruikt om de gebruiksvriendelijkheid te verhogen. Deze libraries bevatten vaak elementen die voor desktops of servers bedoeld zijn, deze elementen kunnen uit de libraries verwijderd worden indien de licentie dit toestaat.

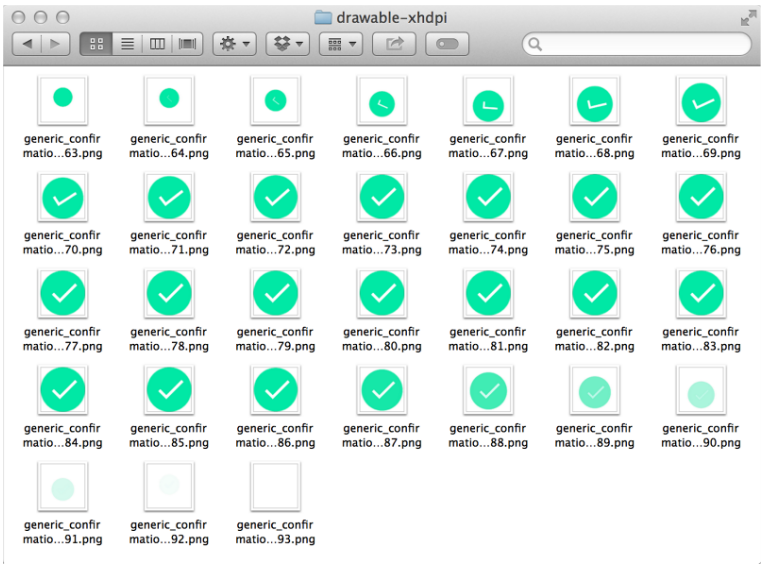
Ondersteun enkele specifieke schermdichtheden

Vanaf Android 4.4 worden verschillende schermdichtheden ondersteund (ldpi, mdpi, tvdpi, hdpi, xhdpi, xxhdpi en xxxhdpi). Het is niet nodig om elke afbeelding te exporteren naar elke dichtheid. Android zal automatisch de afbeelding schalen naar andere schermdichtheden als er geen specifieke export voorzien is.

Verminder animatie frames

Voor elke frame in een animatie wordt een aparte afbeelding opgeslaan. Als er bijvoorbeeld een animatie aanwezig is met 30 FPS (frames per second) zullen er 30 afbeeldingen opgeslagen worden, maar vaak is bijvoorbeeld 15 FPS meer dan voldoende, en wordt dus de helft van de ingenomen opslagruimte gebruikt.

Figuur 3.3: *Door vermindering FPS zullen minder aparte afbeeldingen per animatie opgeslagen worden*



Hergebruik resources

Het is mogelijk om voor elke variatie op een afbeelding (getint, met schaduw of geroteerd) een aparte resource op te slaan. Google raadt echter aan om 1 afbeelding te voorzien, en deze dynamisch tijdens runtime aan te passen. Android voorziet verschillende utiliteiten om de kleur van een afbeelding aan te passen.

Comprimeer afbeeldingen

De aapt tool kan de afbeelding resources in res/drawable/ optimaliseren met lossless compressie. De tool kan bijvoorbeeld een true-color PNG bestand dat niet meer dan 256 kleuren bevat omzetten naar een 8-bit PNG met een kleurenpallet. Dit resulteert in een afbeelding van eenzelfde kwaliteit maar met een kleinere geheugenvoetafdruk. Deze tool kan geen PNG bestanden in de asset/ folder verkleinen.

Gebruik WebP bestandformaat

In plaats van PNG of JPEG bestanden, kan er ook gebruik gemaakt worden van het WebP formaat voor afbeeldingen. Dit formaat voorziet lossless compressie (zoals JPEG) en transparantie (zoals PNG) maar voorziet betere compressie dan beide andere formaten. Het gebruik van WebP heeft echter ook enkele nadelen. WebP-ondersteuning is niet beschikbaar bij versies lager dan Android 3.2 (API level 13). Het neemt ook een langere tijd voor het systeem om WebP bestanden te decoderen dan PNG bestanden. Bestaande BPM, JPG, PNG of statische GIF afbeeldingen kunnen naar het WebP formaat omgezet worden door Android Studio.

Gebruik vector graphics

Vector graphics kunnen gebruikt worden om resolutie-onafhankelijke iconen of andere schaalbare media te creëren. Deze afbeeldingen worden in Android gerepresenteerd als VectorDrawable objecten. Met een VectorDrawable object kan een 100-byte bestand een scherpe afbeeldingen genereren op de grootte van het scherm. Echter, het neemt a significante tijd voor het systeem om elk VectorDrawable object te genereren. Grote afbeeldingen kunnen zelfs nog langer nemen om op het scherm te verschijnen. Daarom wordt aangeraden de VectorDrawable objecten enkel voor kleine afbeeldingen te gebruiken.

3.3.1 Verminder gebruik code

Verwijder automatisch gegenereerde code

Zorg dat u verstaat wat elke lijn gegenereerde code inhoudt, en wat de voetafdruk ervan is. Zo zullen veel protocol buffer tools een groot aantal methodes en klassen genereren, die de grootte van de app kunnen verdubbelen of verdriedubbelen.

Verwijder enumeraties

Elke enum kan 1 tot 1.4 KB grootte toevoegen aan uw app's classes.dex bestand. Deze toevoegingen kunnen snel oplopen voor complexe systemen of gedeelde libraries. Indien mogelijk gebruikt u best de @IntDef annotatie en ProGuard om enumeraties te strippen en om te zetten in integers. Deze conversie behoudt alle voordelen van enums.

Verklein grootte native binaries

Bij het gebruik van native code en de Android NDK (Native Development Kit, hierdoor kan je bepaalde delen van je app implementeren in native-code talen zoals C of C++) kan je de grootte van de app verkleinen door het optimaliseren van de code. Hiervoor zijn 2 technieken bruikbaar : 1) Debug symbolen verwijderen Wanneer de app niet meer in development is, zijn debug symbolen overbodig. Deze kan je uit de applicatie verwijderen door de ärm-eabi-strip"tool te gebruiken, deze zit ingebakken in Android NDK. 2) Vermijd het extraheren van native libraries Bewaar .so bestanden ongecompresseerd in de APK, and stel de ändroid:extractNativeLibs"vlag op false in het "<application>ëlement van je app manifest. Dit zal vermijden dat de PackageManager de .so bestanden uit de APK naar het bestandssysteem zal kopiëren tijdens de installatie, en zal als bijkomend voordeel hebben dat delta updates van de app kleiner zullen zijn.

3.3.2 Voorzie meerdere APK's

De app kan inhoud bevatten die gebruikers downloaden maar nooit gebruiken, zoals regio- of taalgerelateerde informatie. Om een minimale download voor de gebruikers te voorzien, kan de app gesegmenteerd worden in verschillende APKs, gedifferentieerd door factoren

zoals schermgrootte of GPU structuur. Wanneer gebruikers de app downloaden zal hun toestel de correcte APK ontvangen gebaseerd op het toestel zijn instellingen en kenmerken. Op deze manier ontvangen hun toestellen geen inhoud bedoeld voor opties die het toestel niet heeft.

3.4 Verband tussen compressie en prestaties apps

4. Resultaten

4.1 Vergelijking verschillende technieken

4.2 Invloed compressie op prestaties apps

4.2.1 Aanpak bij Cozmos

Er wordt zeker gecontroleerd dat de performantie van de app goed zit. Bij het uitvoeren van network calls moet alles zo compact mogelijk zijn zodat een gebruiker ook onder een slechte verbinding kan blijven werken. Bij het tonen van meerdere afbeeldingen is het memory management zeer belangrijk om onder andere out of memory exceptions te vermijden.

Op welk vlak de performantie een invloed heeft hangt af van ieder geval op zich. Er kan gesteld worden dat er op 1 of meerdere van volgende punten verbetering merkbaar is:

1. Grootte beperken

2. Geheugengebruik beperken

3. Sneller laden

Echt actief gaan monitoren welke prestatieverschillen een compressietechniek met zich meebrengt daar is meestal de tijd niet voor. Het is pas wanneer er bij het testen zaken naar boven komen dat er echt wordt gekeken welke extra stappen er kunnen ondernomen

worden en welke winst er geboekt wordt. Uiteraard worden bovenstaande technieken tijdens het developen in het achterhoofd gehouden zodat er een optimaal resultaat bekomen wordt.

4.3 Testresultaten framework Brian Pinsard

5. Conclusie

Lijst van figuren

Lijst van tabellen