TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Desenvolupament d'aplicacions multimèdia i d'anàlisi de trànsit de xarxa en smartphones Android

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Pablo Peinado Martínez

DIRECTOR: David Rincón Rivera

DATA: 3 de maig de 2013

Títol: Desenvolupament d'aplicacions multimèdia i d'anàlisi de trànsit de xarxa en smartphones Android

Autor: Pablo Peinado Martínez

Director: David Rincón Rivera

Data: 3 de maig de 2013

Resum

En aquest treball s'han estudiat les característiques multimèdia, les capacitats de captura de transit de xarxa i el servei de localització que ofereix la plataforma Android, amb la idea d'aprofitar l'auge de les aplicacions mòbils, juntament amb l'avantatge d'utilitzar software lliure, per introduir la programació Android en activitats docents. Aquesta plataforma, desenvolupada per un grup d'empreses del sector tecnològic i mòbil (la Open Handset Alliance), és un sistema operatiu lliure dissenyat per a dispositius mòbils o tabletes i actualment el trobem en *smartbooks*, *smartTVs* o càmeres.

D'una banda s'han detallat els codificadors, descodificadors, contenidors i protocols de xarxa que suporta Android, per després recollir aquests coneixements en una aplicació de captura de medis on és permet triar el format dels fitxers de sortida. En concret, l'aplicació pot generar imatges en format JPEG, PNG, GIF i BMP, capturar vídeo codificat amb H.263, H.264 i MPEG-4, i encapsular vídeo (H.263 i H.264) en paquets RTP per enviar-los a través d'una xarxa IP en temps real.

D'altra banda, hem analitzat el sistema de localització que inclou la plataforma, determinant el format de les dades retornades i la seva precisió. També hem calculat, a través dels resultats de les proves, la freqüència d'actualitzacions que proporciona el servei. Aprofitant part del codi de l'aplicació Shark realitzada per un altre estudiant de l'EETAC, i posant en pràctica els coneixements sobre localització a Android, hem desenvolupat un servei que emmagatzema, en paral·lel, les dades d'ubicació i el tràfic generat i rebut per un terminal mòbil. La finalitat d'aquest servei és, a través de l'anàlisi conjunt de les dades generades per molts dispositius que corrin l'aplicació, millorar la planificació i desplegament de xarxes de telefonia mòbil i Wi-Fi. De forma addicional s'ha implementat una aplicació que, amb les dades generades a un terminal, determina a quina localització hem rebut (o enviat) cada paquet de xarxa i representa aquesta informació en un mapa de Google Maps.

Title: Development of multimedia and network traffic monitoring applications in Android smartphones

Author: Pablo Peinado Martínez

Director: David Rincón Rivera

Date: May 3rd, 2013

Overview

In this thesis we have studied the capabilities of the Android platform regarding multimedia, network traffic capture and location-based services. Our ultimate goal was to develop code to help introduce Android programming in teaching activities, taking advantage of the boom in mobile applications coupled with the benefits of using free software. Android was developed by a group of technology and mobile companies (the Open Handset Alliance), and is a free operating system, designed for mobile devices and tablets and currently found on smartbooks, SmartTVs or cameras.

We studied and documented the use of encoders, decoders, containers and networking protocols supported by Android. We applied this knowledge for develop a media application that allows to choose the coders and output formats. The application can take JPEG, PNG, GIF and BMP pictures, record video encoded with H.263, H.264 and MPEG-4, and encapsulate video (H.263 and H.264) in RTP packets to send them through an IP network in real time.

Moreover, we analyzed the location capabilities of the platform, determining the format of the returned data and its accuracy. We evaluated, through the results of tests, the frequency of the location updates offered by the service. We developed a service that stores, in parallel, the location data and network traffic generated and received in the mobile device. To do this, we used part of Shark application developed by another student at EETAC and added location capabilities. The purpose of this service is, through the joint analysis of generated data in many devices that run this application, improve the planning and deployment of mobile and Wi-Fi networks. Additionally, we implemented an application that analyzes the data generated in one device and draws it on Google Maps.

ÍNDEX

INTRODUCCIÓ

Els dispositius mòbils han evolucionat a una velocitat vertiginosa des de la digitalització de les comunicacions a la dècada dels 90 fins l'actualitat. Hem deixat enrere veus sintètiques, pantalles monocromes, missatges multimèdia i gairebé els (tan populars fins fa poc) missatges SMS, entre d'altres. Això ha provocat canvis en la forma en que ens comuniquem, ens relacionem i, sobretot, ha introduït noves necessitats a les nostres vides.

Avui dia podem observar contínuament a persones aprofitant les noves funcionalitats que ofereixen els *smartphones*: ja sigui al metro llegint les notícies, fent fotos o vídeos davant la Sagrada Família o caminant, amb cara de perduts, mentre consulten la seva ubicació al telèfon.

Com en tots els mercats, al de la telefonia mòbil hi trobem diverses empreses competint per desenvolupar nous sistemes operatius que revolucionin el mercat. Apple deixa expectant mig món amb les novetats del nou sistema operatiu iOS6, on sembla que inclourà el seu propi software de mapes amb un model 3D [1]. Nokia no es queda enrere amb els dispositius Lumia sobre Windows Phone o la nova gama tàctil Asha [2]. Google va comprar la companyia Android Inc. el 2005 [3] i, juntament amb la Open Handset Alliance, ha desenvolupat el primer sistema operatiu lliure per telèfons mòbils [4].

Actualment, Android és la plataforma per mòbils més popular del món. Està publicada, la majoria part, com a codi obert sota la llicència *Apache Software License 2.0*[1]. El projecte AOSP (*Android Open Source Project*) s'encarrega del manteniment i del desenvolupament [5].

Aquestes característiques afavoreixen introduir la programació Android en docència. En l'àmbit de la Telemàtica, podem enfocar l'aprenentatge d'Android cap a aplicacions multimèdia: captura i compressió de continguts audiovisuals i transmissió d'aquests fluxos comprimits sobre xarxes IP.

Aquest treball vol, per una banda, analitzar les capacitats multimèdia del sistema operatiu Android i recollir-les de forma pràctica en una aplicació senzilla. Això inclou capacitat de codecs en reproducció o captura d'àudio, imatge i vídeo, els contenidors suportats i els protocols de xarxa disponibles per a transmissió de fluxos audiovisuals.

Uns altres temes interessants de cara a la docència són els serveis de localització inclosos a Android, així com les capacitats de captura de trànsit. Per això un altre dels objectius del TFC ha estat desenvolupar una aplicació capaç de capturar el tràfic de xarxa generat o consumit per el mòbil, extreure'n la informació més rellevant, i emmagatzemar-la incloent-hi la ubicació des de on ha estat capturat. Això obriria la possibilitat de fer estudis de caracterització de trànsit no només en funció del temps, com és habitual, sinó també de la

[1] Apache Software License 2.0 http://www.apache.org/licenses/LICENSE-2.0

localització geogràfica, possibilitant millores en els algoritmes de disseny i d'assignació de capacitats a xarxes mòbils. Cal estudiar com es guardaran aquestes dades: modificació del format *pcap* (utilitzat per Wireshark [6]) o bé desar per separat i en paral·lel la informació de localització.

El treball s'estructura en cinc capítols, a través dels quals veurem què és i com a evolucionat Android, analitzarem les eines i capacitats que ofereix la plataforma en l'àmbit de la multimèdia i finalment relacionarem a través d'una aplicació els conceptes de tràfic de xarxa amb geolocalització. A continuació descrivim els continguts de cada capítol.

El primer capítol presenta un resum de la història del sistema operatiu Android i la seva evolució, detallant l'arquitectura de la plataforma i les versions publicades.

Al segon capítol explicarem l'estructura, les capacitats i el funcionament de l'aplicació CustomCam, desenvolupada partint de l'anàlisi de les capacitats multimèdia d'Android a nivell de codificadors, contenidors i protocols de xarxa. La nostra aplicació permet, per una banda, capturar imatges i vídeo en diferents formats i, per l'altra, treballar com un servidor d'*streaming*: encapsula fluxos de vídeo i àudio (codificat) en paquets RTP i els envia en temps real sobre una xarxa IP.

Al tercer capítol detallem el servei de geolocalització de la plataforma Android: com accedir-hi i demanar la localització, el format de les dades retornades i de quina manera podem representar-les. Finalitzem el capítol realitzant un petit experiment per mesurar el ritme d'actualitzacions d'ubicació que ofereix el servei.

El quart capítol descriu les eines que utilitzarem per capturar i analitzar el tràfic de xarxa. I com obtenir els permisos necessaris per executar la eina de captura, *tcpdump*.

Al cinquè capítol, explicarem què és i per a que serveix localitzar el tràfic de xarxa. Detallarem quines dades volem capturar i el seu format. Finalment ho il·lustrarem amb la implementació d'una aplicació que, posant en comú els coneixements apresos al tercer i quart capítol, permet capturar aquestes dades i fer-ne un anàlisi.

La memòria finalitza amb les conclusions obtingudes i les línies futures de desenvolupament.

CAPÍTOL 1. LA PLATAFORMA ANDROID

Android és una plataforma de *software* amb sistema operatiu basat en Linux, dissenyada per a dispositius tàctils (mòbils i posteriorment *tablets)*. En aquest capítol veurem com va començar Android i la seva evolució, seguidament detallarem l'arquitectura de la plataforma i justificarem quina versió de la API hem utilitzat per desenvolupar les aplicacions.

1.1. Visió global

1.1.1. Història

A l'octubre de 2003 Andy Rubin, Rich Miner, Nick Sears i Chris White van fundar l'empresa Android Inc. Dos anys més tard, a l'agost de 2005, Google va comprar la marca. Finalment, Android va veure la llum a l'any 2007, juntament amb la fundació *Open Handset Alliance*: un consorci d'empreses especialitzades en *hardware*, *software* i telecomunicacions [7].

La primera versió comercial del *software*, Android 1.0 Apple Pie, va ser llançada el vint-i-tres de setembre de 2008. En l'actualitat, després de aproximadament trenta versions publicades i cinc anys al mercat, es comercialitza la versió 4.2 Jelly Bean [8]. La taula 1.1 mostra la cronologia d'actualitzacions d'Android.

Taula 1.1 Cronologia de les versions d'Android

Versió	Nom clau	API	Data
1.0	Apple Pie	1	Setembre de 2008
1.1	Banana Bread	2	Febrer de 2009
1.5	Cupcake	3	Abril de 2009
1.6	Donut	4	Setembre de 2009
2.0 – 2.1	Eclair	5, 6 i 7	Octubre de 2009 a gener de 2010
2.2 – 2.2.3	Froyo	8	Maig de 2010 a novembre de 2011
2.3 – 2.3.7	Gingerbread	9 i 10	Desembre de 2010 a setembre de 2011
3.0 – 3.2.6	Honeycomb	11, 12 i 13	Febrer de 2011 a febrer de 2012
4.0 – 4.0.4	Ice Cream Sandwich	14 i 15	Octubre 2011 a març de 2012
4.1 – 4.2.2	Jelly Bean	16 i 17	Novembre de 2012 a febrer de 2013

1.1.2. Arquitectura

L'arquitectura del sistema operatiu Android està estructurada en cinc components principals: les aplicacions, el marc de treball, les llibreries, el temps d'execució d'Android i el nucli de Linux [9]; tal i com mostra la figura 1.1.

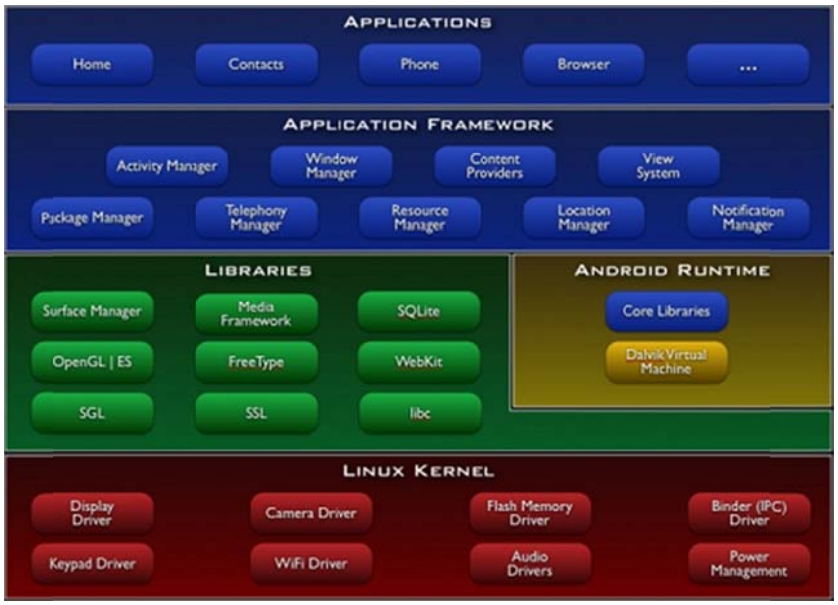

Fig 1.1 Arquitectura del sistema operatiu Android. Extret de [9].

1.1.2.1. Aplicacions

Android incorpora un conjunt d'aplicacions preinstal·lades que inclou un client de correu electrònic, un programa de missatges de text (SMS), un navegador web i un gestor de contactes entre d'altres. Totes aquestes aplicacions han estat escrites utilitzant el llenguatge de programació Java. A l'annex 1 podem trobar els conceptes clau per iniciar el desenvolupament d'aplicacions Android.

1.1.2.2. Marc de treball

Amb l'objectiu d'oferir una plataforma de desenvolupament lliure, Android permet accés complet al marc de treball (API *framework*). Els desenvolupadors, a través d'aquestes eines, poden beneficiar-se del hardware del dispositiu, accedir a la informació de localització, executar serveis en segon pla, afegir alarmes i notificacions, capturar i reproduir formats multimèdia, etc.

El paradigma de programació Android està dissenyat per fomentar i optimitzar la reutilització de components; les aplicacions poden aprofitar funcionalitats ofertes per altres. Per exemple, si volem visualitzar una imatge des de la nostra aplicació no és necessari implementar tot el mecanisme, ja que podem fer una petició al sistema (llençar un `Intent`) per comprovar si existeix alguna aplicació al telèfon capaç de realitzar aquesta tasca.

1.1.2.3. Llibreries

Android inclou un conjunt de llibreries escrites en C/C++ que proporcionen capacitats clau a la plataforma: emmagatzematge a base de dades (SQLite), gestor de finestres, navegador web, gràfics 2D i 3D, i llibreries multimèdia,

entre d'altres. Els desenvolupadors poden accedir a aquestes funcionalitats a través del marc de treball, utilitzant les classes de la API escrites en Java.

No obstant, Android ofereix una eina per desenvolupar les nostres pròpies llibreries escrites en C/C++ o utilitzar altres no incloses a la plataforma. Podem trobar més informació sobre el desenvolupament d'aplicacions natives a l'apartat A1.3 de l'annex 1.

1.1.2.4. Runtime d'Android

La plataforma inclou un conjunt de llibreries que proporcionen les principals funcionalitats del nucli del llenguatge de programació Java. Totes les aplicacions s'executen en el seu propi procés; la seva instància de la màquina virtual Dalvik. Els dispositius poden executar diverses màquines virtuals de forma eficient. Dalvik executa classes compilades amb Java i transformades en fitxers .dex (*Dalvik Executable*).

La màquina virtual Dalvik delega la gestió dels *threads* i de la memòria a baix nivell al nucli de Linux.

1.1.2.5. Nucli de Linux

El nucli de Linux actua com una capa d'abstracció entre el *hardware* i la resta de components de la plataforma. Android basa els serveis bàsics del sistema (seguretat, gestió de memòria i dels processos, pila de xarxa i *drivers*) en el *kernel* de Linux. La versió del *kernel* s'ha actualitzat des d'Android 1.5 (versió 2.6.27 del nucli de Linux) fins Android 4.1 (versió 3.0.31 del nucli de Linux).

1.1.3. Manifest i permisos

El manifest d'Android (*AndroidManifest.xml*) és la carta de presentació i la declaració d'intencions d'una aplicació; el trobarem sempre a l'arrel del projecte. El manifest conté la informació essencial de l'aplicació: descripció dels seus components (activitats, serveis, *broadcast receivers* i proveïdors de contingut), el nom del paquet Java, la versió mínima requerida de la API i el llistat de permisos necessaris.

Android bloqueja certes parts de la API, forçant que només siguin accessibles a través de la declaració, al manifest, dels permisos corresponents. D'aquesta manera es garanteix que les aplicacions són transparents, ja que han d'informar a l'usuari per accedir a la informació de localització, escriure a la targeta SD, realitzar trucades o recuperar l'estat de la xarxa Wi-Fi. Podem trobar el llistat complet de permisos a la documentació oficial [10].

El següent exemple mostra el manifest d'una aplicació Android que inclou una activitat (la que es llença inicialment) un servei i requereix el permís per obrir *sockets* de xarxa.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.tfc" android:versionCode="1" android:versionName="1.0">
  <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="17" />
  <uses-permission android:name="android.permission.INTERNET" />
  <application android:icon="@drawable/ic_launcher">
    <activity android:name="com.tfc.MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <service android:name="com.tfc.SampleService" />
  </application>
</manifest>
```

1.2. Versió d'Android utilitzada al TFC

CustomCam i LocatedTraffic són les dos aplicacions Android desenvolupades, a nivell Java, durant la realització d'aquest projecte. Es van dissenyar per a dispositius amb Android 2.2 (*Froyo*) o superior per dos motius: d'una banda, utilitzar la versió d'Android del telèfon que es disposa per testejar, i d'altra banda en el moment d'iniciar el projecte (agost de 2011), *Froyo* era la versió més estesa comercialment. Actualment, la versió més comercialitzada és *Gingerbread* (Android 2.3).

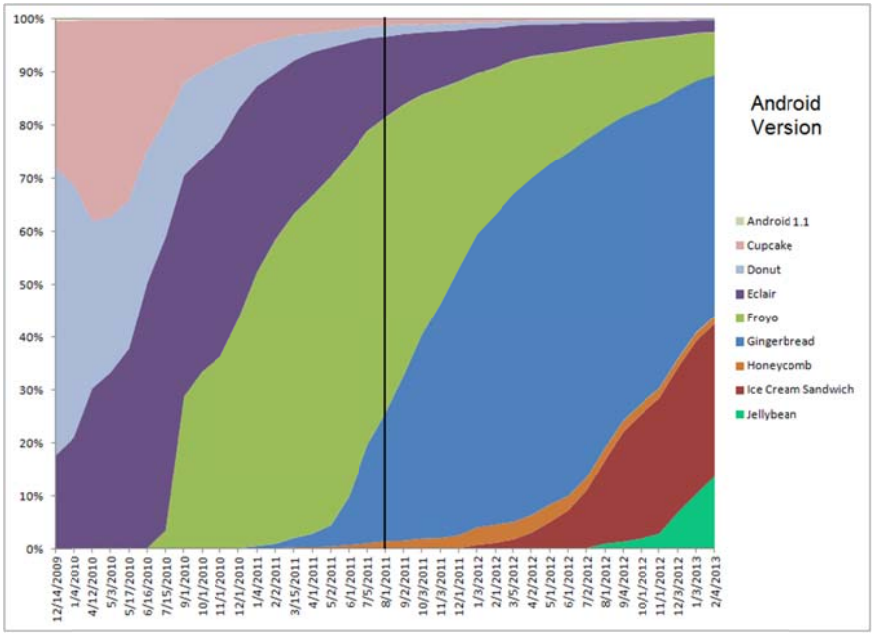

Fig 1.2 Evolució de les distribucions Android. Extret de [11].

CAPÍTOL 2. L'APLICACIÓ CUSTOMCAM

La capacitat de reproduir i capturar continguts multimèdia és una de les funcionalitats més demandades en la nova generació de telèfons mòbils. Ja sigui per diversió, o com a eina de treball, realment és molt còmode disposar d'un multimèdia complet (videocàmera, gravadora de so, reproductor de música, etc) integrat en un dispositiu de volum tan reduït.

Per aconseguir aquest objectiu, Android ofereix un *framework* multimèdia capaç de reproduir i capturar àudio, vídeo i imatges estàtiques en els formats més comuns. Aquest *framework* utilitza una llibreria escrita en C/C++, basada en el projecte OpenCORE de PacketVideo [17].

En aquest capítol veurem l'aplicació CustomCam, desenvolupada partint de l'anàlisi dels codificadors, contenidors i protocols de xarxa que suporta la plataforma; donada la limitació d'espai a la memòria, hem detallat les capacitats multimèdia d'Android [18] a l'annex 3.

L'aplicació CustomCam permet capturar imatges en quatre formats diferents, gravar so, capturar vídeo (diferents codificacions) i, per últim, fer *streaming* de vídeo "en viu" encapsulant el flux resultant del codificador en paquets RTP. A continuació detallen la estructura, els components i les funcionalitats de l'aplicació. Primerament la descriurem a nivell de vistes i recursos: el que l'usuari veu. Després, veurem com està estructurada: les activitats, classes i interfícies. Finalment, detallarem com fer servir l'aplicació i quina és la informació que emmagatzema, tant multimèdia com de configuració.

2.1. Recursos utilitzats

Les aplicacions Android disposen de certs recursos com ara: imatges, textos i *layouts*. Aquests elements són de fàcil accés des del codi i s'agrupen en carpetes dins del directori *res* (veure l'apartat A1.2 de l'annex 1).

A la carpeta *values* definim els textos que mostra l'aplicació en els diferents elements: `TextView`, `Button` i `Spinner`. Però també definim les paraules claus que farem servir per desar i recuperar les preferències d'usuari. Tots aquests valors es poden accedir des dels *layouts* utilitzant les etiquetes *@string* i *@array*. O bé es poden recuperar des del codi utilitzant la funció `getResources()`.

Fem servir els elements *string-array* per definir els següents llistats d'opcions: formats d'imatge, codecs de vídeo, resolucions de vídeo, temps de gravació i nombre de canals d'àudio.

Al treballar amb imatges com a recursos d'aplicació, la plataforma defineix tres carpetes: *drawable-hdpi*, *drawable-mdpi*, *drawable-ldpi*. D'aquesta forma

l'aplicació pot triar les imatges més òptimes en funció de la mida i la densitat de la pantalla del dispositiu on s'executa. En el nostre cas, l'aplicació fa servir imatges prou petites com per no haver d'optimitzar-les.

Finalment, a la carpeta *layout,* definim les vistes que carregarà l'aplicació. Aquestes pantalles es construeixen utilitzant elements que hereten de la classe `View`. La nostra aplicació presenta quatre vistes XML que definim a continuació:

`main.xml`

Aquesta és la primera pantalla de l'aplicació (figura 2.1), conté cinc elements `Button` que serveixen per triar la funcionalitat: càmera de fotos, càmera de vídeo, gravadora d'àudio o servidor d'*streaming*. Els botons estan posicionats de forma relativa en un element `RelativeLayout`.

Fig 2.1 Pantalla inicial de l'aplicació CustomCam

`customcam.xml`

Independentment de la funcionalitat triada sempre carreguem el *layout* `customcam.xml`, que conté un element `SurfaceView` per mostrar el preview de la càmera i un *layout* lineal on posarem, a través de codi i segons la funcionalitat, els botons de control: iniciar/parar gravació, capturar imatges o iniciar/parar el servidor d'*streaming*. L'element de superfície ocupa tota la pantalla, i el *layout* de control està posicionat a la part inferior dreta. La figura 2.2 mostra tres captures de pantalla d'aquest *layout* en les funcionalitats de càmera de fotos, gravadora de so i videcàmera.

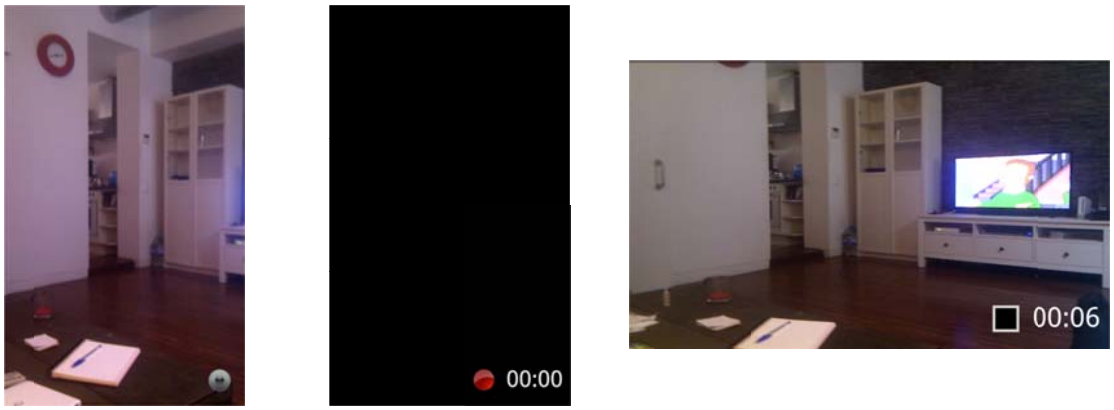

Fig 2.2 Pantalles de captura d'imatges, àudio i vídeo respectivament

`preferences.xml`

Aquest *layout* consta de l'element contenidor `PreferenceScreen`, el contingut es carrega des de codi i és diferent segons la funcionalitat. Es va decidir implementar-ho així per dos motius: per una banda, aprofitar una mateixa `Activity` on atendre les preferències de totes les funcionalitats de l'aplicació, i per altra banda, necessitem carregar les preferències de la càmera dinàmicament ja que depenen del dispositiu. A l'apartat 2.3 veurem amb més detall quines són i com es fan servir les preferències.

La figura 2.3 mostra tres captures de pantalla amb les preferències de les funcionalitats d'imatge, vídeo i *streaming*.

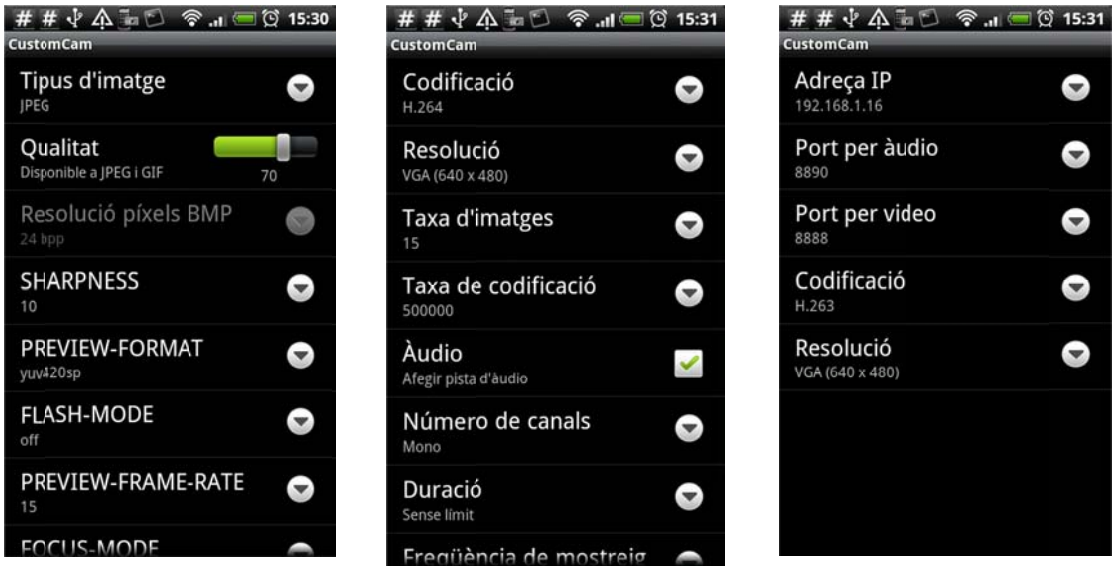

Fig 2.3 Preferències d'imatge, vídeo i *streaming* respectivament

`slider.xml`

La qualitat d'imatge, per la seva naturalesa és més entenedora si la mostrem com una barra progressiva entre dos valors. Fem servir aquest *layout* per mostrar propietats d'aquest tipus. La implementació està basada en un exemple de la pàgina web robobunny [19].

2.2. Estructura

Generalment, en una aplicació Android, cada pantalla correspon amb un element `Activity`. CustomCam fa servir una mateixa activitat per mostrar continguts diferents segons la funcionalitat triada per l'usuari. Es a dir, l'activitat `CustomCam` acull les funcionalitats: càmera de fotos, càmera de vídeo i gravadora de so. En aquest apartat es detalla el diagrama d'activitats de l'aplicació i les classes que intervenen en cada funcionalitat.

2.2.1. Diagrama d'activitats

L'aplicació CustomCam està formada per cinc activitats que detallem a continuació:

`Main.java`

És l'activitat principal, la que s'executa quan iniciem l'aplicació. Carrega la vista `main.xml` i implementa l'esdeveniment *click* en cadascun dels botons: actualitzar la propietat `APP_MODE` amb el valor de la funcionalitat triada i iniciar l'activitat corresponent amb la funció `startActivity()`.

Si la funcionalitat és fer fotos, capturar àudio o gravar vídeo s'inicia l'activitat `CustomCam`. En el cas del servidor d'*streaming* s'inicia l'activitat `Streaming` (o bé `Streaming_v2`).

`CustomCam.java`

Es iniciada per `Main`. Configura la orientació de la pantalla, el preview de la càmera, els controls i els esdeveniments *click* en funció del valor de la propietat `APP_MODE`. Les opcions del botó Menu permeten accedir a les preferències o bé obrir la galeria multimèdia llençant un `Intent` amb l'acció `ACTION_PICK` (a través de la funció `startActivityForResult()`).

El control de captura i reproducció per la càmera de fotos és un element `ImageView`. En el cas de la gravadora d'àudio o vídeo consisteix en un element `ImageView` i un `Chronometer` per visualitzar el temps de la gravació.

Aquesta activitat utilitza tres objectes per realitzar les tasques de gravació i captura, un per a cada funcionalitat: `AudioRecorder`, `TakePicture` i `VideoRecorder`. Veurem amb més detall aquests objectes en els propers apartats d'aquest capítol.

`Streaming.java`

Es iniciada per `Main`. Configura el botó que inicia i para el procés d'*streaming*. Les opcions del botó Menu permeten accedir a les preferències, on podem configurar l'adreça IP de destinació, els ports d'àudio i vídeo, el codificador de vídeo, la taxa d'imatges/segon i la taxa binaria (bit/s) del codificador de vídeo.

`Streaming_v2.java`

Aquesta semblant a `Streaming.java`, però fa servir un sistema de *sockets* i d'encapsuladors RTP diferent. A l'apartat 2.2.4 expliquem perquè hem implementat dos versions i quines són les diferències entre elles.

`Preferences.java`

Aquesta activitat hereta de la classe `PreferenceActivity`, es iniciada per `CustomCam`, per `Streaming` o per `Streaming_v2`. Mostra les preferències d'usuari corresponents amb la funcionalitat, en funció de l'atribut `APP_MODE`. Les preferències es guarden automàticament quan l'usuari modifica el valor de qualsevol d'elles.

La figura 2.4 mostra el diagrama d'activitats de l'aplicació CustomCam, hem omès l'activitat `Streaming_v2` ja que segueix la mateixa lògica que l'activitat `Streaming`.

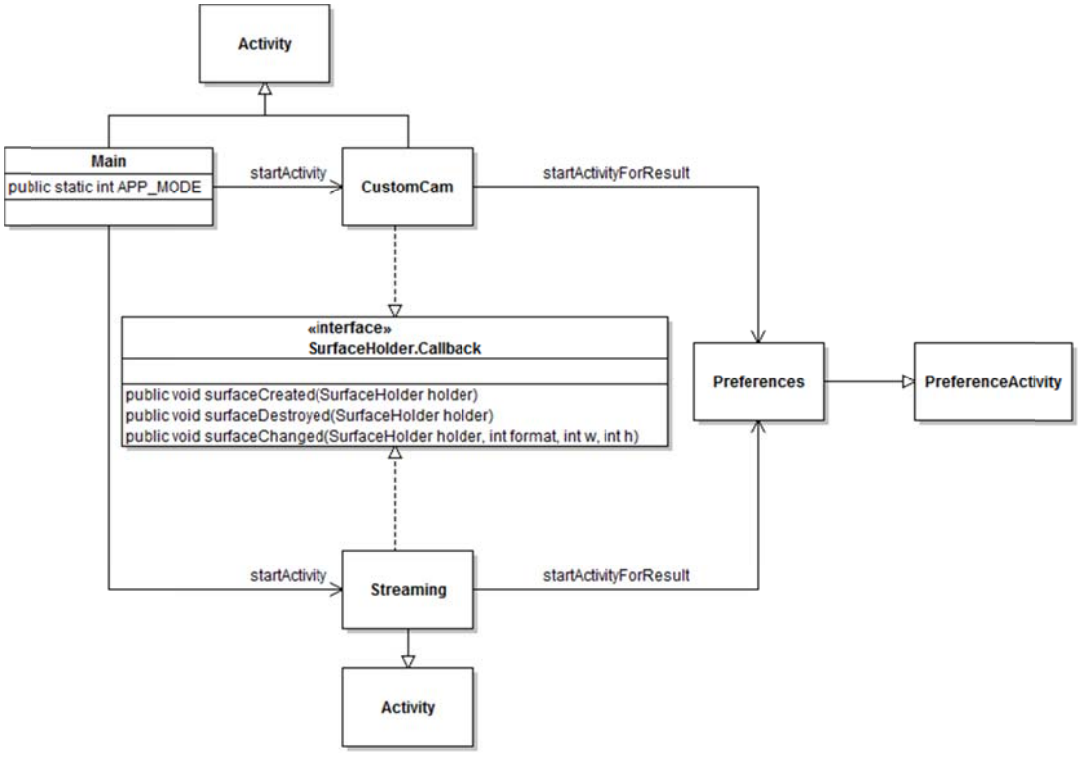

Fig 2.4 Diagrama d'activitats de l'aplicació CustomCam

2.2.2. Càmera de fotos

L'activitat `CustomCam` implementa la interfície `SurfaceHolder.Callback` (figura 2.4) necessària per rebre informació sobre l'estat i els canvis que es produeixen en la superfície on pintem el *preview*. En el següent exemple es mostra com crear la superfície i assignar-li un *callback* per rebre les notificacions.

```
surfaceView = (SurfaceView) findViewById(R.id.surfaceview);
surfaceHolder = surfaceView.getHolder();
surfaceHolder.addCallback(this);
surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

Al crear la superfície es crida el mètode `surfaceCreated()`, i després sempre es crida el mètode `surfaceChanged()`. Quan la superfície canvia fem les següents accions:

- Parar el *preview* de la càmera si està en funcionament.
- Indicar la orientació, afegir el nou *holder* i tornar a iniciar el *preview*.
- Aplicar els paràmetres segons les preferències d'usuari.
- Iniciar i registrar la funció de *callback* per enfocar automàticament, fent servir l'objecte `AutoFocusSensor`.

La classe `AutoFocusSensor` (figura 2.5) s'encarrega d'executar el mètode `autoFocus()` de la càmera quan el sensor acceleròmetre mesura canvis en la posició del telèfon, d'aquesta forma aconseguim que la càmera sempre estigui enfocada. Per realitzar aquesta tasca, l'objecte `AutoFocusSensor` implementa la interfície `SensorEventListener` (per rebre canvis en les mesures dels sensors) i la interfície `AutoFocusCallback` (per implementar la funció d'enfocament). automàtic).

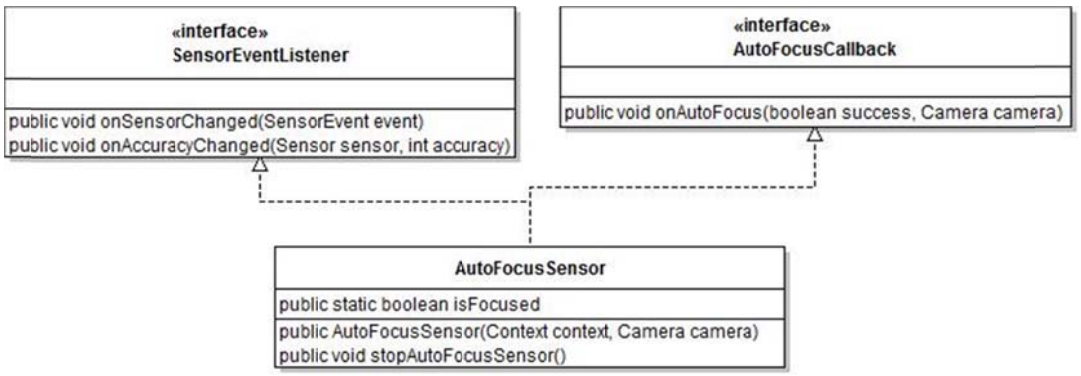

Fig 2.5 Implementació de l'enfocament automàtic

Quan es destrueix la superfície, així com quan l'activitat queda en pausa, parem el *preview* i alliberem la càmera perquè la pugui fer servir un altre procés.

La captura d'una imatge es llença des de l'esdeveniment *click* del botó de control i s'executen les següents accions:

- Cancel·lar el procés d'enfocament automàtic i parar el *preview*.
- Crear un objecte `TakePicture`.
- Cridar el mètode `takePicture()` de la càmera i passar l'objecte `TakePicture` com a funció de *callback*.

La classe `TakePicture` s'encarrega de codificar i guardar a la targeta SD la imatge capturada en el format triat a les preferències: JPEG, PNG, BMP o GIF.

Les imatges en format JPEG i PNG es codifiquen utilitzant la classe `Bitmap`, com veiem a continuació:

```
bitmap.compress(Bitmap.CompressFormat.JPEG, quality, outStream);
bitmap.compress(Bitmap.CompressFormat.PNG, quality, outStream);
```

La qualitat només es té en compte en imatges JPEG. El compressor PNG és sense pèrdues i ignora aquest valor.

Hem creat la classe `BMP` per guardar imatges en format BMP a partir d'un `Bitmap` en format `ARGB_8888` (quatre canals – BGR i transparència – codificats amb vuit bits cadascun; es a dir profunditat de píxel de 32 bits). Aquesta classe permet codificar imatges BMP amb precisions de 16, 24 o 32 bits per píxel. Per codificar imatges amb 24 bits de profunditat eliminem el canal de transparència, calculem el número de bytes de *padding* per fila (ja que el nombre de bytes en cada fila a de ser múltiple de 4). En el cas d'imatges de 16 bits per píxel, a part dels passos anteriors és necessari reduir el nombre de bits amb que es codifica cada canal; passem a sis bits per al canal verd (que és el color que l'ull humà percep millor) i cinc bits per als canals blau i vermell (RGB_565). Es fa servir la capçalera BMP de Windows (BITMAPINFOHEADER) de 54 bytes de longitud.

0	1	2	3
'B' (0x42)	'M' (0x4D)		
Mida de la imatge en bytes.			
Reservat (valor zero)		Reservat (valor zero)	
Número de bytes abans de la imatge (offset)			
Mida de la estructura BITMAPINFOHEADER en bytes (40 bytes)			
Amplada de la imatge en píxels.			
Alçada de la imatge en píxels.			
Número de plans (valor 1)		Bits per píxel (1, 4, 8, 16, 24 o 32)	
Tipus de compressió (0=sense compressió, 1=RLE-8, 2=RLE-4)			
Mida de la imatge en bytes (incloent padding)			
Resolució horitzontal en píxels per metre.			
Resolució vertical en píxels per metre.			
Número de colors a la imatge (o valor zero)			
Número de colors importants (o valor zero)			

Fig 2.6 Capçalera BMP (Windows)

Hem fet servir la NDK per compilar gifflen[2], una petita llibreria nativa que permet codificar imatges en format GIF, a partir d'un `Bitmap`. Gifflen fa una reducció de colors als 256 màxims que permet el format GIF, seleccionant els que són més freqüents a la imatge bitmap, i arrodonint a aquests els que no són tan freqüents.

Com podem veure a la figura 2.7, la classe `TakePicture` implementa les interfícies `PictureCallback` (per rebre les dades de la imatge capturada) i `ShutterCallback` (per indicar el so que es reproduirà en el moment de la captura). A través de les tres funcions natives definides al la llibreria *gifflen*, aquesta llibreria permet codificar un `Bitmap` en format `ARGB_8888` a una imatge GIF.

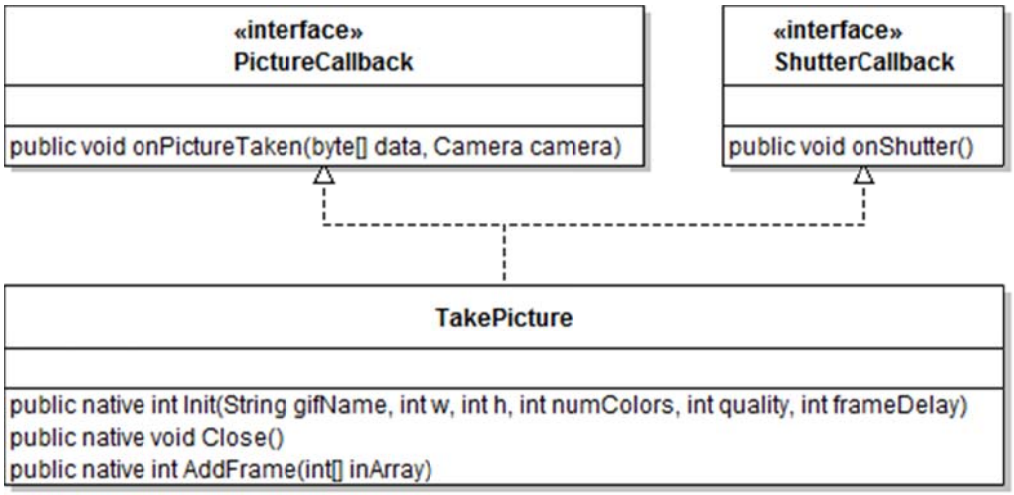

Fig 2.7 Diagrama de la classe `TakePicture`

2.2.3. Gravadora de so i càmera de vídeo

Per realitzar captures de vídeo i àudio s'han creat les classes `AudioRecorder` i `VideoRecorder`. Ambdues hereten de la classe `MediaRecorder`, l'objecte que proporciona la plataforma per poder capturar medis, i implementen la interfície `MediaRecorder.OnInfoListener` per rebre notificacions durant la captura.

L'activitat `CustomCam` fa servir l'objecte `AudioRecorder` en la funcionalitat gravadora de so, i l'objecte `VideoRecorder` en la funcionalitat càmera de vídeo. Aquest últim també es capaç de capturar àudio de forma opcional.

[2] Llibreria nativa gifflen http://jiggawatt.org/badc0de/android/index.html

La forma d'utilitzar aquests objectes és el següent:

- Crear una instància passant el context.
- Cridar la funció `startRecording()` de l'objecte. Per iniciar la font, el format i la sortida de la captura, i començar amb la gravació.
- Finalment, cridar el mètode `stopRecording()`. Per aturar i alliberar l'objecte `MediaRecorder` i afegir la gravació a la llibreria multimèdia, a través de la funció `addRecordingToMediaLibrary()`.
- Si hem limitat el temps màxim de duració de la gravació des de les preferències, el mètode `stopRecording()` es crida automàticament al rebre la notificació.

La figura 2.8 mostra el diagrama de les classes `AudioRecorder` i `VideoRecorder`.

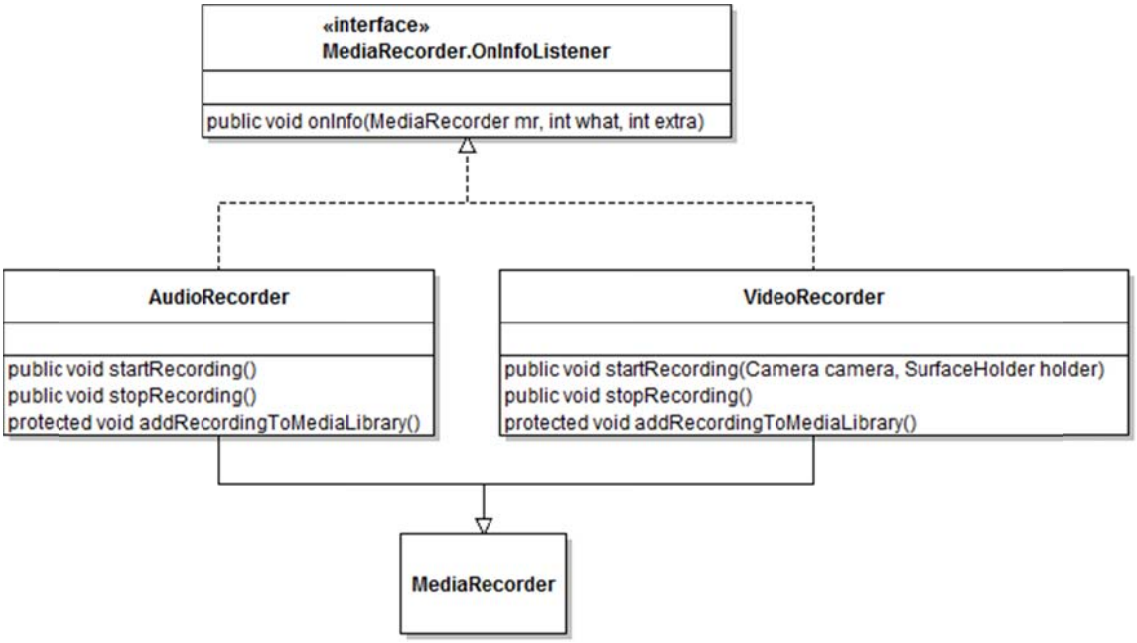

Fig 2.8 Diagrama de les classes `AudioRecorder` i `VideoRecorder`

2.2.4. Servidor d'*streaming*

La primera versió que hem implementat del servidor d'*streaming* està basada en el PFC de l'estudiant Simón Tabaré [20] de l'ETSETB. L'objectiu d'aquest projecte és transmetre, a través d'una xarxa Wi-Fi, els fluxos multimèdia capturats per un dispositiu mòbil. La solució proposada és empaquetar àudio (codificació AMR) i vídeo (codificació H.264) amb RTP, i enviar els dos fluxos per separat sobre *sockets* UDP.

En aquesta decisió es valora la simplicitat d'implementar la transmissió sobre UDP, creant un sistema no orientat a connexió, i la possibilitat de crear un

servei *multicast* sense haver de multiplicar els fluxos d'àudio i vídeo. Per contra, la transmissió de dades no és fiable. No obstant, considerem que en aquesta aplicació (com a totes les de tipus multimèdia) és més important una transmissió fluïda i en temps real que no pas fiable (sempre que les pèrdues estiguin acotades).

A la nostra aplicació CustomCam afegim la possibilitat d'escollir el codificador de vídeo: H.263, H.264 o MPEG-4. Per fer-ho s'han escrit les classes `H263Packetizer`, `H264Packetizer` i `MPEG4Packetizer`.

Les noves versions d'Android (nivell 10 de la API o superior) incorporen altres codificadors d'àudio (veure l'apartat A3.1 de l'annex 3) que haguéssim pogut incloure per enriquir la funcionalitat d'*streaming*. No obstant, desenvolupant amb la versió 8 no tenim accés a les noves parts de la API; per tant l'aplicació no està preparada per aprofitar les noves funcionalitats tot i fer-la córrer en un terminal actualitzat.

A continuació es descriuen les classes que conformen la primera versió del sistema d'*streaming*. Donada la limitació d'espai a la memòria i respectant l'ordre en que s'ha realitzat el desenvolupament, podem trobar el motiu i l'especificació del segon sistema d'*streaming* a l'annex 4.

`RTPPacketizer.java`

S'encarrega de crear el sistema de tres *sockets* locals per connectar el flux de sortida de la càmera o el micròfon (`MediaRecorder`) amb l'entrada del transmissor (`Sender`). També proporciona mètodes per treballar amb paquets RTP (crear capçalera, escriure o llegir bytes, control de número de seqüència, etc).

A la figura 2.9 es mostra la connexió *socket* emissor amb *socket* receptor a través del *socket* servidor. El flux de sortida de la càmera (o del micròfon) s'escriu al descriptor de fitxer del *socket* emissor. Durant el procés d'*streaming* llegirem aquest flux des del descriptor de fitxer del *socket* receptor, crearem el paquet RTP i l'afegirem a la cua d'enviament del `Sender`.

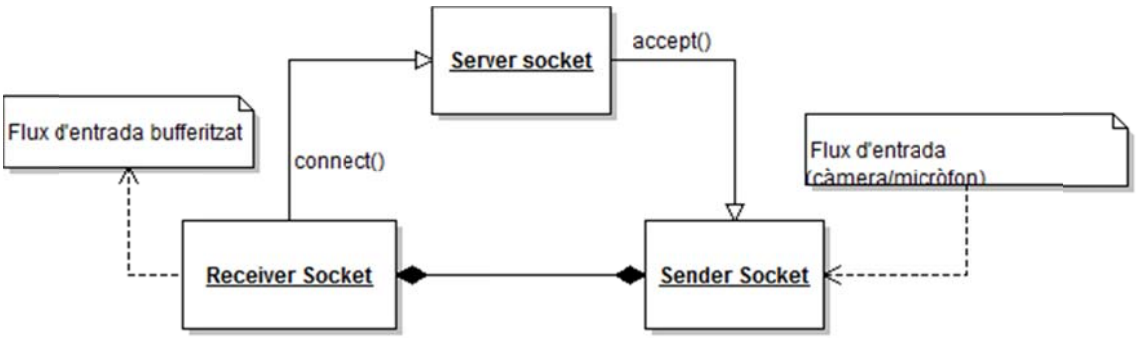

Fig 2.9 Diagrama de connexió de fluxos

El que aconseguim amb aquest sistema de *sockets* és *bufferitzar* el flux de sortida de la càmera i així garantir la integritat de les dades. Si no ho féssim, i el procés d'empaquetament fos més lent que el de captura, provocaria pèrdua d'informació.

Les classes `AMRPacketizer`, `H263Packetizer`, `H264Packetizer` i `MPEG4Packetizer` hereten d'aquesta classe.

`AMRPacketizer.java`

S'encarrega d'empaquetar el flux d'àudio codificat en paquets RTP seguint la recomanació RFC 3267 [21] i després els afegeix al *buffer* d'enviament del `Sender`. Implementa la interfície `Runnable` per ser executat en un procés diferent.

El *timestamp* RTP correspon a l'instant de mostreig de la primera mostra dins la primera trama que conté el paquet. En el cas d'AMR, cada paquet només conté una trama, llavors la unitat de mesura del *timestamp* són mostres.

A l'annex 5 hem fet un anàlisi teòric de la transmissió d'un flux d'àudio AMR encapsulat en paquets RTP, que després hem comparat amb dues captures Wireshark corresponents a dos trames AMR consecutives.

`H263Packetizer.java`

S'encarrega d'empaquetar el flux de vídeo codificat en H.263 en paquets RTP, seguint la recomanació RFC 4629 [22] i després els afegeix al *buffer* d'enviament del `Sender`. Implementa la interfície `Runnable` per ser executat en un procés diferent.

Cada trama H.263 comença amb un codi de 22 bits anomenat *Picture Start Code* (PSC = `0000 0000 0000 0000 1 00000`). La mida de *payload* dels paquets RTP és variable, una trama pot ser fragmentada en més d'un paquet RTP. De la mateixa manera, un paquet RTP no pot contenir més d'una trama H.263, o fragments de dues trames H.263.

Quan el paquet actual conté el final d'un frame H.263 cal posar a 1 el bit M (*Marker bit*) de la capçalera RTP. D'altra banda, el *timestamp* RTP està basat en un rellotge de 90 kHz. Es necessari que els diferents paquets que continguin parts d'una mateixa imatge tinguin el mateix *timestamp*.

`H264Packetizer.java`

S'encarrega d'empaquetar el flux de vídeo codificat en H.264 en paquets RTP, seguint la recomanació RFC 3984 [23] i després els afegeix al *buffer* d'enviament del `Sender`. Implementa la interfície `Runnable` per ser executat en un fil diferent.

El flux H.264 es divideix en unitats descodificables per si soles: *Network Abstraction Layer* (NAL) *units*. Abans de transmetre cada unitat NAL llegim la seva longitud, i si aquesta és més gran que la mida màxima del paquet, cal fragmentar-la. Per fer-ho cal afegir una capçalera Fragment Unit (FU) d'un byte, per indicar si és el primer o l'últim fragment de la unitat fragmentada. Els diferents paquets que contenen fragments d'una mateixa NAL han de tenir números de seqüència consecutius.

De la mateixa manera que amb H.263, cal indicar a la capçalera RTP el paquet que conté l'últim fragment de la NAL.

El *timestamp* RTP està basat en un rellotge de 90 kHz; si suposem una taxa de 25 imatges per segon a la sortida del codificador podem calcular el delta de *timestamp* en imatges consecutives.

$$\frac{90000\,ticks/segon}{25\,imatges/segon} = 3600\frac{ticks}{imatge}$$

Els diferents paquets amb fragments d'una mateixa imatge han de tenir el mateix *timestamp*.

2.3. Dades

L'aplicació CustomCam emmagatzema per una banda els fitxers multimèdia que genera i, per l'altre, les preferències de l'usuari que la fa servir.

2.3.1. Fitxers multimèdia

Amb la intenció d'agrupar els fitxers multimèdia generats per la nostra aplicació i distingir-los de la resta (descàrregues, fitxers generats per altres aplicacions, etc), tots els fitxers es guarden al directori CustomCam, creat a l'arrel de la targeta SD.

Dins del directori CustomCam separem en subcarpetes els diferents tipus de fitxers multimèdia. Així, trobarem els directoris *Audio*, *Imatge* i *Video*.

2.3.2. Preferències d'usuari

Implementem les preferències d'usuari fent ús de la classe `SharedPreferences`, que proporciona els mètodes per guardar i recuperar dades de tipus primitiu (*booleans, floats, ints, longs* i *strings*) en parelles (clau – valor), juntament amb la classe `PreferenceActivity`, que ofereix un marc de treball adequat per desenvolupar aquesta tasca.

Un conjunt d'aquestes preferències són els paràmetres ajustables de la càmera. Per recuperar-los fem servir la funció `flatten()` de l'objecte

`Camera.Parameters`, que retorna un `String` (en format clau – valor) amb els possibles valors i el valor actual de cadascun d'aquests paràmetres. Analitzant aquest text, generem un llistat dinàmic de preferències que varia en funció de les capacitats del terminal.

L'estructura de dades que defineix els possibles valors d'un paràmetre no segueix un únic patró. A continuació mostrem els tres patrons parsejats en la nostra aplicació.

La mida d'imatge defineix els possibles valors (d'amplada i d'alçada) separats per comes. El valor actual de qualsevol paràmetre el trobem darrera del nom d'aquest paràmetre i separat per el símbol "=".

```
picture-size=1024x768;
picture-size-values=2592x1952,2592x1728,
2592x1552,2560x1920,2560x1712,2048x1536,2048x1360,2048x1216,2016x1344,
1600x1200,1584x1056,1280x960,1280x848,1280x768,1248x832,1024x768,640x4
80,640x416,640x384,624x416,512x384,400x400,272x272;
```

En el cas del zoom de la imatge s'indica un valor màxim, un valor mínim i el valor actual. Això vol dir que el paràmetre pot adoptar un valor lliure entre els límits definits.

```
taking-picture-zoom=0;
taking-picture-zoom-min=0;
taking-picture-zoom-max=20;
```

La compensació de l'exposició es defineix amb un màxim, un mínim i un pas entre valors. En aquest cas, el conjunt de possibles valors del paràmetre va des de el mínim fins al màxim de forma discontinua (fent salts del valor del pas).

```
exposure-compensation=0;
max-exposure-compensation=4;
min-exposure-compensation=-4;
exposure-compensation-step=0.5;
```

La taula 2.1 mostra les preferències de l'aplicació CustomCam per cada funcionalitat. A l'annex 6 s'adjunta el text complet retornat per la funció `flatten()` en un dispositiu HTC Desire amb Android 2.2.

Taula 2.1 Preferències de l'aplicació

Funcionalitat	Preferència	Possibles valors	Valor per defecte
Àudio	Número de canals	Mono (1 canal)	Mono
		Stereo (2 canals)	
	Durada de la gravació	Sense límit	Sense límit
		30 segons	
		1 minut	
		5 minuts	
		10 minuts	
	Freqüència de mostreig	En funció del codec: AAC (8 – 96 kHz), AMR-NB (8 kHz) i AMR-WB (16 kHz).	8 kHz
	Taxa de codificació	Valor lliure	128 kbps
Imatge	Tipus d'imatge	JPEG	JPEG
		PNG	
		GIF	
		BMP	
	Qualitat (només per JPEG i GIF)	Valor entre 1 i 100	75
	Paràmetres de la càmera	En funció del dispositiu	En funció del dispositiu
Vídeo	Codificació	H.263	H.264
		H.264	
		MPEG-4	
	Resolució	QCIF (176x144)	VGA
		QVGA (320x240)	
		CIF (352x288)	
		VGA (640x480)	
		WVGA (800x480)	
		720P (1280x720)	
	Taxa d'imatges	Enter (entre 1 i 30)	15 imatges/segon
	Taxa de codificació	Valor lliure	2 Mbps
Streaming (les dues versions)	Adreça IP	Adreça IPv4 vàlida	192.168.1.12
	Port àudio	Enter (entre 0 i 65535)	8890
	Port vídeo	Enter (entre 0 i 65535)	8888
	Codificació	H.263	H.264
		H.264	
		MPEG-4 (no implementat)	

CAPÍTOL 3. GEOLOCALITZACIÓ EN APLICACIONS ANDROID

En aquest capítol es detalla com recuperar i representar la localització del dispositiu des d'una aplicació. Aquest és un aspecte necessari de cara al desenvolupament de l'aplicació de captura geolocalitzada de trànsit, i per la seva entitat i interès es descriu en un capítol separat. A l'últim apartat s'explica un petit experiment que es va fer per mesurar el ritme d'actualitzacions que proporciona el servei de localització.

Durant tot el capítol es fa referència a les coordenades geogràfiques: latitud i longitud. Aquestes ens permeten definir, amb precisió, la ubicació d'un punt qualsevol a la superfície de la terra.

a) La latitud és la distància angular (normalment en graus) entre qualsevol punt i l'equador, mesurada per el meridià que passa per aquest punt.

b) La longitud és la distància angular (normalment en graus) entre qualsevol punt i el meridià zero (*Greenwich*), mesurada pel paral·lel que passa per aquest punt.

3.1. Obtenir les dades

Actualment, amb la nova generació de telèfons mòbils intel·ligents (*smartphones*), els usuaris demanen aplicacions potents als seus terminals. Ja sigui un sistema de navegació GPS per al cotxe, o un cercador de cinemes, es fa imprescindible poder determinar de forma senzilla i precisa la ubicació tant del terminal com dels elements d'interès que estan a propers a ell.

Android ofereix els mètodes i eines necessaris per obtenir les dades d'ubicació d'un terminal des de qualsevol aplicació. La informació d'ubicació és mesurada pels proveïdors de servei i inclou latitud, longitud, velocitat, localització, altitud i precisió entre d'altres paràmetres.

Les classes i interfícies Java esmentades en aquest apartat i totes les relacionades amb la ubicació es troben al *package* `android.location`. El component central d'aquest paquet és la classe `LocationManager`.

3.1.1. Proveïdors de servei: LocationProvider

A l'hora de determinar la ubicació d'un terminal tenim la possibilitat d'escollir a quina o quines fonts d'informació interroguem. La plataforma Android defineix tres tipus de proveïdors: GPS, Xarxa i Passiu. Per accedir a cadascun d'ells ho fem a través de la classe `LocationManager`*:*

```
LocationManager.<Nom del proveïdor>
```

A continuació es detallen les característiques més significatives de cada proveïdor.

- Proveïdor de GPS: Descrit per la constant `GPS_PROVIDER`. Permet determinar la ubicació a través dels satèl·lits GPS. En funció de les condicions geogràfiques és possible experimentar un retard abans de rebre les coordenades. Requereix del permís `ACCESS_FINE_LOCATION`.

- Proveïdor de Xarxa: Descrit per la constant `NETWORK_PROVIDER`. Permet determinar la ubicació a través de les antenes de telefonia mòbil o els punts d'accés Wi-Fi disponibles. Requereix del permís `ACCESS_FINE LOCATION` o bé del `ACCESS_COARSE_LOCATION`.

  Generalment, les localitzacions calculades amb triangulació d'antenes de telefonia mòbil no són precises. Per millorar el servei en aplicacions basades en localització, els cotxes de *Google Street View* [12] guarden la informació Wi-Fi difosa per els punts d'accés que detecten al seu pas. Aquesta informació permet determinar, amb molta més precisió, la ubicació d'un dispositiu.

- Proveïdor Passiu: Descrit per la constant `PASSIVE_PROVIDER`. Permet rebre actualitzacions d'ubicació sense demanar-les. Es a dir, rep de forma passiva les actualitzacions d'ubicació que sol·liciten altres aplicacions. Requereix del permís `ACCESS_FINE_LOCATION`.

  Google Latitude – "descobreix on són els teus amics" – és un servei basat en localització passiva, que aconsegueix mantenir actualitzada la ubicació d'un usuari sense accelerar el temps de descàrrega de la bateria. Cada vegada que un altre aplicació demana la localització aquest servei s'actualitza en segon pla [13].

3.1.2. Accés al servei: LocationManager

La classe `LocationManager` proporciona accés als serveis d'ubicació del sistema. Per recuperar l'administrador del servei ho fem a través de la funció *`getSystemService(Context.LOCATION_SERVICE)`*, ja que no és possible crear instàncies de la classe `LocationManager`.

```
LocationManager locManager =
  (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

L'objecte `LocationManager` permet consultar l'última ubicació coneguda, determinar quins proveïdors de servei estan disponibles, registrar un `LocationListener` per a rebre actualitzacions periòdiques d'ubicació o, fins i tot, registrar una alerta de proximitat a una certa ubicació. En el proper apartat es dona més informació sobre el model `LocationListener`.

3.1.2.1. Registrar `LocationListener`

És possible demanar actualitzacions d'ubicació de varies maneres: de forma periòdica o un únic cop, definir criteris per escollir el proveïdor adient, definir l'acció a realitzar al rebre l'actualització, indicar el període mínim o la distància mínima entre actualitzacions. En aquest exemple, registrem un *listener* per rebre actualitzacions d'ubicació de forma periòdica, basades en el proveïdor GPS, i sense limitació, de temps o distància, entre actualitzacions. Es a dir, rebrem el màxim d'actualitzacions possibles.

```
locManager.requestLocationUpdates(
  LocationManager.GPS_PROVIDER, 0, 0, locationListenerImpl);
```

Després d'aquesta instrucció, el servei ens enviarà actualitzacions d'ubicació que podrem gestionar des de l'objecte `locationListenerImpl`. És important, de cara a estalviar energia, eliminar tots els *listeners* un cop hem acabat de tractar les dades.

```
locManager.removeUpdates(locationListenerImpl);
```

3.1.2.2. Última ubicació coneguda

Per recuperar la darrera ubicació obtinguda per un proveïdor podem fer servir la funció `getLastKnowLocation()`, com mostra el següent exemple:

```
locManager.getLastKnowLocation(LocationManager.GPS_PROVIDER);
locManager.getLastKnowLocation(LocationManager.NETWORK_PROVIDER);
```

Cal tenir en compte que el valor retornat pot no correspondre amb el valor d'ubicació actual. El programador ha de determinar la validesa d'aquesta posició, avaluant la diferència entre la marca de temps de l'objecte `Location` i l'instant actual. Considerem que una diferència de dos minuts és suficient gran com per descartar la posició.

3.1.3. Actualitzacions d'ubicació: `LocationListener`

Cal implementar `LocationListener` per rebre actualitzacions periòdiques d'ubicació. La interfície presenta quatre mètodes, tres dels quals fan referència a l'estat i els canvis que es produeixen en la connexió amb el proveïdor de servei. El mètode restant s'executa quan les dades de localització canvien, es a dir, cada cop que rebem una actualització d'ubicació.

El següent codi és una possible implementació (sense funcionalitat) de la interfície `LocationListener`:

```
public class LocationListenerImpl implements LocationListener {

      @Override
      public void onProviderDisabled(String provider) { }

      @Override
      public void onProviderEnabled(String provider) { }

      @Override
      public void onStatusChanged(String provider, int status,
       Bundle extras) { }

      @Override
      public void onLocationChanged(Location location) { }
}
```

El primer mètode, `onProviderDisabled(String provider)`, ens serveix per avisar a l'usuari que no podem seguir recuperant informació de localització. També podem aprofitar per comprovar si està disponible un altre proveïdor.

Després de la crida a `onProviderEnabled(String provider)` podem iniciar l'escolta d'actualitzacions periòdiques d'aquest proveïdor.

Es crida `onStatusChanged(String provider, ...)` quan no es possible trobar una ubicació o si un proveïdor torna a estar actiu després d'un període d'inactivitat.

Per últim, el mètode `onLocationChanged(Location location)` s'executa al rebre una actualització d'ubicació.

Seguidament ens centrarem en els formats de les dades recuperades: la classe `Location`. També veurem com interpretar aquesta informació amb les classes `Geocoder` i `Address`.

3.1.4. Format de les dades: `Location`, `Geocoder`, `Address`

La classe `Location` representa una ubicació geogràfica detectada en un instant de temps determinat. Ho podem veure com una "foto" amb informació geogràfica i temporal, que consta de latitud, longitud, una marca de temps UTC i, de forma opcional, informació sobre la altitud, la velocitat i el rumb.

Android defineix la precisió de les ubicacions com una probabilitat del 68% dins d'un radi. Es a dir, si dibuixem una circumferència centrada a la latitud i longitud de la localització, amb radi igual al valor de la precisió, hi ha un 68% de probabilitat que la localització real estigui dins d'aquesta circumferència. En termes estadístics, equival a suposar que els errors comesos, al determinar

una localització, segueixen una distribució normal (campana de Gauss), de manera que el cercle de confiança del 68% representa una desviació estàndard [14].

Taula 3.1. Funcions més rellevants de la classe `Location`

Descripció	Tipus de dades	Funció
Precisió estimada, en metres.	`Float`	`getAccuracy()`
Altitud de la ubicació (en metres)	`Double`	`getAltitude()`
Desplaçament en graus a l'est respecte al nord	`Float`	`getBearing()`
Informació extra específica del proveïdor	`Bundle`	`getExtras()`
Latitud de la ubicació	`Double`	`getLatitude()`
Longitud de la ubicació	`Double`	`getLongitude()`
Nom del proveïdor	`String`	`getProvider()`
Velocitat en metres per segon	`Float`	`getSpeed()`
Temps UTC en segons (des de l'1 de gener de 1970).	`Long`	`getTime()`

Els objectes de tipus `Geocoder` permeten transformar l'adreça o descripció d'un lloc en coordenades (latitud i longitud); és el que s'anomena geocodificació. A més, també permeten fer el procés invers, es a dir, a partir de les coordenades de localització retornen una adreça.

La classe `Geocoder` depèn d'un servei extern, no inclòs al nucli d'Android, per realitzar el procés de geocodificació [15]. *Google APIs Add-On* és una extensió de l'entorn de desenvolupament d'Android que proporciona una implementació d'aquest servei. Per altre banda, el mètode `isPresent` permet verificar que existeix la implementació dels mètodes necessaris: `getFromLocation` i `getFromLocationName`.

Els mètodes `getFromLocation` i `getFromLocationName` permeten recuperar un llistat d'objectes `Address` que compleixin amb la localització (latitud i longitud) o la descripció indicades, respectivament.

El següent exemple mostra com recuperar, amb la classe `Geocoder`, un llistat d'objectes `Address` a partir d'una descripció. El llistat recuperat, com a màxim de cinc elements, pot contenir adreces de diferents ciutats del món, ja que la descripció utilitzada no és gaire específica.

```
int MAX_RESULTS = 5;
List<Address> address = null;
String DESC = "Carrer Mallorca 401";

Geocoder mGeocoder = new Geocoder(this);

if (mGeocoder.isPresent()) {
    address = mGeocoder.getFromLocationName(DESC, MAX_RESULTS);
    // Començar a treballar amb el llistat d'adreces
}
```

La classe `Address` representa la descripció d'un lloc concret (o adreça) en una versió simplificada de xAL (*eXtensible Address Language*) [16]. La taula en 3.2 recull els mètodes més rellevants.

Taula 3.2. Mètodes més rellevants de la classe `Address`

Descripció	Tipus de dades	Funció
Latitud de l'adreça. Cal fer-la servir juntament amb la funció `hasLatitude()`.	`Double`	`getLatitude()`
Longitud de l'adreça. Cal fer-la servir juntament amb la funció `hasLongitude()`.	`Double`	`getLongitude()`
Línea de l'adreça indicada per `index`. Cal fer-la servir juntament amb la funció `getMaxAddressLineIndex()`.	`String`	`getAddressLine(int index)`
Nom del país localitzat per l'adreça.	`String`	`getCountryName()`
Nom característic de l'adreça.	`String`	`getFeatureName()`
Nom de la localitat (província) de l'adreça.	`String`	`getLocality()`
Codi postal de l'adreça.	`String`	`getPostalCode()`

Continuant amb l'exemple anterior, i per enllaçar amb la classe `Location`, el següent algoritme omple un nou llistat, d'elements `Location`, inicialitzats amb les dades dels objectes Address.

```
ArrayList<Location> mLocations = new ArrayList<Location>();
Iterator<Address> it = address.iterator();

while (it.hasNext()) {
    Address addr = it.next();
    Location location = new Location(LocationManager.GPS_PROVIDER);

    location.setLatitude(addr.getLatitude());
    location.setLongitude(addr.getLongitude());

    mLocations.add(location);
}
```

3.2. Representar la informació

La forma més clara de representar les dades d'ubicació és sobre un mapa. En aquest treball hem decidit utilitzar Google Maps com a proveïdor de mapes per dues raons: és una llibreria completament integrada a la plataforma Android i compta amb una documentació extensa.

A continuació veurem dos possibles maneres d'utilitzar els mapes de Google: la primera, i més senzilla, és llençar un `Intent` per invocar l'aplicació desitjada, la

segona és utilitzar el complement Google APIs per encastar un element `MapView` a la nostra aplicació.

3.2.1. Invocar aplicacions

Android permet, a través de la classe `Intent`, delegar certa feina a un altre aplicació capaç de realitzar-la. Generalment per descriure un `Intent` s'indica una acció i una URI. Donada la limitació d'espai al cos d'aquesta memòria podem trobar més detalls sobre la classe `Intent` i els mètodes on s'utilitza a l'annex 2.

A la pàgina oficial per a desenvolupadors Android [9] es defineix un llistat d'*Intents* que la nostra aplicació pot enviar. Així, podem demanar l'execució d'aplicacions Google en el nostre projecte.

3.2.1.1. Aplicació Google Maps

Imaginem que hem encès el GPS a la nostra aplicació i hem recuperat les coordenades d'ubicació. Ara, volem veure la nostra localització sobre un mapa. El següent codi mostra com invocar l'aplicació *Maps* a partir de les coordenades *latitude* i *longitude:*

```
String intentUri = String.format("geo:%f,%f", latitude, longitude);
Intent mapIntent = new Intent(ACTION_VIEW, Uri.parse(intentUri));
startActivity(mapIntent);
```

Maps també es pot invocar fixant una adreça o descripció de la ubicació. El format de la petició es el següent:

```
geo:0,0?q=my+street+address
```

3.2.1.2. Aplicació Google StreetView

Invocar l'aplicació *StreetView* requereix un procediment molt semblant. El canvi més rellevant és el format de la petició:

```
google.streetview:cbll=lat,lon&cbp=1,yaw,,pitch,zoom&mz=mapZoom
```

Els paràmetres requerits son: latitud (*lat*) i longitud (*lon)*; la resta defineixen la finestra *StreetView* i el zoom inicial.

3.2.2. Google APIs

El complement Google APIs és una extensió del SDK d'Android, la seva funcionalitat principal és la llibreria externa *Google Maps*. Fent ús d'aquesta llibreria podem incloure elements `MapView` a la nostra aplicació i aconseguir, així, un control més precís dels mapes i una millor experiència per l'usuari.

Per poder fer servir `MapView` descarreguem l'extensió *Add-on* a través de *Android SDK and AVD Manager* i la instal·lem. Després, cal generar una empremta digital MD5 del certificat que farem servir per signar les nostres aplicacions. Finalment, hem de registrar la marca digital a *Google* per sol·licitar una clau - *Maps API Key* - que inclourem a l'element `MapView`.

```
<com.google.android.maps.MapView
   android:id="@+id/mapview"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:clickable="true"
   android:apiKey="Maps API Key" />
```

3.3. Freqüència d'actualitzacions

Un dels objectius d'aquest treball és geolocalitzar el trànsit de xarxa, indicant la ubicació on s'ha capturat cada paquet. En aquest procés és fonamental conèixer amb quina freqüència rebem actualitzacions d'ubicació.

Per determinar aquesta freqüència farem un petit anàlisi de les capturades per una aplicació que emmagatzema les actualitzacions GPS i Network.

3.3.1. Funcionament

L'aplicació presenta un element `Activity` que carrega una pantalla amb tres elements `Button` (Start, Stop i Reset) i dos `TextView` per mostrar els resultats "en viu". El botó Start crea un servei (component `Service`) que s'encarrega de rebre les actualitzacions de localització i d'escriure dos fitxers de text amb aquesta informació. El botó Stop elimina els *listeners* i para el servei. El botó Reset elimina els dos fitxers de dades. Cal eliminar-los en cada nou anàlisi, per a que no afecti als resultats.

A l'iniciar el servei, recuperem l'element `LocationManager` i registrem un `LocationListener` per rebre les actualitzacions del proveïdor GPS i un altre per rebre actualitzacions Network (apartat 3.1). Quan registrem els *listeners* indiquem que el temps mínim entre actualitzacions sigui cero; d'aquesta manera aconseguim rebre actualitzacions a la màxima freqüència possible.

Cada nova localització queda marcada en els fitxers de dades, per poder analitzar els resultats posteriorment. Per altre banda, l'activitat conté un procés `Runnable` que s'encarrega de refrescar la pantalla cada cinc segons amb la informació dels fitxers.

3.3.2. Fitxers generats

Quan rebem una nova localització s'actualitzen els dos fitxers de dades. En el fitxer *updates* s'afegeix una nova entrada al final amb la següent informació:

- Nom del proveïdor (GPS/Network)
- Instant de temps (en mil·lisegons)
- Latitud
- Longitud
- Altitud
- Precisió
- Velocitat

En el fitxer *statistics* només s'hi guarda informació del número d'actualitzacions rebudes en el període d'activitat del servei. Aquestes dades es resumeixen en una línia de text amb el següent format:

```
<Instant_inicial_(ms)>;<Instant_actual_(ms)>;<N_Actualitzacions_GPS>;
<N_Actualitzacions_Network>
```

3.3.3. Prova i conclusions

L'objectiu d'aquesta prova és verificar que la freqüència d'actualitzacions és suficient elevada com per determinar a quina localització es capturen els paquets de xarxa.

Vam fer córrer l'aplicació 891,92 segons, en un trajecte a peu al barri de Sant Martí, al voltant dels carrers: Ciutat de Granada, Escultors Claperos i Avinguda Meridiana. Durant aquest període de temps es reben 888 actualitzacions basades en GPS i 23 en antenes de telefonia mòbil o punts d'accés Wi-Fi. La prova es va realitzar amb un dispositiu HTC Desire amb versió d'Android 2.2.

Des que l'aplicació inicia el servei, instant inicial, fins que rep la primera actualització GPS passen 10 segons. Analitzant el fitxer de *log* de l'aplicació, calculem que la diferència de temps entre la primera actualització GPS i la última són 889,94 segons. Això vol dir que, un cop sincronitzat el senyal dels satèl·lits rebem, en mitja, una actualització GPS cada segon.

Per altra banda, el temps en rebre la primera actualització Network és menor, entre un i dos segons. La diferència de temps entre la primera i la última actualització en aquest cas són 889,042 segons. Llavors podem dir que, en mitja, rebem una actualització Network cada 38 segons, tot hi que aquestes actualitzacions no arriben de forma periòdica. Durant la prova el temps màxim entre actualitzacions Network és de 73 segons.

No obstant, la diferència màxima entre dos actualitzacions consecutives es de 2,408 segons, i és un cas aïllat. La resta d'actualitzacions es reben amb períodes de temps inferiors als dos segons. Amb aquesta dada i després de veure el resultat de representar en a Google Maps les 911 localitzacions capturades (figura 3.1), podem afirmar que la freqüència és suficient elevada com per determinar amb precisió a quina ubicació capturem els paquets de xarxa.

Fig 3.1 Representació a Google Maps de les 911 ubicacions capturades durant la prova.

CAPÍTOL 4. CAPTURA DE TRÀFIC DE XARXA

El fet de conèixer el volum i la distribució del tràfic de xarxa d'un conjunt gran d'usuaris, no només en funció del temps (com és habitual) sinó també de l'espai, permetrà fer anàlisis de caracterització de xarxes per optimitzar la planificació i la implantació d'estacions de telefonia i Wi-Fi.

Volem geolocalitzar el tràfic de xarxa generat per els dispositius mòbils, i per fer-ho, aprofitarem el servei de localització (GPS i Network) del sistema operatiu Android i farem ús de la eina de captura de paquets tcpdump.

Fent una cerca de projectes que vulguin fer el mateix hem trobat NTS (*Network Security Toolkit*), un *live* CD basat en Linux on es recullen les eines de software lliure més potents en seguretat i xarxa; el projecte permet, entre altres coses, geolocalitzar les converses sobre adreces IPv4 [24], no obstant està orientat sobretot a monitorització i administració de xarxes. No hem trobat cap projecte Android enfocat a geolocalització de tràfic.

Seguidament detallem les eines de captura i anàlisi de trànsit de xarxa utilitzades en el projecte, i els passos a seguir per poder utilitzar-les en un dispositiu Android. El capítol s'estructura en dos parts principals: **captura de paquets amb tcpdump** i **anàlisi del tràfic capturat amb jNetPcap**.

4.1. Tcpdump

Tcpdump és un potent analitzador de paquets que s'executa a través de consola (o línia de comandes). És software lliure, distribuït sota la llicència BSD simplificada[3]. Típicament, la majoria de distribucions Unix inclouen el paquet binari *tcpdump*. Aquest analitzador utilitza la llibreria *libpcap*, escrita en C/C++, per capturar el tràfic emès o rebut a través de la xarxa.

El nucli del sistema operatiu Android no inclou l'executable *tcpdump*; per fer-lo servir haurem de seguir els següents pasos.

1. Copiar el fitxer binari a la carpeta *raw*, dins del directori de recursos (detallem el directori de recursos a l'apartat A1.2 de l'annex 1).

2. Ser superusuari del dispositiu on executem l'aplicació basada en *tcpdump*.

3. Instal·lar l'executable en el directori de dades que Android reserva per cada aplicació.

[3] The BSD 3-Clause http://opensource.org/licenses/BSD-3-Clause

4.1.1. Permís superusuari

La majoria de fabricants i operadores implementen certa seguretat en els dispositius que venen, impedint que els usuaris puguin fer modificacions en el sistema que provoquin la inutilització del dispositiu. No obstant, aquesta limitació és un obstacle per a usuaris especialitzats que necessiten realitzar tasques d'investigació o anàlisi que requereixen permisos elevats.

El procés per obtenir aquests permisos no segueix un manual estàndard, ja que depèn del dispositiu que es vulgui "rootejar" i es resumeix en aconseguir afegir l'aplicació *su* al nostre dispositiu. Aquesta aplicació serà l'encarregada de notificar, acceptar i denegar les peticions de permisos superusuari que demanin altres aplicacions. Cal destacar que aquest procés pot ser perillós, i és recomanable estar segur de les operacions que fem, ja que existeix el risc de deixar el dispositiu inservible.

En aquest projecte vam adquirir privilegis de superusuari en un mòbil HTC Desire A8181, seguint les instruccions de la pàgina AlphaRev [25], que explica com desactivar la seguretat que implementa HTC en la memòria flash dels dispositius.

La figura 4.1 mostra la aplicació superusuari, que permet concedir o denegar permisos elevats a la resta d'aplicacions que ho demanin. En l'exemple prohibim fer servir permisos de superusuari a les aplicacions d'exploració de fitxers.

Fig 4.1 Aplicació permisos superusuari (AlphaRev)

4.1.2. Instal·lació del binari

El projecte `RootTools` presenta un conjunt d'eines per realitzar accions que requereixen permisos elevats com instal·lar binaris, executar instruccions com a superusuari o muntar i desmuntar una partició [26].

El següent exemple mostra com utilitzar `RootTools` per desempaquetar el paquet *tcpdump*, des de la carpeta *raw* al directori de dades reservat per a l'aplicació (`/data/data/app.package/files/`).

```
if (RootTools.installBinary(Main.this,
     R.raw.tcpdump, "tcpdump") == false) {
  // Error en la instal·lació
}
```

4.1.3. Executar l'analitzador

Un cop tenim instal·lat *tcpdump* el podem executar des de consola amb permisos superusuari. Per fer aquesta tasca ens basem en el projecte `Shark` [27], un capturador de tràfic desenvolupat per l'estudiant de l'EETAC Sergio Jiménez Feijóo com a pràctiques d'empresa, que a més permet visualitzar certa informació dels paquets capturats, com veurem amb més detall a l'apartat 4.2.

Per engegar el procés de captura de tcpdump necessitem indicar la interfície de xarxa que escoltarem. `Shark` inclou una classe que recupera el llistat d'interfícies disponibles, analitzant la informació retornada al executar en un terminal la comanda *netcfg*.

El següent exemple mostra com llençar el procés *tcpdump* en segon pla, amb la configuració passada per paràmetre. La classe `TCPdump` és la encarregada d'executar aquesta instrucció a la consola.

```
protected static final String tcpdumpBinaryPath =
  "/data/data/com.upc.locatedtraffic/files/tcpdump";

public int start(String params) {
  int r;
  if ((r = openShell()) != 0) {
    return r;
  }
  return runCommand(tcpdumpBinaryPath + " " + params + "&");
}
```

El paràmetre que rep la funció `start()` conté les opcions de configuració de *tcpdump* [28]; com són el nom de la interfície, el nivell de detall d'informació en els paquets capturats, el nom del fitxer de sortida o el format dels *timestamps*, entre d'altres. Una possible configuració és la que es mostra a continuació.

```
String params = "-i eth0 -p -vvv -ttt -w /sdcard/captura.pcap";
```

4.2. jNetPcap

`jNetPcap` és una llibreria de codi obert escrita en Java. Permet accedir a la majoria de funcions de la llibreria nativa *libpcap*, inclou un marc de treball per descodificació dels paquets capturats en temps real i proporciona una àmplia

llibreria de protocols de xarxa [29]. El projecte ofereix aquestes funcionalitats a través de dinou paquets descrits en la següent taula.

Taula 4.1 Descripció dels *package* del projecte `jNetPcap` versió 1.3

Nom del paquet	Descripció
`org.jnetpcap`	Funcions del nucli de libpcap, disponible a totes les plataformes.
`org.jnetpcap.nio`	Control de memòria nativa i classes per administrar l'entrada i sortida de dades.
`org.jnetpcap.packet`	Marc de treball per descodificació de paquets.
`org.jnetpcap.packet.annotate`	Interfícies d'anotació per la definició de capçaleres.
`org.jnetpcap.packet.format`	Format de classes per als objectes `JPacket` i `JHeader`.
`org.jnetpcap.packet.structure`	Definició de l'estructura dels primitius `Packet`, `Header`, `Field`, `Binding` i `Scanner`.
`org.jnetpcap.protocol`	Protocols de xarxa i definició de capçaleres.
`org.jnetpcap.protocol.application`	Conjunt de protocols d'aplicació.
`org.jnetpcap.protocol.lan`	Conjunt de protocols LAN.
`org.jnetpcap.protocol.network`	Conjunt de protocols de xarxa.
`org.jnetpcap.protocol.tcpip`	Conjunt de protocols TCP/IP.
`org.jnetpcap.protocol.voip`	Conjunt de protocols de veu sobre IP.
`org.jnetpcap.protocol.vpn`	Conjunt de protocols VPN.
`org.jnetpcap.protocol.wan`	Conjunt de protocols WAN.
`org.jnetpcap.util`	Mètodes de suport.
`org.jnetpcap.util.checksum`	Algoritmes CRC.
`org.jnetpcap.util.config`	Propietats de configuració de la SDK.
`org.jnetpcap.util.resolver`	Resolució d'adreces en llenguatge humà.
`org.jnetpcap.util.winpcap`	Extensió de *libpcap* per certes plataformes.

L'ús de `jNetPcap` en dispositius Android actualment es troba en fase experimental i només permet la descodificació de paquets en mode *offline*.

Va ser l'autor de l'aplicació Shark, Sergio Jiménez, qui va aconseguir compilar i executar la versió 1.3 de `jNetPcap` a Android. En aquest projecte hem seguit les seves instruccions [30] per poder analitzar els paquets capturats mitjançant aquesta llibreria.

4.2.1. Compilar les llibreries

Per compilar les llibreries *libpcap* i *libjnetpcap* cal descarregar l'eina NDK que ofereix Android per treballar amb llibreries natives. Seguidament crear la

carpeta *jni* a l'arrel del nostre projecte i copiar-hi les llibreries C/C++ que adjunta Sergio a les seves instruccions. Finalment, s'ha de navegar fins a la carpeta *jni* del nostre projecte des d'un terminal i executar la instrucció *ndk-build*. Si anteriorment no em afegit el directori de la NDK a les variables d'entorn del sistema haurem d'especificar la ruta absoluta. Per exemple, a linux:

```
$> cd /home/pau/development-android/LocatedTraffic/jni
$> ./home/pau/android-ndk-r8d/ndk-build
```

La següent figura mostra la sortida en consola del procés de compilació.

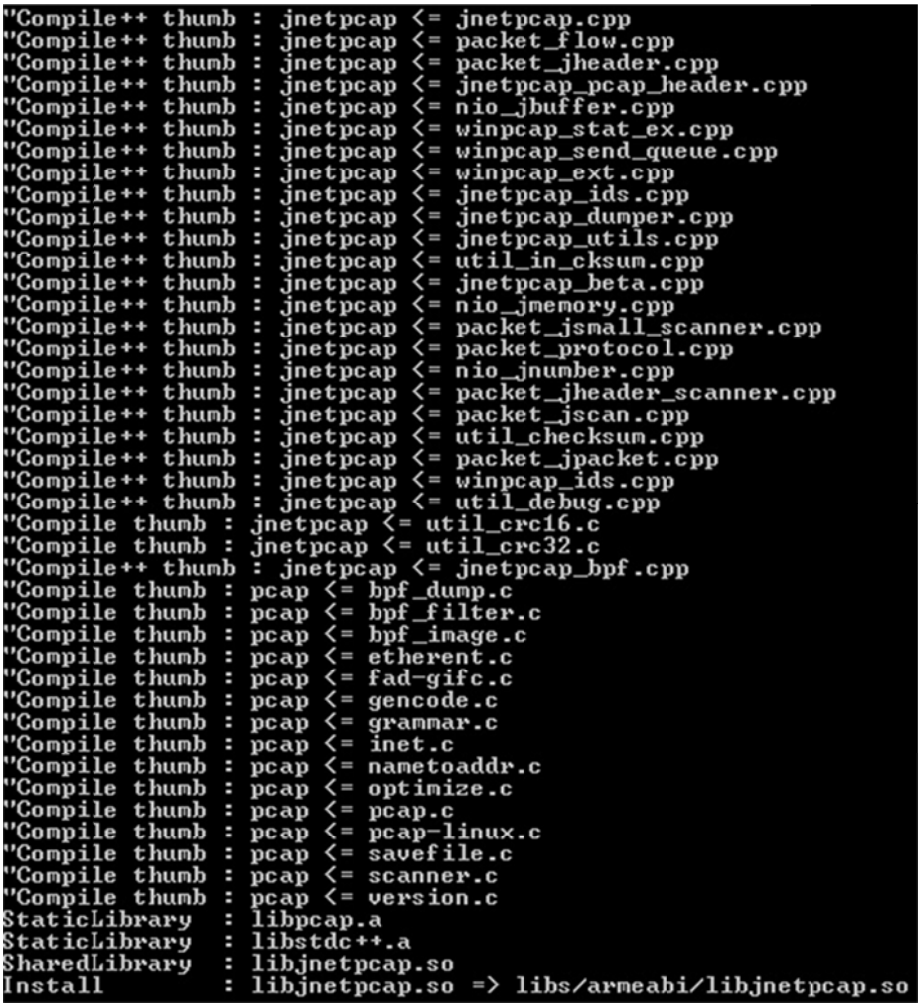

Fig 4.2 Sortida del procés de compilació amb la NDK

Si la compilació és satisfactòria, en el nostre projecte trobarem les carpetes *libs* i *obj*. El directori *obj/local/armeabi* conté els binaris generats i el directori *libs/armeabi* conté una copia d'aquests binaries després d'eliminar els símbols de depuració. La llibreria que finalment es empaquetada a la aplicació (*apk*) és la del directori *libs*.

4.2.2. Analitzar captures

Després de compilar les llibreries podem utilitzar les classes Java de `jNetPcap` per obrir fitxers prèviament generats per *tcpdump*. El següent codi mostra com obrir el fitxer *captura.pcap* utilitzant la classe `Pcap`.

```
private static int PACKETS_PER_LOOP = 5;
StringBuilder err = new StringBuilder();

final Pcap parser = Pcap.openOffline("/mnt/sdcard/captura.pcap", err);
```

Ara podem processar un conjunt o la totalitat dels paquets mitjançant la funció `loop()` i un controlador que implementi la interfície `JPacketHandler`. La constant per indicar que volem processar tots els paquets és `Pcap.LOOP_INFINITE`.

```
public interface JPacketHandler<T> {
     public void nextPacket(JPacket packet, T user);
}

parser.loop(PACKETS_PROCESSED_PER_LOOP, handler, null);
parser.close();
```

En el bucle `nextPacket()` podem accedir a cada paquet, carregat en un objecte `JPacket`, o per exemple guardar-los en un `ArrayList<JPacket>` per analitzar-los posteriorment. El següent exemple implementa la interfície `JPacketHandler` i determina el protocol de xarxa (IPv4 o IPv6) i el protocol de transport (TCP o UPD) de cada paquet.

```
JPacketHandler<String> handler = new JPacketHandler<String>() {
     @Override
   public void nextPacket(JPacket packet, String user) {
     if (packet.hasHeader(Ip4.ID))
       Log.i(TAG, “Protocol de xarxa: Ipv4”);
     if (packet.hasHeader(Ip6.ID))
       Log.i(TAG, “Protocol de xarxa: Ipv6”);
     if (packet.hasHeader(Tcp.ID))
       Log.i(TAG, “Protocol de transport: TCP”);
     if (packet.hasHeader(Udp.ID))
       Log.i(TAG, “Protocol de transport: UDP”);
     }
};
```

La figura 4.3 mostra el detall d'un paquet capturat amb l'aplicació LocatedTraffic. A la primera línia mostrem l'identificador del paquet juntament amb la diferència de temps, en mil·lisegons, entre la marca de temps de la localització (a la que s'associa el paquet) i la marca de temps del paquet. Seguidament presentem el valor del *timestamp* del paquet, també en mil·lisegons. La resta d'informació es divideix en tres nivells xarxa, transport i aplicació. Del primer nivell en detallem el protocol (IPv4, IPv6 o ICMP) i les adreces IP. Del segon nivell detallem el protocol (TCP o UPD) i els ports origen

i destí de la comunicació. Finalment, al tercer nivell presentem el protocol (HTTP, RTP, SDP o SIP) i el tipus de contingut.

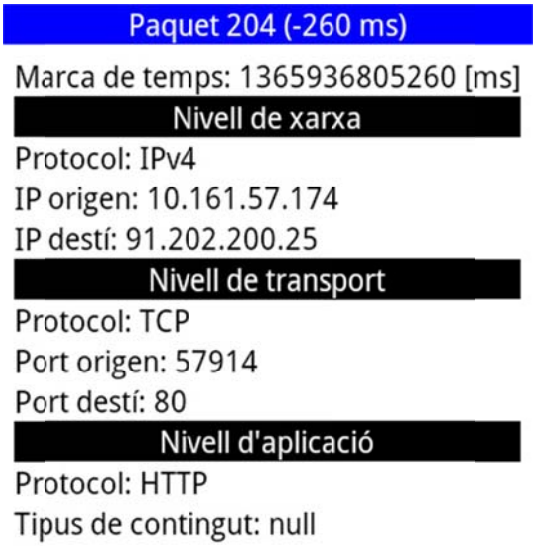

Fig 4.3 Dissecció d'un paquet de xarxa capturat i analitzat amb LocatedTraffic

CAPÍTOL 5. LOCALITZACIÓ DEL TRÀFIC DE XARXA

Als apartats anteriors hem detallat els mecanismes necessaris per capturar les dades d'ubicació i el tràfic de xarxa generat en dispositius mòbils Android. En aquest capítol veurem com relacionar aquests coneixements per acabar implementant una aplicació on recollim i vinculem aquestes dades.

La nostra aplicació LocatedTraffic ofereix, d'una banda, un servei de localització de paquets de xarxa i, d'altra banda, presenta una eina d'anàlisi de les dades capturades. Volem recalcar que aquest analitzador només està geolocalitzant les dades d'un sol terminal; la utilitat real, relacionada amb planificació de xarxes, passa per recollir les dades de molts usuaris, *off-line*, i analitzar-les conjuntament. Això queda obert per futurs projectes.

El capítol es divideix en dos seccions. A la primera detallarem el servei de localització de paquets: les pantalles i la informació que presenten, els paràmetres que podem ajustar, l'estructura de classes i el format de les dades que emmagatzema. A la segona secció detallarem l'analitzador de les dades capturades: l'algoritme implementat per agrupar el tràfic de xarxa a cada localització, les classes creades per carregar les dades, com representar-les en un mapa de Google Maps i la informació que es mostra quan pressionem un *item* del mapa.

5.1. El servei LocatedTraffic

La plataforma Android permet implementar, a través de serveis, tasques de llarga durada que s'executen en segon pla. Generalment, aquestes operacions no aporten informació a l'usuari fins que no han finalitzat i bloquejarien la interfície gràfica si s'executessin al mateix procés.

El servei inclòs a l'aplicació LocatedTraffic hereta de la classe `Service`, i s'encarrega d'escoltar les actualitzacions d'ubicació i d'iniciar la captura de paquets. En aquest apartat detallarem les preferències del servei, l'estructura de classes i el format, relació i granularitat de les dades capturades.

5.1.1. Paràmetres ajustables

Les preferències és divideixen en tres categories: generals (del servei), de tràfic de xarxa i de localització.

5.1.1.1. Preferències generals

Com veurem amb més detall a l'apartat 5.1.3, el servei genera dos fitxers cada vegada que s'executa. Amb la intenció de segmentar les dades i facilitar l'anàlisi hem afegit la opció de generar dos fitxers per dia, en cas que estigui

corrent contínuament. El format de nom proposat per als fitxers és *capture_YYYY-MM-DD* i les extensions són `.pcap` i `.loc`, per als fitxers de paquets i localitzacions, respectivament). Si desmarquem la opció, el servei crea els fitxers `capture.pcap` i `capture.loc`.

5.1.1.2. Preferències de tràfic de xarxa

Els paràmetres predeterminats d'execució de tcpdump són els següents:

```
-i any -p -vvv -ttt -w /sdcard/LocatedTraffic/capture_YYYY-MM-DD.pcap
```

L'aplicació permet ajustar la interfície on escoltarà el procés, el detall de la captura i activar el mode promiscu (captura de paquets per a tercers). Per llistar les interfícies disponibles utilitzem la classe `TcpdumpInterface` del projecte Shark [27] (veure figura 5.3). La següent taula detalla aquestes preferències i els seus possibles valors.

Taula 5.1 Preferències de captura de trànsit de xarxa

1. Preferència	2. Possibles valors	3. Observacions

Interfície	`lo`
	`any`
	`eth0`
	`rmnet0`
		El llistat d'interfícies depèn del dispositiu i de si el mòbil està connectat a una Wi-Fi o a una xarxa 3G.

Detall de la captura	Cap Detallat Molt detallat Màxim de detall	El nivell de detall de la captura (inclou informació addicional, com temps de vida, identificació o opcions dels paquets IP) es controla amb el paràmetre "v" (*verbose)*.
Mode promiscu	Activat Desactivat	Per activar el mode promiscu eliminem el paràmetre "p" de la instrucció que llença tcpdump.

5.1.1.3. Preferències de localització

LocatedTraffic, per defecte, no limita el nombre d'actualitzacions d'ubicació que vol rebre. Per altre banda, com hem vist al final del capítol 3, rebem una actualització aproximadament cada segon.

Amb la intenció de guardar només les localitzacions que són realment noves, l'aplicació permet (de forma opcional) activar un filtre que (comparant les marques de temps, la precisió i el proveïdor) determina si la nova actualització és millor que l'anterior millor. El fragment de codi utilitzat és el que ofereix la documentació a l'apartat *Location Strategies* [31].

5.1.2. Estructura i pantalles

En aquest apartat descriurem el conjunt de classes que formen el servei d'emmagatzematge de dades de l'aplicació LocatedTraffic. Primerament descriurem com estan relacionades les activitats i el servei, a continuació, explicarem com fer servir l'aplicació i la informació que presenta a la pantalla principal. Finalment, detallarem l'estructura del servei de localització de paquets.

L'aplicació LocatedTraffic, sense tenir en compte la part d'anàlisi de les dades, presenta dos activitats i un servei. L'activitat principal, `LocatedTraffic`, s'encarrega d'iniciar tant el servei de captura com l'activitat de preferències. La figura 5.1 mostra la relació entre les activitats i el servei.

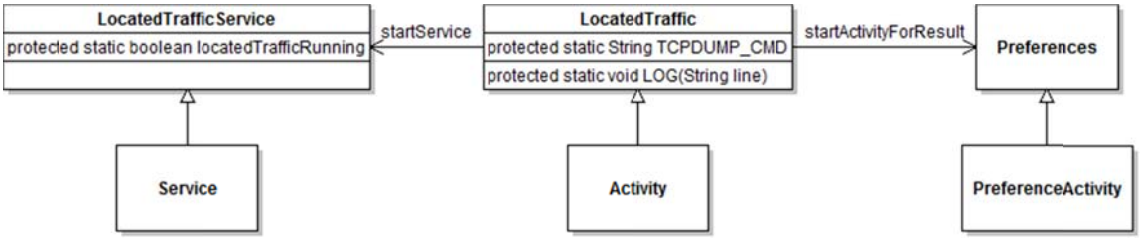

Fig 5.1 Relació d'activitats del servei de localització de paquets

A través del botó menú podem accedir a l'activitat `Preferences`, que presenta el conjunt de paràmetres ajustables detallats a l'apartat anterior. Quan aquesta finalitza, a l'activitat principal es recullen els canvis realitzats a les preferències i s'actualitza la informació de la pantalla.

El servei de localització de tràfic, `LocatedTrafficService`, es llença a través del botó "Iniciar servei" de la pantalla principal. Un cop iniciat, s'executa en segon pla, avisa a l'usuari que el servei està actiu a través de la barra de notificacions i segueix en funcionament quan l'aplicació ja no es visible per l'usuari.

La pantalla principal mostra, a la part superior, la comanda que llençarà el procés tcpdump quan iniciem el servei. A sota tenim dos botons, per iniciar el servei o l'analitzador de dades (que descriurem a l'apartat 5.2), i el tros de pantalla restant s'utilitza per mostrar els esdeveniments que es produeixen en el servei de localització. La figura 5.2 mostra la pantalla principal i la de preferències.

A l'iniciar el servei `LocatedTrafficService` s'instancien les classes `TCPdump` i `LocationGrabber`, encarregades de capturar i emmagatzemar les dades de tràfic de xarxa i localització, respectivament.

Fig 5.2 Pantalla principal i de preferències de l'aplicació LocatedTraffic

La classe `TCPdump` (del paquet *com.shark*) hereta de la classe `RootShell` per poder executar comandes a la consola de Linux com a superusuari. En iniciar el servei, la classe `TCPdump` executa la instrucció que llença el procés tcpdump.

```
/data/data/com.locatedtraffic/files/tcpdump -i eth0 -p
     -vvv -ttt -w /mnt/sdcard/LocatedTraffic/capture_2013-04-13.pcap&
```

Aquest procés escolta a la interfície indicada (en l'exemple eth0) i guarda els paquets que hi arriben o en surten en el fitxer *capture_2013-03-13.pcap*. El signe & afegit al final de la comanda provoca l'execució en segon pla per no bloquejar la consola.

La classe `LocationGrabber` implementa la interfície `LocationListener` (veure l'apartat 2.1.3) per poder rebre les actualitzacions d'ubicació i les notificacions del servei de localització.

Al rebre una nova localització sempre l'afegim al fitxer, excepte si l'usuari a triat avaluar les ubicacions en les preferències. En aquest cas, abans d'introduir-la determinem si és millor que l'anterior i si no ho és queda descartada automàticament. Les actualitzacions arriben de forma contínua i amb la màxima freqüència possible (totes les que rep el servei d'Android); d'aquesta manera obtenim amb exactitud la posició geogràfica del dispositiu a cada moment, però per contra consumim molta bateria. Quan rebem qualsevol de les altres notificacions que descriu la interfície `LocatedListener` actualitzem el quadre de log de la pantalla principal.

La figura 5.3 mostra el diagrama de classes que formen el servei LocatedTraffic.

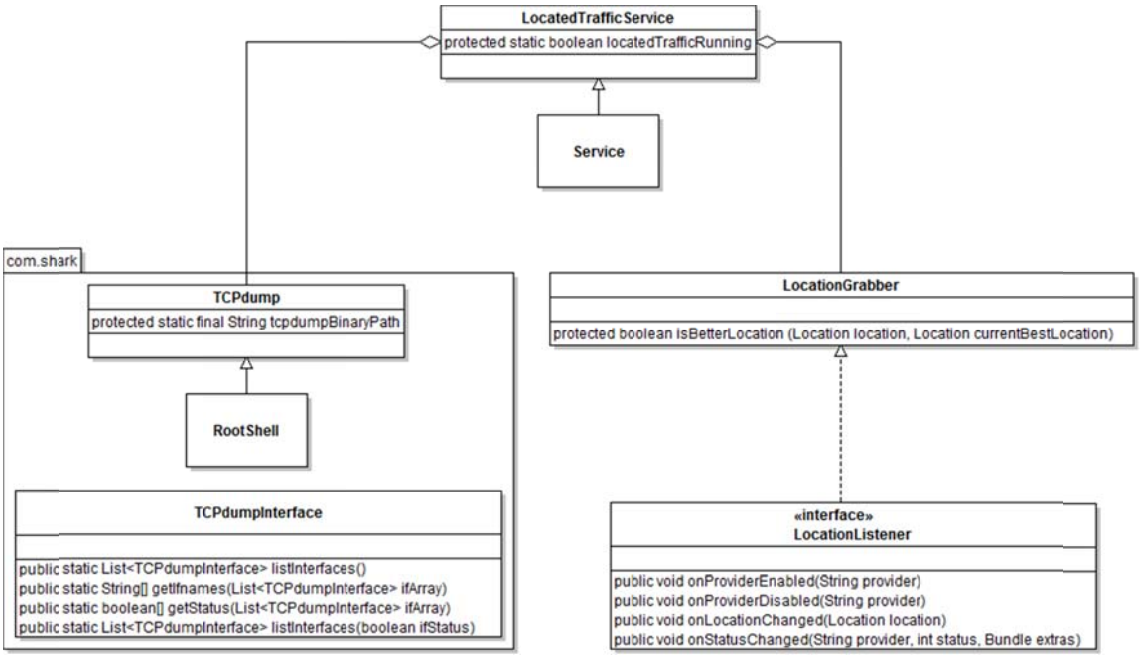

Fig 5.3 Diagrama de classes del servei LocatedTraffic

5.1.3. Descripció de les dades

En aquest apartat veurem les dades emmagatzemades per l'aplicació, en detallarem el format, la resolució i explicarem com relacionar-les.

5.1.3.1. Format libpcap

El format libpcap descriu com estructurar en un fitxer les dades de xarxa capturades. És l'estàndard "de facto" en captura de tràfic de tot tipus de xarxes en sistemes UNIX. L'extensió de fitxer proposada és *.pcap* [32].

Els fitxers .pcap comencen amb una capçalera global, seguida de la capçalera del primer paquet, com es mostra a la figura 5.4.

Capçalera global	Capçalera paquet 1	Dades paquet 1	...	Capçalera paquet N	Dades paquet N

Fig 5.4 Estructura dels fitxers libpcap.

La capçalera global està formada per set camps: número de 32 bytes per detectar el format de fitxer i l'ordre dels bytes, la versió mínima i la màxima, la correcció de temps en segons entre GMT (UTC) i la zona local dels *timestamps*

dels paquets capturats, la precisió dels *timestamps* (a la pràctica els *timestamps* són sempre GMT, per tant aquest camp val zero), la longitud (en bytes) dels paquets capturats i el tipus de capçalera de nivell d'enllaç de dades. A continuació copiem la definició de la capçalera global.

```
typedef struct pcap_hdr_s {
        guint32 magic_number;   /* magic number */
        guint16 version_major;  /* major version number */
        guint16 version_minor;  /* minor version number */
        gint32  thiszone;       /* GMT to local correction */
        guint32 sigfigs;        /* accuracy of timestamps */
        guint32 snaplen;        /* max length of captured packets */
        guint32 network;        /* data link type */
} pcap_hdr_t;
```

Per una altra banda, la capçalera de cada paquet consta de quatre camps: el *timestamp* en segons, el *timestamp* en microsegons, el número de bytes del paquets guardats al fitxer i el número de bytes del paquet original.

```
typedef struct pcaprec_hdr_s {
        guint32 ts_sec;       /* timestamp seconds */
        guint32 ts_usec;      /* timestamp microseconds */
        guint32 incl_len;     /* bytes of packet saved in file */
        guint32 orig_len;     /* actual length of packet */
} pcaprec_hdr_t;
```

5.1.3.2. Format dels fitxers de localitzacions

El servei de localització de paquets registra els *listeners* necessaris per començar a rebre actualitzacions d'ubicació (encapsulades en objectes `Location`). La classe `LocationGrabber` s'encarrega d'emmagatzemar cada actualització rebuda en una nova línia. Per tant, el fitxer amb extensió *.loc* tindrà tantes línies com actualitzacions s'han rebut durant el temps d'execució del servei (en el cas que no en descartem cap).

Per registrar els objectes `Location` en un fitxer utilitzem els vuit camps que el defineixen, i els guardem separats per punt i coma en el següent ordre: el timestamp en mil·lisegons, el nom del proveïdor, la precisió en metres, la latitud en graus, la longitud en graus, l'altitud en metres, la velocitat en metres per segon i el rumb en graus. Hem definit aquests paràmetres a la taula 2.1.

El següent és un exemple d'una ubicació emmagatzemada al fitxer.

```
1365936709000;gps;8.0;41.48666024208069;2.3190218210220337;
     123.0;27.5;78.39844
```

Resumint, els fitxers amb extensió *.loc* ens ofereixen un llistat de localitzacions, ordenat cronològicament, que podem carregar en un vector d'objectes `Location`.

5.1.3.3. Relació entre les dades

El que volem és determinar a quina ubicació s'ha generat un paquet determinat. Per fer-ho hem d'utilitzar les marques de temps, que són el paràmetre en comú entre les dades de localització i el tràfic de xarxa.

Com hem vist als apartats anteriors, la precisió dels *timestamps* és diferent: els paquets de xarxa presenten marques de temps en microsegons, mentre que les localitzacions són en mil·lisegons.

Teòricament la distància més propera, en temps, entre dos localitzacions és d'un mil·lisegon. Per tant, la decisió de si un paquet s'ha generat en una ubicació presenta un error absolut de ±0,5 ms.

Un cop processats, els *timestamps* presenten granularitat d'un mil·lisegon, ja que queda fixat per les dades menys precises.

En el cas de les ubicacions, la latitud i longitud presenten una gran quantitat de decimals de grau (des de 6 fins a 14 decimals); això significa precisió des de 0,11 m fins a 110 µm (els decimals, a partir de la posició 9, no aporten més grau de precisió) [33]. No obstant, el nivell d'exactitud d'una localització es defineix com una probabilitat del 68% dins d'una superfície (*radius of 68% confidence*). Es a dir, si dibuixem una circumferència centrada a la latitud i longitud amb radi igual al valor *accuracy*, existeix un 68% de probabilitat que la ubicació real caigui dins d'aquesta àrea.

5.2. Analitzador

L'aplicació LocatedTraffic inclou un analitzador que, a partir d'un parell de fitxers de dades (localització i traça de paquets), agrupa a cada localització els paquets més propers i ho representa en un mapa de Google Maps. Les ubicacions on s'ha capturat tràfic de xarxa es marquen en verd i les que no en vermell. Al seleccionar una de les marques del mapa es mostra la informació capturada en aquest punt.

En aquest apartat explicarem l'algoritme utilitzat per fer l'agrupació de paquets, les classes creades per vincular aquestes dades i representar-les a Google Maps i, finalment, detallarem la informació que proporciona l'analitzador.

5.2.1. Algoritme d'agregació

Inicialment, l'usuari ha de triar quin parell de fitxers vol analitzar, identificats pel dia en que van ser generats. Seguidament, l'aplicació carrega les dades dels fitxers, generant dos llistes ordenades cronològicament i relacionades per marques de temps. La llista de localitzacions és un vector d'objectes `Location`, i la llista de paquets és un vector d'objectes `JPacket`. Les classes `LocatedPacket`

i `Marker`, a les que fem referència durant aquest apartat, es detallen en el punt 5.2.2.

L'algoritme recorre el vector de paquets dins d'un bucle de localitzacions. A cada iteració es compara el *timestamp* de la ubicació actual (t_a) i el de la següent (t_{a+1}) amb la marca de temps del paquet corresponent (t_p); mentre el valor absolut de la diferència de temps ($t_a - t_p$) és menor que ($t_{a+1} - t_p$) guardem aquests paquets en un vector `JPacket` temporal.

En el moment que la condició no es compleix, creem un objecte `LocatedPacket` (amb la localització actual (t_a) i el vector de paquets temporal) i l'afegim al `Marker` de paquets amb tràfic, ens guardem l'última posició dins del bucle de paquets i reiniciem el vector de paquets temporal. Si no es compleix la condició i el vector temporal té longitud zero, vol dir que aquesta localització no té paquets vinculats; llavors afegiríem l'objecte `LocatedPacket` al `Marker` de paquets sense tràfic. Quan arribem a l'últim element del vector de localitzacions forcem $t_a = t_{a+1}$ i afegim en el vector temporal tots els paquets restants.

Exemple numèric de l'algoritme

Suposem que volem agrupar les dades d'un escenari amb 4 localitzacions i 5 paquets de xarxa. Sabem que les localitzacions s'han capturat en els instants (A=2000 ms, B=3500 ms, C=4200 ms i D=6000 ms) i que els paquets s'han generat en els instants (P_1=1300 ms, P_2=2745 ms, P_3=3900 ms, P_4=5100 ms, P_5=6800 ms).

En la primera iteració recuperem l'instant en que s'ha capturat la primera localització i la següent (t_a = A i t_{a+1} = B) i comparem el valor absolut de la diferència de temps entre aquests instants i el del primer paquet ($t_p = P_1$).

$$| t_a - t_p | < | t_{a+1} - t_p | \qquad |2000 - 1300| < |3500 - 1300| \qquad \mathbf{700 < 2200}$$

Com la desigualtat és certa, entrem a la primera condició, afegim el paquet P_1 al vector temporal i fem la mateixa operació amb el següent paquet ($t_p = P_2$).

$$| t_a - t_p | < | t_{a+1} - t_p | \qquad |2000 - 2745| < |3500 - 2745| \qquad \mathbf{745 < 755}$$

La desigualtat torna a ser certa. Afegim P_2 al vector temporal i continuem amb el següent paquet ($t_p = P_3$).

$$| t_a - t_p | < | t_{a+1} - t_p | \qquad |2000 - 3900| < |3500 - 3900| \qquad \mathbf{\cancel{1900 < 400}}$$

La desigualtat no es compleix i $t_a \neq t_{a+1}$. Llavors, creem un objecte `LocatedPacket` (que associa la localització A amb els paquets P_1 i P_2) i l'afegim al `Marker` de localitzacions amb tràfic. Seguidament ens guardem la posició dins del bucle de paquets i reiniciem el vector temporal.

En la segona iteració, la ubicació actual és t_a = B, la següent és t_{a+1} = C i comencem la comparació amb $t_p = P_3$.

$$| t_a - t_p | < | t_{a+1} - t_p | \qquad |3500 - 3900| < |4200 - 3900| \qquad \sout{400 < 300}$$

La desigualtat no es compleix i $t_a \neq t_{a+1}$. Ara, com que el vector temporal té longitud zero, igualment creem un objecte `LocatedPacket` (amb la localització B) però l'afegim al `Marker` de localitzacions sense tràfic.

En la tercera iteració comparem t_a = C i t_{a+1} = D amb el paquet $t_p = P_3$.

$$| t_a - t_p | < | t_{a+1} - t_p | \qquad |4200 - 3900| < |6000 - 3900| \qquad \mathbf{300 < 2100}$$

Com es compleix la condició, afegim P_3 al vector temporal i continuem amb el següent paquet ($t_p = P_4$).

$$| t_a - t_p | < | t_{a+1} - t_p | \qquad |4200 - 5100| < |6000 - 5100| \qquad \sout{900 < 900}$$

La condició no es compleix i $t_a \neq t_{a+1}$. Llavors, creem un objecte `LocatedPacket` (que associa C amb P_3) i l'afegim al `Marker` de localitzacions amb tràfic, ens guardem la posició i reiniciem el vector temporal. Tot i que la diferència de temps en aquest cas és la mateixa, el criteri de desempat de l'algoritme determina que el paquet P_4 no quedi vinculat a la localització C.

En la quarta iteració t_a = D i com aquesta és la última localització de la llista forcem t_{a+1} = D. Comencem a comparar amb $t_p = P_4$.

$$| t_a - t_p | < | t_{a+1} - t_p | < |6000 - 5100| < |6000 - 5100| = \sout{900 < 900}$$

La condició no es compleix, però $t_a = t_{a+1}$, llavors afegim P_4 al vector temporal i continuem amb el següent paquet ($t_p = P_5$).

$$| t_a - t_p | < | t_{a+1} - t_p | < |6000 - 6800| < |6000 - 6800| = \sout{800 < 800}$$

La condició no es compleix, però $t_a = t_{a+1}$, llavors afegim P_5 al vector temporal. Com aquest és l'últim paquet de la llista, creem un objecte `LocatedPacket` (que associa D amb P4 i P5) i l'afegim al `Marker` de localitzacions amb tràfic.

Com ja no queden més localitzacions ni paquets a comparar l'algoritme finalitza i l'agregació queda de la següent manera:

- Els paquets P_1 i P_2 queden associats a la localització A.
- La localització B no té paquets vinculats.
- El paquet P_3 queda associat amb la localització C.
- Finalment, els paquets P_4 i P_5 estan vinculats amb la localització D.

Donada la limitació d'espai a la memòria i per facilitar la lectura d'aquesta, podem trobar la implementació Java de l'algoritme a l'annex 7.

5.2.2. Representació localitzada dels paquets capturats

Un cop tenim clar com relacionar els paquets de xarxa amb les localitzacions, ens proposem representar aquesta informació en un mapa. Per fer-ho creem l'activitat `Analizer`, heretant de la classe `MapActivity`, per poder treballar amb objectes `MapView`.

Els elements `MapView` requereixen ser validats amb una clau - *Maps API key* - que ens proporciona Google a canvi de registrar l'empremta digital MD5 del certificat que farem servir per signar les nostre aplicacions. Google obliga a inscriure les nostres aplicacions al servei de mapes. A l'apartat 2.2.2 hem detallat el procés per crear aplicacions basades en el servei Google Maps; no obstant, a partir del 18 de març de 2013 ja no és possible obtenir les claus seguint aquest procés, a causa d'una actualització del servei [34]. Els passos a seguir per accedir a la nova versió els podem trobar a "*Google Maps Android API v2*" [35].

La figura 5.5 mostra el diagrama d'activitats de l'analitzador, indicant els elements més importants de l'activitat `Analizer`.

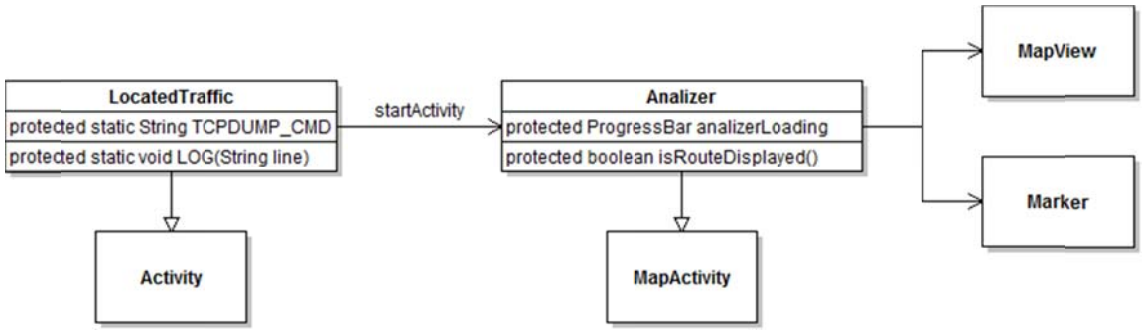

Fig 5.5 Diagrama d'activitats de l'analitzador

La classe `MapView` treballa amb vectors d'objectes `OverlayItem` denominats `ItemizedOverlay`, que utilitza per posicionar ítems sobre el mapa. La solució proposada, per carregar i representar la informació capturada pel servei `LocatedTrafficService`, és heretar dels ítem i vectors bàsics que defineix la API; concretament hem creat dos classes: `LocatedPacket` (que hereta de `OverlayItem` [36]) i `Marker` (que hereta de `ItemizedOverlay` [37]).

La classe `LocatedPacket` hereta un punt geogràfic (`GeoPoint`) amb un títol i una descripció; nosaltres afegim més detall de localització (*timestamp*, proveïdor, precisió, altitud, velocitat i rumb) i, a més, annexem un llistat de paquets que s'han capturat en aquest punt (vector d'objectes `JPacket`).

La classe `Marker` consta d'una llista d'objectes `LocatedPacket` i ofereix les funcions necessàries per dibuixar cada punt al mapa i per gestionar l'acció a realitzar quan l'usuari prem un dels ítems. En l'activitat `Analizer`, quan l'usuari toca sobre d'un ítem mostrem la informació completa d'aquesta localització (les

dades emmagatzemades a l'objecte `LocatedPacket`). A més, si aquesta localització té paquets de xarxa associats donem l'opció a visualitzar la informació detallada de cada paquet. La figura 5.6 mostra l'estructura de classes utilitzada per carregar les dades del servei `LocatedTrafficService` i representar-les en el mapa.

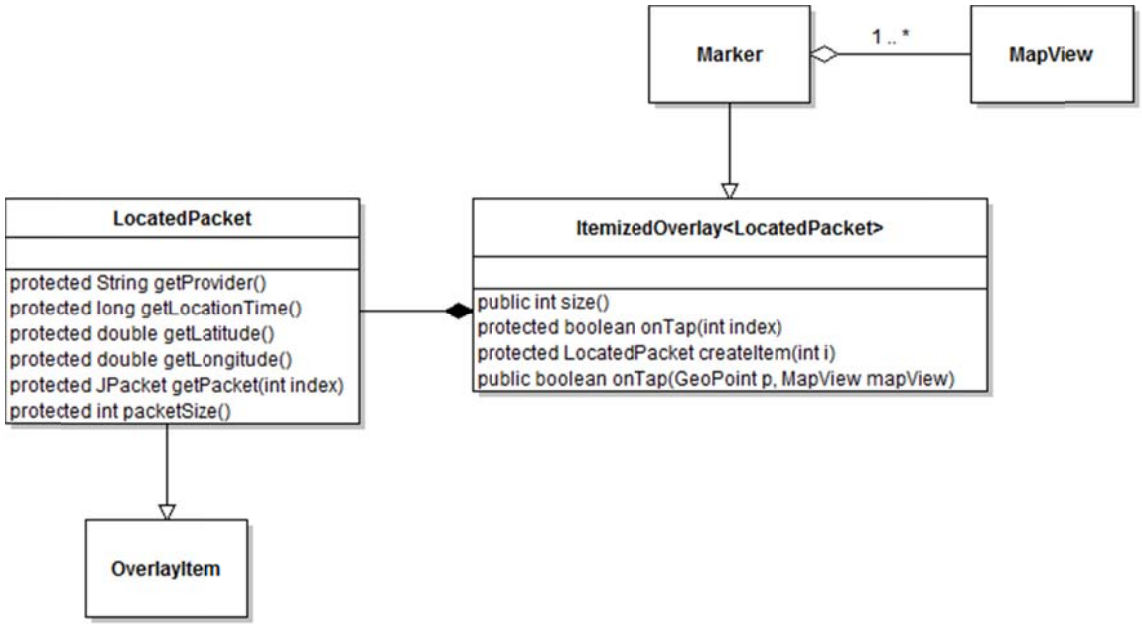

Fig 5.6 Diagrama de classes per carregar les dades a un element `MapView`

5.2.3. Informació reportada

Inicialment, l'aplicació mostra un desplegable amb les dades diàries disponibles per analitzar, cadascuna formada per un parell de fitxers (dades de localització i traça de paquets). Al seleccionar una de les opcions, carreguem les dades en memòria, apliquem l'algoritme d'agregació de paquets i, finalment, mostrem a l'usuari un mapa de *Google Maps* marcat amb banderes. Hem utilitzat banderes de color verd per marcar ubicacions que tenen paquets de xarxa associats, i banderes de color vermell per ressaltar les ubicacions on no em rebut tràfic.

A sobre del mapa, en la part inferior de la pantalla, hem incorporat dos botons. El botó de l'esquerra serveix per triar si volem veure totes les localitzacions o només les que tenen paquets de tràfic associat. El botó de la dreta mostra un missatge amb el resum de les dades analitzades: número total de localitzacions, número de localitzacions amb tràfic vinculat, número total de paquets de xarxa i nombre total de bytes (enviats i rebuts).

La figura 5.7 presenta dos captures l'aplicació LocatedTraffic on es mostren les localitzacions capturades durant un trajecte en cotxe per l'autopista C-32 en direcció a Premià de Dalt.

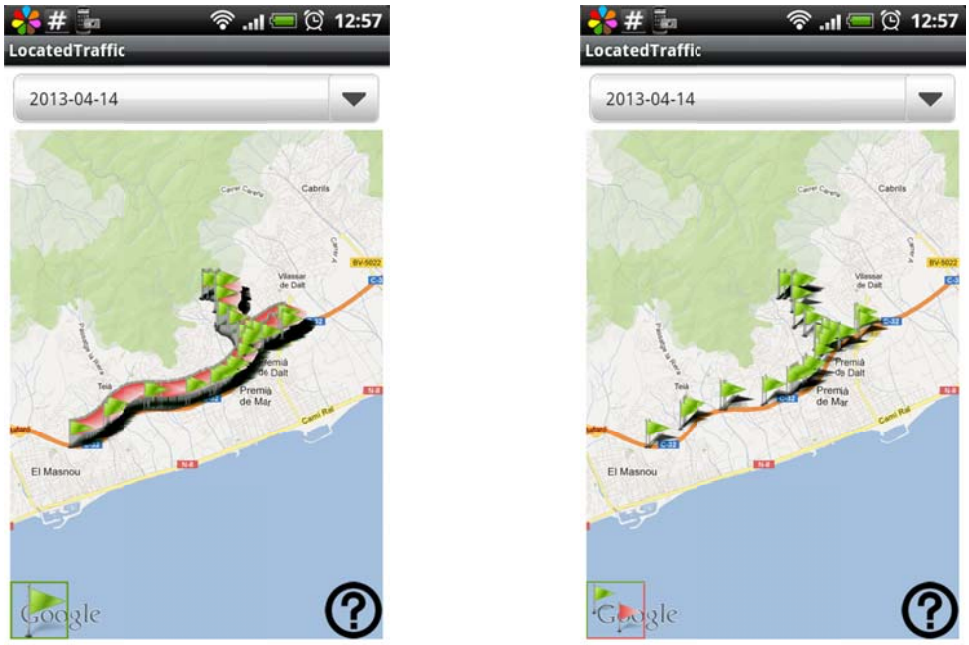

Fig 5.7 Representació de les dades capturades pel servi LocatedTraffic

Al prémer un dels ítem del mapa, l'aplicació mostra la informació d'aquesta localització: *timestamp*, proveïdor, latitud, longitud, altitud, velocitat i rumb. Si la icona tocada és de color verd, a més, tenim l'opció de carregar les dades disponibles dels paquets vinculats a la ubicació; detallant els nivells de xarxa, transport i aplicació de cadascun d'ells.

La figura 5.8 mostra dos captures de pantalla de l'aplicació LocatedTraffic on es mostren les dades d'una localització (a l'esquerra) i la informació dels paquets de xarxa vinculats a aquesta (a la dreta).

Fig 5.8 Informació de localització i paquets associats al prémer un ítem

5.3. Proves i validacions

Es va realitzar una prova, de 9 minuts, el dia 18 d'abril de 2013, durant un trajecte a peu prop del carrers Clot, Aragó i Ter de Barcelona (des de Plaça de les Glòries cap al carrer Mallorca). En aquest període vam rebre 545 localitzacions i 492 paquets de xarxa (amb un total de 532320 bytes).

L'analitzador implementat a l'aplicació LocatedTraffic agrupa els 492 paquets en 80 ubicacions diferents; això suposa només el 14,68% del total de localitzacions capturades pel servei.

La taula 5.2 detalla com queden agrupats els paquets a les deu primeres i deu darreres localitzacions que tenen tràfic de xarxa associat. Les dues últimes columnes mostren la diferència de temps entre el *timestamp* de la localització i el *timestamp* del primer i últim paquet associat a ella.

Taula 5.2 Detall de l'agrupació de paquets en la prova del 18 d'abril de 2013

Localització	Proveïdor	Paquets associats	δt amb el primer paquet	δt amb el darrer paquet
1	Network	31 (1 – 31)	+3391 ms	-1060 ms
2	Network	11 (32 – 42)	+160 ms	-11 ms
3	GPS	5 (43 – 47)	+28 ms	-82 ms
4	GPS	4 (48 – 51)	-272 ms	- 304 ms
5	GPS	5 (52 – 56)	+398 ms	+296 ms
6	GPS	1 (57 – 57)	+348 ms	---
7	GPS	1 (58 – 58)	-254 ms	---
8	GPS	1 (59 – 59)	-39 ms	---
9	GPS	12 (60 – 71)	+365 ms	-464 ms
10	GPS	6 (72 – 77)	+485 ms	+215 ms
...				
71	GPS	1 (427 – 427)	+365 ms	---
72	GPS	18 (428 – 445)	+308 ms	-328 ms
73	GPS	7 (446 – 452)	-75 ms	-187 ms
74	GPS	5 (453 – 457)	-241 ms	-477 ms
75	GPS	16 (458 – 473)	+428 ms	-474 ms
76	GPS	1 (474)	+433 ms	---
77	GPS	9 (475 – 483)	+320 ms	-477 ms
78	GPS	7 (484 – 490)	+483 ms	+143 ms
79	GPS	1 (491 – 491)	-155 ms	---
80	GPS	1 (492 – 492)	+485 ms	---

Com podem observar, quan el senyal GPS es sincronitza, el grup de paquets associats a cada localització es fa més petit, i la diferència de temps màxima (entre *timestamp* de localització i de paquet) no sobrepassa els 500 mil·lisegons.

Es va realitzar una segona prova de consum de bateria, en un mòbil HTC Desire amb dos anys d'antiguitat. Normalment la bateria del dispositiu té una

durada de 18 hores (sense realitzar trucades llargues ni jugar). El servei `LocatedTrafficService` a ple rendiment fa que la bateria es descarregui en 13 hores (sense carregar el telèfon amb altres processos importants).

El fet de mantenir encès el GPS durant tot el temps que el servei està en execució provoca un gran consum d'energia. Si la forma de treballar amb el GPS fos intermitent (només encès quan es genera trànsit de xarxa) podríem reduir el consum.

Basant-nos en el percentatge de localitzacions útils per geolocalitzar el tràfic (valor calculat a la prova anterior), podem fer una estimació de l'estalvi de consum. Assumim que la taxa de localitzacions és 1 loc/seg (es a dir, en mitja rebem una localització cada segon) i que només el 15% d'aquestes localitzacions són útils per geolocalitzar els paquets capturats. El primer raonament lògic és pensar que, amb aquestes característiques, podrem mantenir apagat el GPS el 85% del temps d'execució del servei.

El següent raonament és assumir que la creació de parelles de dades (localització més grup de paquets vinculats) segueix una distribució de Poisson amb taxa mitja λ=0,15 unitats/seg (el 15% de la taxa de localitzacions). Això és equivalent a dir que la distribució de temps entre dues creacions consecutives és una exponencial negativa de mitja 1/λ = 6,67 s entre arribades. En aquest cas, la probabilitat de que es produeixin *n* arribades en un temps *t* és:

$$Pn(T) = \frac{(\lambda T)^n}{n!}\, e^{-\lambda T}$$

Amb aquesta distribució de temps trobem que l'interval de temps mínim per assegurar que cobrim el temps mig entre dos arribades i, a la vegada, aconseguim la màxima precisió són 7 segons. Llavors podem calcular la probabilitat que es produeixin una o més arribades d'actualitzacions en aquest interval,

$$P_{n>0}(7) \;=\; 1 - P_0(7) = 1 - \frac{(0{,}15 \cdot 7)^0}{0!} e^{-0{,}15 \cdot 7} = 0{,}650 \rightarrow\; 65\%$$

Amb un interval de 7 segons, reduïm la taxa d'actualitzacions en un factor 7 (comparant amb la freqüència d'1 localització/segon). Això provoca un estalvi d'energia que caldria quantificar empíricament. Per contra, perdem precisió d'ubicacions, fet que pot desvirtuar les dades si fem córrer l'aplicació en un dispositiu que porta una velocitat (v) en metres per segon tal que:

$$|v \cdot 7segons| > \; Accuracy$$

ja que, amb aquesta velocitat, al rebre una nova actualització d'ubicació hem sortit, segur, del cercle de confiança del 68% marcat pel valor de precisió (*accuracy*). A l'annex 8 podem trobar un petit anàlisi, amb dades reals, per intentar corroborar la hipòtesis de partida (l'interval de temps entre arribades segueix una distribució de Poisson).

CONCLUSIONS I LÍNIES FUTURES

Conclusions

Durant el desenvolupament del projecte hem pogut comprovar la ràpida evolució de la API d'Android. Aquest fet, d'una banda, afavoreix als desenvolupadors: correcció dels errors trobats i addició de mètodes, classes o eines que mancaven en versions anteriors; en resum, disposar d'una plataforma més potent. D'altra banda, és necessari decidir a quin nivell mínim de la API es vol fixar l'aplicació, ja que això obliga a estar contínuament revisant les actualitzacions per valorar de quines podem prescindir.

S'ha desenvolupat l'aplicació CustomCam, capaç de capturar imatges i vídeo en diferents formats i de fer *streaming* de vídeo en viu. En la implementació s'hi podria haver inclòs més codificadors utilitzant llibreries natives, sobretot per codificació d'àudio; aquesta opció s'ha descartat degut al temps limitat de desenvolupament, la complexitat i el fet que les noves versions d'Android ja amplien el ventall de formats multimèdia. No obstant, tot hi estar limitats per la versió del sistema operatiu utilitzat en el desenvolupament, considerem que la varietat de codificadors i formats que suporta l'aplicació juntament amb la capacitat de fer *streaming* en temps real cobreixen els objectius marcats per la part multimèdia.

D'altra banda, també hem estudiat les capacitats de localització i de monitorització de trànsit de xarxa, creant una aplicació que permet associar la localització i els paquets capturats. El servei de localització que ofereix Android ha complert les nostres expectatives. Hem pogut calcular que, en entorns oberts, la freqüència amb la que el receptor GPS s'actualitza (un cop connectat) és d'un segon; aquesta cadència és suficient per geolocalitzar amb detall el tràfic de xarxa generat al dispositiu.

Una de les complicacions que hem trobat durant el desenvolupament de l'aplicació de localització de trànsit és aconseguir privilegis de superusuari. En general, tots els fabricants d'*smartphones* impedeixen l'accés root als seus dispositius; HTC, en concret, bloqueja la memòria NAND del telèfon i el procés per aconseguir privilegis passa per modificar el *bootloader* (programa que s'inicia al engegar el dispositiu).

Hem pogut observar que, un cop sincronitzat el senyal GPS, la diferència de temps entre el *timestamp* d'una localització i el dels paquets associats a ella oscil·la entre 20 i 500 ms. Aquesta precisió és suficient per geolocalitzar amb precisió el trànsit de xarxa.

A través de l'aplicació Android Assistant, que podem trobar a Google Play, hem mesurant el consum de CPU de les aplicacions CustomCam i LocatedTraffic en un telèfon HTC Desire amb Android 2.2. El percentatge de CPU utilitzat per l'aplicació CustomCam varia entre el 4 i el 12% funcionant com a servidor d'*streaming*. Per altra banda, el consum de LocatedTraffic mentre funciona com

a servei (captura de localitzacions i paquets de xarxa) és molt més baix, varia entre 1,5 i 2,5% (amb pics de fins a 5%), en canvi, si fem la mesura mentre l'aplicació analitza les dades capturades, el consum arriba fins al 40%.

Durant la realització d'aquest projecte he après a desenvolupar aplicacions per a dispositius Android des de zero; els fonaments en programació Java assumits durant els estudis de telemàtica juntament amb l'experiència en programació web adquirida en l'activitat laboral han ajudat a agilitzar l'aprenentatge. Cal destacar, per una banda, la riquesa d'exemples i l'abundant documentació de la pàgina web de desenvolupadors [9] i, per altra banda, la seguretat basada en permisos que obliga a les aplicacions a declarar quins components farà servir per evitar males pràctiques. He trobat dificultats a l'hora d'aprofundir en els formats i codificadors multimèdia; l'API descriu amb detall com capturar imatges JPEG, però no ofereix les eines per fer la codificació en altres formats.

Aspectes mediambientals

El fet de mantenir el GPS encès de manera indefinida durant l'anàlisi del tràfic provoca un gran consum d'energia, que es tradueix en una acceleració brutal del temps de descàrrega de la bateria. Aquest problema és podria minimitzar analitzant les dades que captura el procés tcpdump a cada moment, i utilitzar-les com a interruptor del GPS. D'aquesta manera, si es dona un llarg període de temps durant el qual no hi ha activitat de xarxa, podrem apagar el GPS sense desvirtuar les dades capturades.

Suposant que l'aplicació LocatedTraffic es distribuís a una mostra gran d'usuaris, podríem analitzar totes les dades de forma conjunta. Això permetria localitzar millor la generació del tràfic, el que portaria a desplegaments més eficients d'estacions base cel·lulars i Wi-Fi, minimitzant tant consum energètic com radiacions.

Línies futures

La capacitat de memòria dels dispositius mòbils es reduïda (típicament, memòries microSD, de 4 a 32 GB). Els fitxers de localitzacions i tràfic generats el dia 18 d'abril, durant 9 minuts d'execució, ocupen 42 kB i 46 kB respectivament. Si ho extrapolem a 24 hores trobem que els fitxers pesarien 6,72 MB i 7,36 MB; arrodonint, l'aplicació genera 15 MB de dades diàriament. Per aquest motiu, caldria plantejar un sistema de copia de fitxers a un servidor, aprofitant una aturada del servei de localització de paquets.

En l'analitzador, no reportem el total de bytes rebuts i el total de enviats durant una sessió (ho fem en global, la suma). Per poder detallar aquesta informació hauríem de guardar, cada cop que s'inicia el servei, l'adreça IP origen. Això ho podríem fer en un tercer fitxer, comú per totes les sessions, on cada línia guardaria l'identificador de les dades (la data) i l'adreça IP origen. No obstant,

durant l'execució del servei, és probable que el dispositiu es connecti a diferents xarxes (Wi-Fi i 3G) i que, per tant, generi tràfic amb diverses adreces IP. Llavors, el fitxer d'adreces ha de emmagatzemar, a cada línia, l'identificador o data de la sessió i un llistat d'adreces IP (ja que caldrà sumar el tràfic entrant i sortint a cadascuna d'elles).

L'aplicació LocatedTraffic només geolocalitza el tràfic generat en un terminal, queda pendent guardar aquesta informació en una base de dades (externa al dispositiu). Aquesta base de dades emmagatzemaria la informació d'ubicació i tràfic generat per molts terminals diferents. El següent pas, seria processar les dades conjuntament, per intentar determinar la distribució i volum del tràfic en funció de la zona geogràfica, el moment del dia o l'època de l'any. Les observacions i conclusions extretes d'aquest anàlisi ajudaran a optimitzar el disseny de xarxes mòbils i Wi-Fi, i permetran una implantació més eficient a la regió geogràfica analitzada.

Per altra banda queda pendent realitzar més proves i mesures amb el servei LocatedTraffic per corroborar quina estadística segueixen les dades (amb l'anàlisi fet, veure annex 8, sembla raonable dir que segueixen una distribució de Poisson). Finalment caldria aplicar les mesures adients, segons la estadística, per optimitzar el consum energètic i aconseguir un servei més eficient.

Bibliografia

Pàgines web

[1] Wikipedia; iOS6; http://en.wikipedia.org/wiki/IOS_6

[2] Wikipedia; Nokia Asha Series; http://en.wikipedia.org/wiki/Nokia_Asha_series

[3] Bussiness Week; Google Buys Android for Its Mobile Arsenal; http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal

[4] Google Inc. & Open Handset Alliance; Android overview; http://www.openhandsetalliance.com/android_overview.html

[5] Google Inc.; Android Open Source Project; http://source.android.com/about/index.html

[6] Wireshark; Libpcap file format; http://wiki.wireshark.org/Development/LibpcapFileFormat#Libpcap_File_Format

[7] Visually; A brief history of Android; http://visual.ly/brief-history-android

[8] Wikipedia; Historial de versiones de Android; http://es.wikipedia.org/wiki/Anexo:Historial_de_versiones_de_Android

[9] Google Inc.; Android Architecture; http://developer.android.com/about/versions/index.html

[10] Google Inc.; Android permission; http://developer.android.com/reference/android/Manifest.permission.html

[11] Wikipedia; Global Android version distribution from November 2009 to February 2013; http://en.wikipedia.org/wiki/Android_version_history

[12] Google Inc.; Google Maps Privacy: The Street View & Wifi Scorecard – Search Engine Land; http://searchengineland.com/google-street-view-scorecard-55487

[13] MG SIEGLER; AOL; Latitude's New Dashboard View Is Exactly What Passive Location Needs; http://techcrunch.com/2010/05/26/google-latitude-location-history-dashboard/

[14] Wikipedia; 68-95-99.7 rule; http://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule

[15] Google Inc.; Geocoder class; http://developer.android.com/reference/android/location/Geocoder.html

[16] Oasis; Extensible Address Language (xAL);
http://www.immagic.com/eLibrary/TECH/OASIS/XAL_V2.PDF

[17] Google Inc.; Android 2.2 Platform Highlights;
http://developer.android.com/about/versions/android-2.2-highlights.html

[18] Google Inc.; Android Supported Media Formats;
http://developer.android.com/guide/appendix/media-formats.html

[19] Kirk Baucom; Android SeekBar preference; August, 2011:
http://robobunny.com/wp/2011/08/13/android-seekbar-preference/

[20] Simón Bisbal, Jordi Tabaré; Implementació de càmera Wi-Fi mitjançant dispositiu mòbil; Projecte Fi de Carrera; Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona (ETSETB); Setembre, 2012;
http://upcommons.upc.edu/pfc/handle/2099.1/16146

[21] RFC 3267: Real-Time Transport Protocol (RTP) Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs; J. Sjoberg, M. Westerlund, A. Lakaniemi, Q. Xie; June 2002

[22] RFC 4629: RTP Payload Format for ITU-T Rec. H.263 Video; J. Ott, C. Bormann, G. Sullivan, S. Wenger, R. Even; January 2007

[23] RFC 3984: RTP Payload Format for H.264 Video; S. Wenger, T. Stockhammer, M. Westerlund, D. Singer; February 2005

[24] Network Security Toolkit; 2003 – 20013;
http://networksecuritytoolkit.org/nst/index.html

[25] AlphaRev; AlphaRev 1.8 HBOOT reflash utility; http://alpharev.nl

[26] Stephen Erickson, Chris Ravenscroft, Dominik Schuermann, Adam Shanks; RootTools – Set of tools for rooted developers; 2012; https://code.google.com/p/roottools/

[27] Jiménez Feijóo, Sergio; Aplicació Shark;
http://prueba-android.svn.sourceforge.net/viewvc/prueba-android/trunk/

[28] Van Jacobson, Craig Leres and Steven McCanne, all of the Lawrence Berkeley National Laboratory, University of California, Berkeley, CA.; TCPDUMP; March, 2009; http://www.tcpdump.org/tcpdump_man.html

[29] Sly Technologies, Inc.; jNetPcap; 2005 – 2012; http://jnetpcap.com/

[30] Sly Technologies, Inc.; jNetPcap – Android OS; 2005 – 2012;
http://jnetpcap.com/userguide/android

[31] Google Inc.; Location Strategies;
http://developer.android.com/guide/topics/location/strategies.html

[32] Wireshark; Libpcap file format
http://wiki.wireshark.org/Development/LibpcapFileFormat#File_Format

[33] Stack Exchange Inc.; Geographic Information Systems;
http://gis.stackexchange.com/questions/8650/how-to-measure-the-accuracy-of-latitude-and-longitude

[34] Google Inc.; Obtaining a Google Maps Android v1 API Key;
https://developers.google.com/maps/documentation/android/v1/mapkey

[35] Google Inc.; Google Maps Android API v2;
https://developers.google.com/maps/documentation/android/

[36] Google Inc.; Class OverlayItem;
https://developers.google.com/maps/documentation/android/v1/reference/com/google/android/maps/OverlayItem

[37] Google Inc.; Class ItemizedOverlay;
https://developers.google.com/maps/documentation/android/v1/reference/com/google/android/maps/ItemizedOverlay

[38] Karlheinz Brandenburg; MP3 and AAC explained; http://telos-systems.com/techtalk/hosted/Brandenburg_mp3_aac.pdf

[39] 3GPP TS 26.090 - Mandatory Speech Codec speech processing functions; Adaptive Multi-Rate (AMR) speech codec; Transcoding functions:
http://www.3gpp.org/ftp/Specs/html-info/26090.htm

[40] G.722.2 – Wideband coding of speech at around 16 kbit/s using Adaptive Multi-Rate Wideband (AMR-WB); July, 2003; http://www.itu.int/rec/T-REC-G.722.2-200307-I/en

[41] Josh Coalson; Free lossless audio codec (FLAC);
http://flac.sourceforge.net/format.html

[42] Wikipedia; MP3; http://en.wikipedia.org/wiki/MP3

[43] Xiph.Org Foundation; Vorbis I specification; March, 2012;
http://xiph.org/vorbis/doc/Vorbis_I_spec.html

[44] Wikipedia; Waveform Audio File Format (WAVE); 1991;
http://en.wikipedia.org/wiki/WAV

[45] ISO/IEC 10918-1:1994 Information technology -- Digital compression and coding of continuous-tone still images: Requirements and guidelines

[46] RFC 2083: PNG (Portable Network Graphics) Specification; T. Boutell; March, 1997

[47] Google Inc.; WebP: A new image format for the Web; https://developers.google.com/speed/webp/

[48] ITU-T Rec. H.263 (01/2005) Video coding for low bit rate communication: http://www.itu.int/rec/T-REC-H.263-200501-I/es

[49] ITU-T Rec. H.264 (01/2012) Advanced video coding for generic audiovisual services: http://www.itu.int/rec/T-REC-H.264-201201-I/es

[50] RFC 2326: Real Time Streaming Protocol (RTSP); H. Schulzrinne, A. Rao, R. Lanphier; April, 1998

[51] RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1; R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee; June, 1999

[52] RFC 2818: HTTP Over TLS; E. Rescorla; May, 2000

[53] Internet-Draft: HTTP Live Streaming draft-pantos-http-live-streaming-08; R. Pantos, W. May; March 23, 2012 – http://tools.ietf.org/html/draft-pantos-http-live-streaming-08

[54] Guigui, Simon; Spydroid-ipcamera; https://code.google.com/p/spydroid-ipcamera/

Annexos

Annex 1: Desenvolupament d'aplicacions Android

Les aplicacions Android s'escriuen en llenguatge Java i és compilen mitjançant les eines de la SDK; que generen fitxers amb extensió *.apk* instal·lables en els dispositius Android. Cada aplicació conviu en el seu propi marc de seguretat: s'executa en un procés Linux separat i té la seva pròpia instancia de la màquina virtual Dalvik. En aquest apartat es descriuen els components i recursos més importants de la plataforma.

A1.1. Fonaments

La SDK d'Android defineix quatre components fonamentals per construir aplicacions: Activity, Service, Content provider o Broadcast receiver. Qualsevol aplicació Android conté, com a mínim, un d'aquests quatre components, que actuen com a punts d'entrada que el sistema pot utilitzar per cridar a l'aplicació.

Cadascun d'aquests components presenta un cicle de vida diferent i serveis per diferents propòsits. A continuació detallem els quatre components:

A1.1.1. Activities

Una activitat ofereix una pantalla amb la qual l'usuari pot interactuar per realitzar alguna tasca. Per exemple, una aplicació d'enviament de missatges presentarà una activitat amb la llista de contactes on seleccionar el destinatari, una altra activitat amb un formulari per escriure el text i finalment un altre activitat on informar del resultat de l'enviament.

Típicament una aplicació està formada per diverses activitats; l'activitat que s'executa inicialment es defineix com principal en el manifest. Qualsevol activitat pot iniciar altres activitats mitjançant els mètodes `startActivity` o `startActivityForResult`. Quan s'inicia una nova activitat, l'actual queda parada però el sistema guarda el seu estat en memòria.

A1.1.2. Services

Els serveis executen processos en segon pla i no inclouen interfície d'usuari. Per exemple, un servei pot gestionar les peticions de xarxa, reproduir música o realitzar operacions sobre fitxers, tot això en segon pla, sense bloquejar la interfície d'usuari.

Un servei es pot iniciar a través de la funció `startService` o bé vincular a un altre component mitjançant la funció `bindService`.

A1.1.3. Content providers

Els proveïdors de contingut gestionen les dades d'una aplicació i permeten (o no) que altres aplicacions puguin consultar o modificar aquestes dades a través d'ells. Per exemple, el sistema Android inclou el proveïdor de contingut per gestionar els contactes; qualsevol aplicació que declari els permisos corresponents pot llegir i escriure la informació dels contactes.

A1.1.4. Broadcast receivers

Els components *broadcast receiver* responen a les crides multidifusió del sistema Android. Moltes d'aquestes crides són llençades des del propi sistema; per exemple quan el nivell de bateria es molt baix o quan la pantalla entre en mode inactiu. Normalment, els components *broadcast receiver* realitzen petites tasques, com iniciar un servei quan reben la comunicació *broadcast*.

A1.2. Recursos

Els recursos en aplicacions Android són els fitxers addicionals i el contingut estàtic que una aplicació necessita utilitzar; com ara imatges, *layouts* o textos. Android permet segmentar aquests recursos en funció de les capacitats del dispositiu. Per exemple, podem definir imatges amb diferents resolucions per aconseguir que siguin el més òptimes possibles per al dispositiu on s'executi l'aplicació.

Els recursos s'organitzen en subcarpetes dins del directori *res/* i s'accedeix a ells a través de l'objecte R. A continuació es detalla la estructura principal d'aquest directori:

res/values

En aquest directori es defineixen els textos, vectors i estils de l'aplicació. El següent exemple mostra com definir un text en un fitxer amb extensió xml.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="nom_app">El nom de l'aplicació</string>
</resources>
```

Un cop definit, podem accedir a aquest recurs des de una activitat:

```
getString(R.string.nom_app);
```

res/layout

La interfície d'usuari es defineix en fitxers xml ubicats al directori *layout* mitjançant els elements gràfics que ofereix Android (*widgets*). Podem accedir a les pantalles definides a través de `R.layout`.

res/drawable

En aquest directori s'emmagatzemen les imatges de l'aplicació. Android també permet guardar animacions predefinides en el directori *res/anim*. El següent exemple mostra com recuperar una imatge ubicada al directori *drawable* des d'una activitat.

```
getResources().getDrawable(R.drawable.ic_launcher);
```

A1.3. Aplicacions natives

La eina *Native Development Kit* (NDK) permet desenvolupar aplicacions que inclouen parts del codi en llenguatge natiu (C/C++). Aquesta eina presenta un gran potencial als desenvolupadors, ja que permet interactuar directament amb el kernel de Linux o ampliar les llibreries d'Android, sense dependre de la API.

Per escriure una aplicació nativa necessitem descarregar la NDK[4], el paquet inclou exemples pràctics i la documentació detallada d'instal·lació i ús. Seguidament crear una carpeta amb el nom ***jni*** al nostre projecte, on ubicarem les llibreries C/C++. Finalment, escriure el fitxer de muntatge ***Android.mk*** i compilar-lo amb l'eina NDK.

[4] Native Development Kit http://developer.android.com/tools/sdk/ndk/index.html

Annex 2: La classe Intent

Els components fonamentals d'Android: `Activity`, `Service` i `BroadcastReceiver`; s'activen a través de missatges denominats *intents*. La classe `Intent` és, en si mateixa, una descripció abstracte d'una operació que s'ha de realitzar. O en el cas dels receptors *broadcast*, és la descripció de com actuar quan es produeix un esdeveniment determinat.

La següent taula mostra els mètodes definits en funció del component que rep l'objecte `Intent`.

Taula A.1 Mètodes més rellevants on intervé l'objecte `Intent`.

Component	Mètode (de la classe `Context`)	Descripció
`Activity`	`startActivity()`	Inicia una activitat o recupera una existent.
	`startActivityForResult()`	Inicia una aplicació que retornarà informació a l'activitat que fa la crida.
`Service`	`startService()`	Inicia un servei o recupera un que ja està en execució.
	`bindService()`	Vincula el component que fa la crida a un servei.
`BroadcastReceiver`	`sendBroadcast()`	Fa arribar l'objecte `Intent` als receptors interessats.
	`sendOrderedBroadcast()`	Difon l'objecte `Intent` de forma ordenada, fent l'entrega als receptors d'un en un. D'aquesta manera el poden consumir primers els receptors preferits.
	`sendStickyBroadcast()`	Difon un objecte `Intent` que es persistent després de consumir-se. Un exemple és el missatge "canvi d'estat de la bateria". Cridant la funció `registerReceiver()` només un cop, sempre rebrem l'última actualització d'aquesta acció.

Les aplicacions del sistema (pre-instal·lades al dispositiu), a més, poden enviar un `Intent` a un usuari concret utilitzant les funcions `sendBroadcastAsUser()`, `sendOrderedBroadcastAsUser()` i `sendStickyBroadcastAsUser()`.

Annex 3: Capacitats multimèdia de la plataforma Android

A3.1. Codecs i contenidors

Android no garanteix que tots els codificadors i descodificadors descrits a la API (i detallats a continuació) estiguin disponibles a tots els telèfons Android. Sempre serà necessari tenir en compte la versió del *software*. De la mateixa manera, és possible que certs terminals ofereixin suport a d'altres formats que no apareixen a la llista. A part dels codecs, també es defineixen els contenidors suportats per la plataforma.

Un contenidor estructura la informació audiovisual (la particiona, protegeix i multiplexa), i descriu el format i el contingut, especificant quin o quins tipus de dades hi ha, com estan estructurats, quina longitud tenen, com s'han de processar, etc. Es a dir, conté totes les meta dades necessàries per poder interpretar-lo de forma correcta.

A3.1.1. Àudio

Els dispositius Android ofereixen tres codificadors d'àudio: AAC LC/LTP, AMR-NB i AMR WB.

AAC LC/LTP (*Advanced Audio Coding*) és un codificador d'àudio amb pèrdues, estandarditzat per la ISO com a part de les especificacions MPEG-2 i MPEG-4 [38]. L'algoritme descarta les components perceptualment irrellevants del senyal i elimina la redundància del flux d'àudio codificat. S'utilitza el perfil de baixa complexitat (*Low Complexity*) i la tècnica LTP (*Long Term Prediction*) que permet reduir la redundància del senyal codificat, ja que està relacionada amb la periodicitat del senya d'àudio.

Aquest codificador està disponible des d'Android 2.3.3 (versió 10 de la API). Els descodificadors AAC estan disponibles a totes les versions, en tres implementacions diferents: AAC LC/LTP, HE-AACv1 i HE-AACv2 (millorat).

Els contenidors suportats són:

- **3GPP** - extensió *.3gp*
- **MPEG-4** - extensió *.mp4* o *.m4a*
- **ADTS** - extensió *.aac* (codificador des d'Android 4.0, descodificador des d'Android 3.1)
- **MPEG-TS (Transport Stream)** - extensió .ts (sense opció de posicionar el cursor de reproducció, a partir d'Android 3.0)

AMR-NB (*Adaptive Multi-Rate - narrowband*), és un codificador optimitzat per a codificació de veu (*vocoder*), i és el codificador estàndard de les xarxes mòbils 3G (3GPP) [39]. És capaç de triar la taxa de codificació més òptima, entre vuit

possibles, en funció de les condicions i requeriments actuals de l'enllaç (*link adaptation*). AMR fa servir una freqüència de mostreig de 8 kHz (ample de banda de 4 kHz, qualitat telefònica) i resolució de 13 bits, amb trames de 160 mostres PCM que després es comprimeixen i donen lloc a trames amb una durada de 20 ms. El senyal queda filtrat a la banda 200 - 3400 Hz (*Narrow Band*).

AMR genera trames de 95, 103, 118, 134, 148, 159, 204 o 244 bits en funció de la taxa de bit escollida 4.75, 5.15, 5.90, 6.70, 7.40, 7.95, 10.2 o 12.2 Kbit/s, respectivament.

El codificador i descodificador AMR-NB estan suportats a totes les versions d'Android. Fa servir el contenidor 3GPP (extensió .3gp).

AMR-WB (*Adaptive Multi-Rate - wideband*), és una extensió de l'AMR, desenvolupada inicialment per Nokia i VoiceAge, definit a l'estàndard ITU-T G.722.2 [40]. Aquest codificador utilitza major ample de banda (50 – 7000 Hz) que es tradueix en una millora de la qualitat de veu.

El codificador genera trames de 132, 177, 253, 285, 317, 365, 397, 461 o 477 bits en funció de la taxa de bit escollida 6.6, 8.85, 12.65, 14.25, 15.85, 18.25, 19.85, 23.05 o 23.85 Kbit/s, respectivament.

AMR-WB està suportat des d'Android 2.3.3 (versió 10 de la API). S'encapsula en el contenidor 3GPP (.3gp).

Finalment, Android suporta els següents descodificadors:

FLAC (*Free Lossless Audio Codec*) és un codificador d'àudio digital sense pèrdues i lliure de patents [41]. Fa servir un algoritme predictiu que permet valors de compressió de fins al 50 o 60% sobre la mida original del fitxer. En funció de la complexitat de la informació. FLAC pot treballar amb qualsevol resolució de mostra PCM (des de 4 fins a 32 bits/mostra), accepta freqüències de mostreig dins del rang 1 – 655350 Hz en increments d'un Hz. Permet des de un fins a vuit canals, Android només dona suport per *Mono/Stereo*.

El format de contenidor és FLAC (extensió .flac) però també es possible encapsular FLAC dins d'un container Ogg. Android només el suporta amb contenidors FLAC.

MP3 (MPEG-1 Layer 3) és un *codec* d'àudio dissenyat per l'Institut Fraunhofer. Utilitza tècniques de codificació perceptual (codificació subbanda i models psicoacústics) per aconseguir factors de compressió alts però garantint una bona qualitat subjectiva [42]. És un format d'àudio especificat per l'estàndard de vídeo MPEG-1.

Utilitza un algoritme asimètric, codificador amb alt cost computacional i descodificador de baix cost, això permet èxit comercial. Està orientat a transmissió gràcies al concepte *frame* (unitat descodificable per si sola). Les

trames (o *frames*) estan formades per: una capçalera de 4 bytes, la signatura CRC (opcional), les dades d'àudio (corresponent a 1152 mostres PCM codificades) i extensions.

Vorbis és un projecte *open source* liderat per *Xiph.Org Foundation*, presenta un format estàndard de codificació d'àudio amb pèrdues, similar a MP3 però lliure de patents [43]. Accepta freqüències de mostreig des de 8 fins a 192 kHz, el codificador genera una sortida de taxa de bit variable (VBR).

L'algoritme converteix la informació d'àudio al domini freqüencial, utilitzant la transformada modificada (MDCT). Després elimina el soroll i els components residuals. Finalment es realitza la quantificació i codificació entròpica.

El flux resultant de la codificació es encapsulat, típicament, dins un contenidor Ogg. Però també es pot encapsular en altres contenidors com Matroska, MPEG-PS/MPEG-TS, WAV, etc., o fins i tot directament al *payload* d'un flux RTP.

PCM/WAVE (Pulse Code Modulation / Waveform Audio File Format). WAVE és un format d'àudio digital propietat de IBM i Microsoft (extensió .wav) [44]. Encara que es compatible amb més codificadors, Android només dona suportat per contenir àudio sense comprimir (PCM). El propi format WAVE limita la mida màxima del fitxer a quatre gigabytes (capçalera amb 32 bits per indicar la longitud del fitxer).

A3.1.2. Imatge

La API presenta tres codificadors (i descodificadors) d'imatge: JPEG, PNG i WebP.

JPEG (*Joint Photographic Experts Group*) és un estàndard de ISO/IEC 10918, per a la codificació d'imatges amb pèrdues [45]. Basat en tècniques de compressió perceptual que permeten descartar els detalls que l'ull humà no pot percebre: eliminació d'altes freqüències i submostreig de crominància (més mostres de luminància o blanc-i-negre que de color).

El codificador segmenta la imatge en blocs de 8x8 píxels, després aplica escalat i transformada freqüencial *Discrete Cosine Transform* (DCT) a cadascun. Realitza el procés de quantificació amb les taules de luminància i crominància. Finalment s'aplica una codificació entròpica (Huffman i Run-Length Encoding) als coeficients.

El format de contenidor és JPEG (extensió .jpg).

PNG (*Portable Network Graphics*), és un format gràfic de compressió d'imatges sense pèrdues [46]. Es pot descriure com l'alternativa a GIF, lliure de patents, i

és el codec recomanat per la W3C en pàgines web. Presenta quatre possibilitats de color: escala de grisos, paleta (afegeix canal de transparència), *truecolor* de 24 bits/pixel (8 bits per a cadascun dels canals blau, verd i vermell, amb un total de 2^{24} = 16.7 milions de colors) i *truecolor* més canal transparència.

El format de contenidor és PNG (extensió .png).

WebP és un format gràfic de compressió d'imatges amb y sense pèrdues, desenvolupat per Google i orientat a web [47]. Està basat en la tecnologia desenvolupada per On2 Tecnologies, empresa comprada per Google. WebP pretén ser el rival directe de JPEG.

El format de contenidor és WebP (extensió .webp). Està disponible des d'Android 4.0.

A part dels codificadors, Android suporta els formats GIF i BMP com a descodificadors.

A3.1.3. Vídeo

La plataforma Android presenta quatre codecs de vídeo: H.263, H.264 AVC, MPEG-4 SP i VP8 (des d'Android 2.3.3).

Els codificadors suportats només són dos, detallats a continuació.

H.263 és un estàndard de compressió de vídeo de la ITU-T, orientat a videoconferències a taxa baixa i successor del codificador H.261 [48]. Presenta les següents característiques:

- Vector de moviment sense límits (pot sortir de la imatge).
- Codificació aritmètica.
- Compensació de moviment avançada (Quatre vectors sobre Mblocs 8x8).
- Imatges PB (codificació més eficient a canvi de més retard).

Els contenidors suportats per H.263 són 3GPP (extensió .3gp) i MPEG-4 (extensió .mp4).

H.264, publicat com ITU-T H.264 i MPEG-4 part 10, és un estàndard de compressió de vídeo, successor tant de H.263 com de MPEG-2. Les aplicacions principals són videoconferència d'alta qualitat a Internet o transmissions *broadcast* [49].

És un codificador dissenyat per xarxa, ja que defineix l'encapsulació RTP juntament amb el codec i incorpora informació redundant a les capçaleres que el fan robust en front errors de transmissió. A més, ofereix flexibilitat de paràmetres: la taxa de bit, la mida dels *buffers* i el retard inicial.

H.264 millora la compensació de moviment, permetent fins a cinc referències en un macrobloc (precisió de fins a un quart de píxel). Aplica una transformada sencera (sense arrodoniments) que permet augmentar la qualitat subjectiva i utilitza versions millorades de codis aritmètics.

Un cop s'apliquen les taules de màscara, es fa un quantització no uniforme (adaptativa) dels coeficients que sobreviuen millorant la SNR de quantització. A més, defineix nivells i perfils orientats a aplicació: *Baseline* (videoconferència, telefonia), *Main* (vídeo qualitat *broadcast*) i *Extended* (per *streaming*).

El perfil base (*Baseline profile*) en els nivells 1, 1b, 1.2, 1.3 i 2 del codificador H264 està disponible des d'Android 3.0 (versió 11 de la API). Permet els següents contenidors 3GPP (extensió .3gp), MPEG-4 (extensió .mp4) i MPEG-TS (extensió .ts, només amb àudio AAC i sense possibilitat de posicionar el cursor).

L'API d'Android suporta els següents protocols de xarxa per a la reproducció d'àudio i vídeo:

A3.2. Protocols de xarxa

RTSP (*Real Time Streaming Protocol*) és un protocol de senyalització dissenyat per a controlar la transmissió i reproducció d'un o diversos fluxos sincronitzats (capacitat multi-servidor) [50]. Habitualment fa servir el port 554. El protocol defineix dos tipus de transmissions: de control i de dades. En l'especificació d'Android, el flux de control utilitza SDP (*Session Description Protocol*) per la descripció dels continguts [18]. El flux de dades es transporta sobre RTP (Real-time Transport Protocol), normalment transportat sobre UDP.

HTTP/HTTPS (*Hypertext Transfer Protocol / HTTP Secure*) és el protocol d'aplicació utilitzat a la *World Wide Web* (WWW), orientat a transaccions y sense informació d'estat (utilitza cookies). A nivell de transport fa servir *Transport Control Protocol* (TCP). Per defecte HTTP defineix el port 80 per les connexions a la banda servidor [51]. En canvi, la versió segura fa servir el port 443 per enviar les dades encriptades d'extrem a extrem a través d'una connexió SSL/TLS [52]. El protocol HTTPS està suportat des d'Android 3.1 (versió 12 de la API).

Android permet dos modes de funcionament en la reproducció de continguts audiovisuals sobre HTTP/HTTPS:

Progressive streaming el concepte de transmissió progressiva descriu la transferència d'un contingut audiovisual entre dos dispositius, amb la característica de que el terminal emissor té emmagatzemat el contingut al disc dur i el transmet “sota demanda”. El receptor pot començar a reproduir el contingut abans d'haver-lo emmagatzemat completament.

Live streaming (transmissió “en viu”) és la transmissió de continguts audiovisuals il·limitats. En aquest procés ni l’emissor ni els receptors emmagatzemen les dades en disc.

Android fa referència a un esbós de protocol (*draft protocol* [53]), on es descriu el format dels fitxers i el comportament que han de seguir, tant l’emissor com els receptors, durant les transmissions “en viu” sobre HTTP/HTTPS. Aquest tipus de documents no són definitius i tenen una validesa de sis mesos. Es qualsevol moment poden ser actualitzats o esdevenir obsolets per altres documents.

Annex 4: Segona versió del servidor d'*streaming*

Després d'implementar el servidor d'*streaming*, modificant el codi annex al PFC de Jordi Tabaré [20], vam realitzar proves en un escenari tancat i amb connexió Wi-Fi. La reproducció del flux de vídeo (tant amb codificació H.263 com H.264) no era gens fluïda: amb talls cada 1 o 2 segons i salts entre *frames*. La reproducció de l'àudio (codificació AMR-NB) era correcta, amb un retard de 3 o 4 segons. Per aquest motiu, i amb la intenció de millorar les prestacions de la nostra aplicació, es va decidir buscar un altre projecte que fes *streaming* RTP/SDP a Android.

El projecte Spydroid-ipcamera [54] ofereix un servidor RTSP (*Real Time Streaming Protocol*) que permet fer *streaming* en viu amb diferents codificadors de vídeo (H.263 i H.264) i d'àudio (AMR-NB i AAC). L'aplicació també inclou un petit servidor HTTP, escoltant al port 8080, que ens ofereix una interfície web des de on podem connectar amb el servidor RTSP i visualitzar l'emissió al navegador. No hem pogut testejar la versió de l'aplicació (8.0) publicada a Google Play en el nostre dispositiu, ja que requereix Android 2.3 o superior; per poder provar-la en un dispositiu amb Android 2.2 hem descarregat el codi font i hem fet les següents modificacions.

Al manifest de l'aplicació (`AndroidManifest.xml`) hem canviat la versió mínima requerida de la API (de la 9 a la 8) i hem forçat que l'aplicació s'instal·li per defecte a la targeta SD.

```
android:minSdkVersion="8"
android:installLocation="preferExternal"
```

Hem comentat les línies 325 fins 330 de la classe `TinyHttpServer` (que és la implementació del servei web) per no fer servir el camp `lastUpdateTime` de la classe `android.content.pm.PackageInfo`, ja que només està disponible a partir de la versió 9 de la API. Les instruccions comentades recuperen el temps en mil·lisegons de la darrera actualització de l'aplicació; en el seu lloc hem posat el següent codi.

```
mLastModified = new Date(0);
```

Finalment, hem modificat la classe `VideoStrem`, concretament les funcions `setCamera()` i `prepare()`, ja que ambdues fan servir parts de la API que només estan disponibles a partir de la versió 9. La primera funció fa servir la classe `CameraInfo` per recuperar el número i identificador de totes les càmeres disponibles al dispositiu; la segona, crida a `Camera.open(int cameraId)` per recuperar la càmera que farà servir l'aplicació. Les modificacions fetes per fer compatible el codi amb Android 2.2 són, per una banda, forçar que l'identificador de la càmera a zero (`this.mCameraId=0`) i, per altra banda, utilitzar la funció `Camera.open()` per recuperar la càmera per defecte.

Després d'aquestes modificacions hem instal·lat l'aplicació al nostre dispositiu HTC Desire i l'hem fet córrer. El resultat ha sigut el desitjat, la reproducció dels fluxos, tant H.263 com H.264, és fluïda. La reproducció de l'àudio AMR-NB és correcta, no hem pogut provar l'encapsulador d'àudio AAC ja que el codificador no està disponible a Android 2.2.

Cal notar que Spydroid-ipcamera utilitza el protocol RTSP, transport de dades amb RTP i connexió RTCP per controlar les sessions multimèdia. A la nostra aplicació, per simplicitat, em decidit treballar amb RTP i fitxers de descripció de sessió SDP.

En la segona implementació del servidor d'*streaming* de l'aplicació CustomCam hem fet servir el sistema de *sockets* i els encapsuladors RTP de l'aplicació Spydroid-ipcamera. A continuació detallem quines classes hem fet servir, i les modificacions fetes. Totes aquestes classes pertanyen a la versió 1.4 de l'aplicació Spydroid-ipcamera; excepte l'empaquetador H.263 que pertany a la versió 8.0.

`AbstractPacketizer.java`

Aquesta classe hereta de `Thread` i implementa la interfície `Runnable`, tots els encapsuladors RTP hereten de la classe `AbstractPacketizer`. Hem eliminat del constructor el paràmetre `InputStream`, ja que l'inicialitzem directament des de l'empaquetador corresponent.

`MediaStreamer.java`

Hereta de la classe `MediaRecorder`, sobreescriu el mètode `prepare()` on crea el sistema de connexions per escriure la sortida al *socket* emissor. El sistema de *sockets* és igual que el de la primera implementació. La classe `MediaStreamer` escriu el flux multimèdia codificat en el *socket* emissor i l'empaquetador s'alimenta d'aquest flux llegint del *socket* receptor.

`AMRNBPacketizer.java`

Hereta de la classe `AbstractPacketizer`, espera rebre un flux AMR en cru per processar les trames i encapsular-les en paquets RTP. Hem afegit un objecte `MediaStreamer` que s'inicialitza al constructor i és l'`InputStream` de la classe pare.

`H263Packetizer.java`

Hereta de `AbstractPacketizer`, espera rebre un flux de vídeo codificat amb l'algoritme H.263 i amb capçalera MPEG-4 o 3GPP. Hem afegir un objecte `MediaStreamer` que s'inicialitza al constructor i és l'`InputStream` de la classe pare. Aquesta classe pertany a la versió 8.0 d'Spydroid-ipcamera, ja que en la versió 1.4 no existia.

`H264Packetizer.java`

Hereta de `AbstractPacketizer`, espera rebre un flux de vídeo codificat amb l'algoritme H.264 i amb capçalera MPEG-4 o 3GPP. Hem afegir un objecte `MediaStreamer` que s'inicialitza al constructor i és l'`InputStream` de la classe pare.

`SimpleFifo.java`

Aquesta classe implementa un cua FIFO (*First In, First Out*). **L**'encapsulador H.264 per una banda, escriu a la cua FIFO les NAL (*Network Abstractaction Layer*) que va generant la càmera i, per l'altra, va llegint de la cua FIFO aquestes NAL, les encapsula en un paquet RTP i les envia al *socket*.

`SmallRtpSocket.java`

És el *socket* que fa servir la classe `AbstractPacketizer`, envia els paquets generats pels encapsuladors. Presenta els mètodes necessaris per treballar amb paquets RTP: escriure capçalera, marca paquet (NAL fragmentada), actualitzar *timestamp*, actualitzar número de seqüència i enviar datagrama.

La figura A.1 mostra el diagrama de classes de la segona implementació del servidor d'*streaming*.

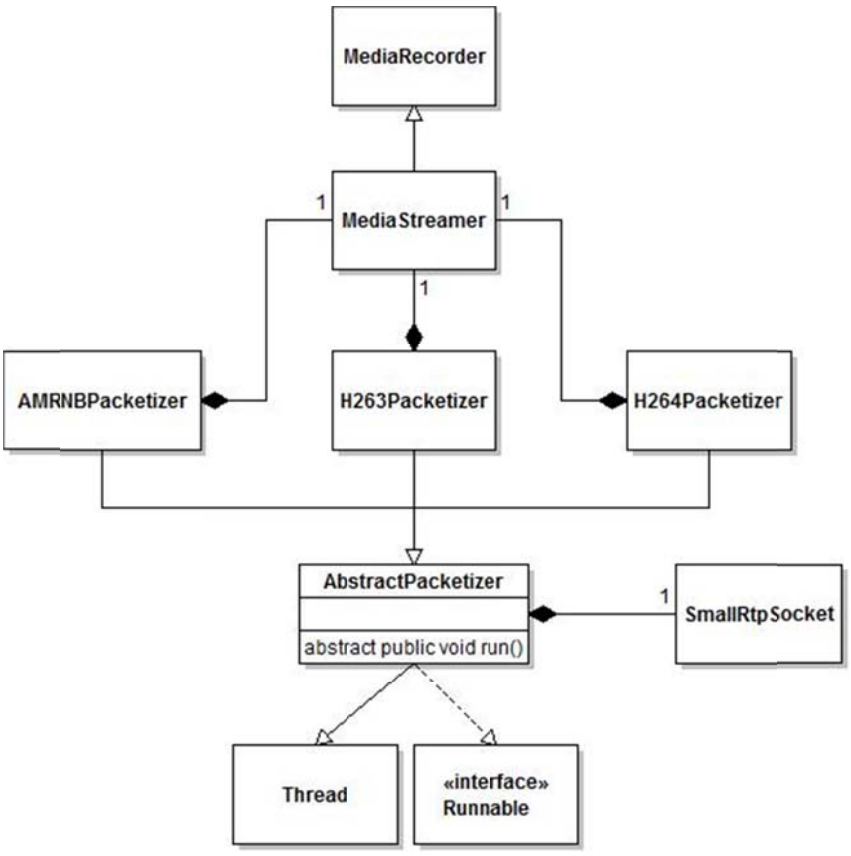

Fig A.1 Diagrama de classes del servei d'*streaming*

Annex 5: Anàlisi de la transmissió d'un flux d'àudio AMR

Analitzem la transmissió d'un flux d'àudio d'un canal (*mono*) suposant que cada paquet RTP només conté una trama AMR. Sabem que la freqüència de mostreig és de 8 kHz = 8000 mostres/s (típic en un senyal de veu telefònica, que té un ample de banda de 4 kHz) i que AMR genera trames que transporten 20 ms de so. Llavors podem calcular el nombre de mostres que contindrà cada trama (i cada paquet).

$$\frac{20\ ms}{trama} x\ \frac{8000\ mostres}{segon} = \mathbf{160\ mostres}$$

Fixant la taxa de codificació a 12.2 kbit/s (GSM-EFR) podem calcular la mida de cada trama en bytes.

$$\frac{20\ ms}{trama}\ x\ \frac{12{,}2\ kbit}{segon} = 244\ bits\ x\ \frac{1\ byte}{8\ bits} = 30.5\ \rightarrow \mathbf{31\ bytes}$$

La transmissió es realitza sobre UDP i les mides de les capçaleres són les mínimes, llavors tenim: Ethernet (14 bytes), IP (20 bytes), UPD (8 bytes) i RTP (12 bytes). A part, la capçalera AMR és d'un byte i s'afegeix un altre byte per descripció de contingut (*payload table of contents*).

$$40\ bytes\ (IP/UDP/RTP)\ + 14\ bytes\ (Ethernet) + 2\ bytes\ (ToC/\ RFC)\ = \mathbf{56\ bytes}$$

Resumint, tenim 56 bytes de capçaleres i 31 bytes de *payload*, un total de **87 bytes** per paquet.

La figura A.2 presenta dues captures Wireshark corresponents a dos paquets consecutius d'una transmissió AMR sobre RTP.

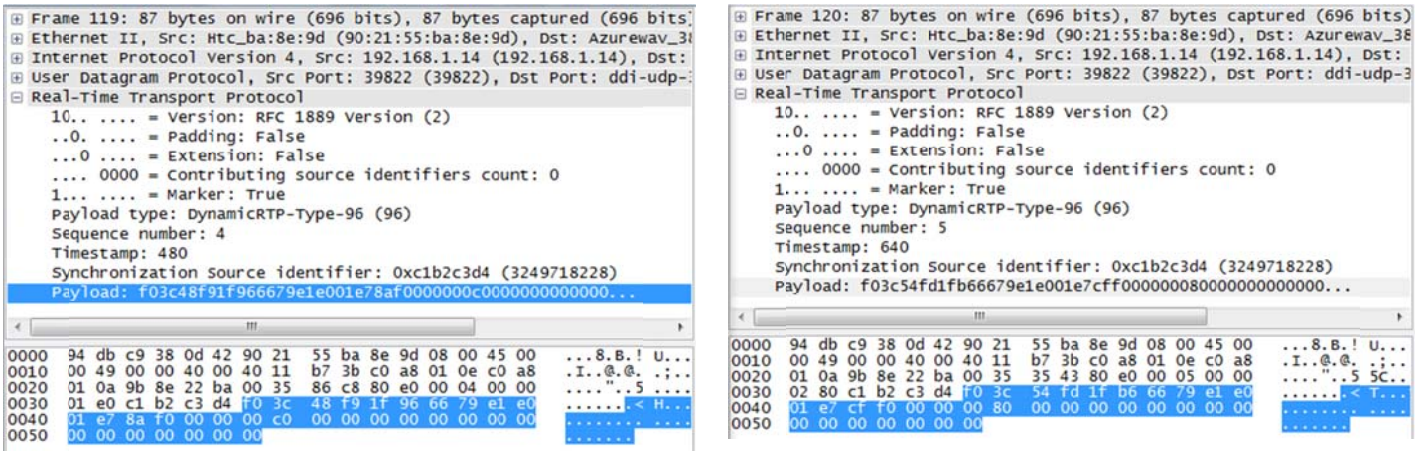

Fig A.2 Paquets consecutius transmissió AMR sobre RTP.

Podem observar que la mida de les trames és la esperada de 87 bytes, tenint 56 bytes de capçaleres (Ethernet + IP + UPD + RTP + AMR) i 31 bytes de càrrega útil.

Per altra banda, si calculem el delta de *timestamp* entre els dos paquets trobem que correspon a la mida de la trama en mostres (160); això es degut a que la freqüència de rellotge del *timestamp* és igual a la freqüència de mostreig (8 kHz).

Annex 6: Paràmetres ajustables de la càmera en un HTC Desire

A continuació adjuntem l'*string* retornat per la funció `Parameters.flatten()` en un dispositiu HTC Desire amb sistema operatiu Android 2.2. Aquest text conté els paràmetres ajustables de la càmera (`Camera.Parameters`) i els seus possibles valors.

El valor actual d'un paràmetre qualsevol el trobem, separat per un símbol igual, darrera del nom del paràmetre:

```
nom-del-parametre=valor_actual
```

Els valors que podem ajustar en el nostre telèfon són: la nitidesa, el zoom, el contrast, la brillantor, el balanç de blancs, la qualitat JPEG, la mida de la vista prèvia, la mida d'imatge, la mida de les miniatures, el flash, el tipus d'enfocament, la compensació a l'exposició, l'efecte de bandes i els efectes de color.

Els possibles valors dels paràmetres queden definits per un màxim i un mínim, aleshores el paràmetre pot prendre un valor lliure entre els extrems definits. En el cas que també es defineixi un valor de pas (*step*), el paràmetre.

```
sharpness-max=30;zoom=0;taking-picture-zoom=0;zoom-
supported=true;sharpness-
min=0;sharpness=10;contrast=5;whitebalance=auto;jpeg-
quality=100;preview-format-values=yuv420sp;jpeg-thumbnail-
quality=75;preview-format=yuv420sp;preview-size=768x432;focal-
length=4.31;iso=auto;meter-mode=meter-center;front-camera-
mode=mirror;flash-mode-values=off,auto,on,torch;preview-frame-rate-
values=15;preview-frame-rate=15;focus-mode-values=auto,infinity;jpeg-
thumbnail-width=768;jpeg-thumbnail-size-values=768x576,640x480,
512x384,0x0;zoom-ratios=100,114,131,151,174,200;saturation-
def=5;preview-size-values=1280x720,800x480,768x432,720x480,640x480,
576x432,480x320,400x240,384x288,352x288,320x240,272x272,240x240,240x16
0,176x144;smart-contrast=off;picture-size-values=2592x1952,2592x1728,
2592x1552,2560x1920,2560x1712,2048x1536,2048x1360,2048x1216,2016x1344,
1600x1200,1584x1056,1280x960,1280x848,1280x768,1248x832,1024x768,640x4
80,640x416,640x384,624x416,512x384,400x400,272x272;contrast-min=0;min-
exposure-compensation=-4;brightness-min=0;antibanding=auto;taking-
picture-zoom-min=0;saturation-min=1;contrast-max=10;vertical-view-
angle=42.5;taking-picture-zoom-max=20;contrast-def=5;brightness-
max=6;horizontal-view-angle=54.8;brightness=3;jpeg-thumbnail-
height=576;cam-mode=0;focus-mode=auto;sharpness-def=10;front-camera-
mode-values=mirror,reverse;picture-format-values=jpeg;saturation-
max=10;max-exposure-compensation=4;exposure-compensation=0;exposure-
compensation-step=0.5;flash-mode=off;effect-values=none,mono,negative,
solarize,sepia,posterize,aqua;meter-mode-values=meter-average,meter-
center,meter-spot;picture-size=1024x768;max-zoom=5;effect=none;
saturation=5;whitebalance-values=auto,incandescent,fluorescent,
daylight,cloudy-daylight;picture-format=jpeg;brightness-def=3;iso-
values=auto,deblur,100,200,400,800,1250;antibanding-
values=off,50hz,60hz,auto
```

Annex 7: Implementació Java de l'algoritme d'agregació

El següent algoritme, escrit en Java, a partir d'un llistat de localitzacions (`locations`) i un llistat de paquets (`packets`) determina quina localització està més a prop, en temps, de cadascun dels paquets.

```
if (locations.size() > 0) {
  for (int i= 0; i< locations.size(); i++) {
    fragmentPackets = new ArrayList<JPacket>();
    long ti = locations.get(i).getTime();
    long tii= (i < locations.size()-1) ?
      locations.get(i+1).getTime() : locations.get(i).getTime();

    if (packets.size() > 0) {
      for (int j= pos; j< packets.size(); j++) {
        long tj = packets.get(j).getCaptureHeader()
          .timestampInMillis();

        if (Math.abs(ti - tj) < Math.abs(tii-tj)) {
          fragmentPackets.add(packets.get(j));
          if (j==(packets.size()-1)) {
            pos = packets.size();
            mMarker.addOverlay(new LocatedPacket(
              locations.get(i), fragmentPackets));
            break;
          }
        } else {
          if (ti == tii) {
            fragmentPackets.add(packets.get(j));
            if (j==(packets.size()-1)) {
              mMarker.addOverlay(new LocatedPacket(
                locations.get(i), fragmentPackets));
            }
          } else {
            pos = j;
            if (fragmentPackets.size() > 0) {
              mMarker.addOverlay(new LocatedPacket(
                locations.get(i), fragmentPackets));
            } else {
              onlyLocationMarker.addOverlay(new LocatedPacket(
                locations.get(i), fragmentPackets));
            }
            break;
          }
        }
      }
    } else {
      onlyLocationMarker.addOverlay(new LocatedPacket(
        locations.get(i), fragmentPackets));
    }
  }
}
```

Annex 8: Anàlisi de la distribució del temps entre localitzacions

En aquest apartat analitzem les dades capturades pel servei LocatedTraffic el dia 14 d'abril, durant un trajecte en cotxe per l'autopista C-32 en direcció Premià de Dalt. Durant la prova es van rebre un total de 587 localitzacions i 635 paquets de xarxa que, un cop processats amb l'algoritme presentat a l'annex 7, els 587 paquets queden agrupats en 62 localitzacions.

Hem creat un histograma de la variable "temps entre localitzacions amb tràfic" seguint els següents passos:

1. Calculem el temps entre les 62 localitzacions amb tràfic (61 intervals).

2. Amb l'eina anàlisi de dades de l'Excel definim rangs temporals des de el mínim fins al màxim dels valors calculats al primer punt i comptem quants intervals cauen a cadascun d'aquests rangs.

3. Finalment representem les dades en un gràfic i l'ajustem a una exponencial negativa.

La figura A.3 mostra l'histograma de la variable "temps entre localitzacions amb tràfic" de les dades capturades el 14 d'abril.

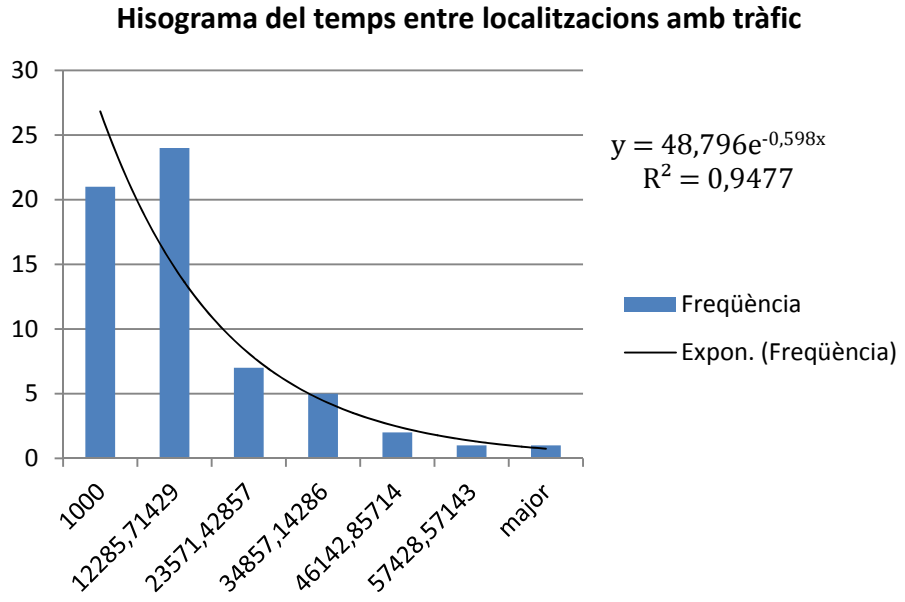

Fig. A.3 Histograma del temps entre localitzacions amb tràfic

Cal destacar que el volum de dades analitzat és molt reduït per considerar que és concloent. No obstant, després d'aquesta primera aproximació, els resultats semblen corroborar la hipòtesis de partida (la creació de localitzacions amb tràfic segueix una distribució de Poisson), ja que la similitud entre el model teòric i el comportament de les dades reals és alt (R^2=0,947).

Per altra banda, a l'aproximació del capítol 5 hem determinat una lambda de 0,15 unitats/segon, aquest valor no correspon amb la taxa d'arribades trobada a partir de l'anàlisi de les dades reals ($\lambda{=}0{,}598 \rightarrow \lambda{=}0{,}6$), llavors el temps mig entre arribades és 1/λ = 1,667 segons.

Això ens dona la possibilitat de reduir l'interval de temps a 2 segons, d'aquesta manera assegurem que seguim cobrim el temps mig entre arribades i augmentem la precisió de les ubicacions. La probabilitat de que es produeixin una o més arribades en un interval de dos segons amb λ=0,6 és:

$$P_{n>0}(2) \;=\; 1 - P_0(2) = 1 - \frac{(0{,}6 \cdot 2)^0}{0!} e^{-0{,}6 \cdot 2} = 0{,}698 \rightarrow\; 69\%$$

ara, teòricament, el 69% de les localitzacions són útils per geolocalitzar el tràfic. Per contra, l'estalvi d'energia és menor, ja que només reduïm a la meitat la taxa d'actualitzacions d'ubicació.

D'altra banda, suposant que tenim la mateixa precisió en les ubicacions que a l'anàlisi del capítol 5, amb l'interval de 2 segons suportem velocitats més altes sense desvirtuar les dades.

$$|v \cdot 2segons| > \; Accuracy$$

En resum, l'elecció de l'interval de temps és un compromís entre precisió d'ubicació i estalvi d'energia (sempre assegurant que cobrim el temps mig entre arribades). Per exemple, si en comptes de triar un interval de 2 segons fem servir un de 3 segons trobem que:

$$P_{n>0}(3) \;=\; 1 - P_0(3) = 1 - \frac{(0{,}6 \cdot 3)^0}{0!} e^{-0{,}6 \cdot 3} = 0{,}834 \rightarrow\; 83\%$$

es millora l'eficiència de les localitzacions, ja que només el 17% no són útils per geolocalitzar i es redueix la taxa d'actualitzacions en un factor 3 (comparant amb 1 localització/segon). Per contra, perdem precisió en les ubicacions i suportem velocitats menys altes.