Departamento de Ingeniería Telemática

PROYECTO FIN DE CARRERA

Estudio de la arquitectura DTN y desarrollo de una aplicación Android como caso de uso

Autor: Daniel de la Casa Riballo

Tutor: Mª Celeste Campo Vázquez

Leganés, 15 de julio de 2015

Título: ESTUDIO DE LA ARQUITECTURA DTN Y DESARROLLO DE UNA APLICACIÓN ANDROID COMO CASO DE USO

Autor: DANIEL DE LA CASA RIBALLO

Director: Mª CELESTE CAMPO VÁZQUEZ

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 15 de Julio de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Quiero empezar agradeciendo a mi familia, principalmente a mis padres Clemente y Ana, a mis hermanos María y Jesús, a mis cuñados Rocío y David, por haber aguantado durante tantos años, esta montaña rusa que ha sido la carrera. A mis sobrinas, Luna y Victoria, que desde que nacieron, me recuerdan siempre en esta faceta de mi vida, a partir de ahora espero poder devolverles todo el cariño que me han dado, ellas han sido un motivo siempre para seguir adelante. A todos mis tíos y mis primos que me han apoyado y me han guiado en esta larga aventura.

A aquellos que empezaron siendo compañeros de carrera, pero han terminado convirtiéndose en amigos. Especialmente a Javi y Josu. Ambos han sido un apoyo y un ejemplo a seguir durante estos años, especialmente en la recta final. Sin su apoyo, su cariño y su generosidad no habría sido posible terminar este reto. Gracias Javi por haber sido un apoyo imprescindible para mí, en estos últimos meses, por haber sabido escucharme y haber intentado comprenderme siempre. A partir de ahora llegarán los buenos tiempos, que tanto esfuerzo y sufrimiento nos han costado, llegó la hora de empezar a disfrutarlos. Gracias a Dani, que aunque compartimos pocos años en la carrera, aún seguimos siendo amigos, y es una gran ayuda y motivación para mí.

Agradecer también a todos los amigos de la infancia. Quizá ellos sean la parte más dura que se ha llevado esta carrera. Tengo que daros las gracias por haberme apoyado y haber sabido entender cuál era mi futuro, y seguir ahí durante tantos años, aunque nos veamos muy poco. Gracias Aroa, Carlos, Jacobo, Javi, Juan, Paloma, Rafa, Sara, siempre recordaré esas tardes interminables en la pista, que pasamos durante tantos años.

Gracias a mi tutora, Celeste, por guiarme en la última fase de esta aventura. A todos los profesores y compañeros que me han aportado sabiduría, experiencia, valores y sobre todo, tantos momentos inolvidables para mí.

A mis compañeros de trabajo, especialmente a Fran, Alberto, Juanma, Óscar y Julio que tantos buenos ratos me han hecho pasar. Gracias a mis jefes, Julio, Joan y Sergi que me han dado todo el apoyo que cualquiera hubiera deseado para poder acabar esta carrera.

No quiero olvidarme de toda la gente importante para mí, que nos ha dejado durante estos años de estudio. Abuelos, tíos, familia… Especialmente a ti, Julia, que tanto me has hecho aprender de la vida y tanto me has ayudado a seguir adelante, aunque nunca haya podido escuchar tu voz, ni tenerte entre mis brazos. Sé que allá donde estés me has ayudado a finalizar este reto y seguirás siendo una luz y una guía durante el resto de mi vida.

El que da, no debe volver a acordarse; pero el que recibe nunca debe olvidar.

A todos,

Muchísimas gracias

Resumen

Este proyecto está enfocado en el estudio del protocolo DTN para redes tolerantes al retardo. A través de este proyecto se pretende realizar una aplicación piloto para el sistema operativo Android que haga uso de esta tecnología de comunicación.

El protocolo DTN es un estándar (RFC 4838) de tipo informativo del IETF definido en 2007. A pesar de ello no se encuentra demasiado extendido, pero las posibilidades en redes que se conecten ocasionalmente o que presenten cortes, hacen de este protocolo un elemento de estudio interesante para resolver todos los retos que se presentan en este tipo de entornos.

El aumento de los dispositivos móviles y del flujo de datos que navegan por internet actualmente, hace necesaria la reflexión sobre la implementación de protocolos de este tipo, de manera puntual, para entornos aislados o de difícil conectividad para los usuarios. Empleando el protocolo DTN, los usuarios serán capaces de comunicarse con otros dispositivos cuando las posibilidades de conexión a internet sean escasas. Los *bundles* son la unidad de datos del protocolo DTN. Empleando el uso de *bundles*, se podrá realizar la comunicación, que será efectuada en el momento en el que el dispositivo recupere la conexión.

El objetivo de este Proyecto Fin de Carrera es demostrar la usabilidad del protocolo para ser empleado en aplicaciones de terminales móviles y para ello se ha desarrollado un piloto de aplicación Android que utiliza DTN para la comunicación entre dispositivos. Esta aplicación será capaz de transmitir información geolocalizada al resto de usuarios que estén empleando el protocolo DTN.

Para la implementación del protocolo DTN se ha utilizado un desarrollo empleado por el grupo IBR-DTN, que se encuentra disponible en Google Play en el momento de la finalización de este proyecto.

Palabras clave: DTN, Redes Tolerantes al Retardo, Android, Aplicaciones Móviles, IBR-DTN

Abstract

This project is based on the study of the DTN protocol for Delay Tolerant Networks. Throughout this project, the final goal is to develop a pilot application based on the Android Operating System using this communication technology.

DTN protocol is an informative IETF standard (RFC 4838) defined in 2007. However, it is not fully accepted or implemented around the world, but the possibilities of communication on networks that connect occasionally or with cuts make this protocol an interesting study to solve the different challenges present in this kind of environment.

The increase of the number of mobile devices available and the amount of data they transmit through the internet make the implementation of protocols like DTN necessary to be used every now and then on hostile environment for the user's communication. Using DTN protocol, users will be able to communicate between different devices, when internet communication is not available, or with very poor coverage. Bundles are the data unit of the DTN protocol. With the use of the bundle protocol, it is be possible to make the communication that would be resumed when the device gets the coverage back from the network.

The goal of this Final Degree Proyect is to study the usability of the protocol to be used in mobile applications in mobile devices. A pilot Android application has been developed to show the DTN usage in communication between mobile devices. This application will be able to transmit geolocalized information to the different subscribers around, using the DTN protocol.

For the DTN protocol implementation it has been used a development environment created by the IBR-DTN group that is available for free on Google Play when this project was finished.

Keywords: DTN, Delay Tolerant Networks, Android, Mobile Applications, IBR-DTN

Índice de Contenidos

Índice de Figuras

Índice de Tablas

Capítulo 1: Introducción y Objetivos

1.1. Introducción

En este capítulo se realizará una pequeña introducción al proyecto, se verán los diferentes objetivos, medios con los que se ha contado para realizar el proyecto y la estructura de la memoria.

Este proyecto se enmarca en los problemas para la transmisión de información que existe entre las redes con mucho retardo o con problemas de conexión. Para estudiar posibles soluciones a este problema, se plantea el estudio del protocolo DTN. Este protocolo, mediante el uso de *bundles*, pretende solucionar el problema de la transmisión de datos en entornos con retardo.

La RFC del protocolo DTN [1] plantea la posibilidad de establecer mecanismos entre los diferentes dispositivos con capacidades DTN como el de almacenamiento y envío para establecer una comunicación entre sus usuarios en redes y entornos propensos al retardo.

El grupo IBR-DTN ha creado una capa de *software* para ser empleada como librería para desarrollar aplicaciones que empleen el protocolo DTN [2]. La aplicación desarrollada en este proyecto empleará la librería desarrollada para las implementaciones en Android. Por este motivo, para ejecutar la aplicación será necesario disponer en los dispositivos de la aplicación IBR-DTN, que contiene el demonio DTN y toda la capa de comunicación del protocolo. La aplicación desarrollada en este proyecto utilizará esa capa para realizar la comunicación.

La aplicación Android se establece en este proyecto como un proyecto piloto, y con visión de poder ser evolucionada y ampliada en el futuro, para integrar más funcionalidades del protocolo, así como una mayor evolución y desarrollo dentro de las novedades de la programación en Android. La alta penetración que aún existe en la versión 4 de Android cuando se realizó este proyecto, hace justificable el desarrollo de la aplicación con el objetivo en esa versión específica.

Este proyecto pretende poner de manifiesto uno de los retos más importantes al que se enfrenta el mundo de las telecomunicaciones en la actualidad. La necesidad de comunicación continua en el que se encuentra actualmente la sociedad hace necesarias implementaciones empleando protocolos como DTN para ayudar a la comunicación en entornos difíciles, donde la conexión a internet puede tener una cobertura inferior a la deseable. Mediante el uso de este tipo de protocolos, se ataca este problema, proponiendo las diferentes soluciones que se detallarán en el apartado apropiado del segundo capítulo de este proyecto.

1.2. Objetivos

El objetivo fundamental de este proyecto es el de estudiar el protocolo DTN y realizar una aplicación Android sencilla que haga uso de este protocolo para las tareas de comunicación entre dispositivos. Este objetivo puede subdividirse en diferentes subobjetivos:

- Estudiar de la RFC 4838 [1] y de la RFC 5050 [3] del IETF. Analizar las principales características del protocolo DTN y del protocolo de *bundles*.
- Analizar el Sistema Operativo Android, especialmente de los elementos de la versión 4, para realizar el desarrollo de la aplicación enfocada en los dispositivos de este tipo.
- Estudiar la implementación de la pila de DTN, creada por el grupo IBR-DTN [2].
- Configurar y manejar el demonio IBR-DTN que facilita la transmisión y recepción de *bundles*.
- Probar con las aplicaciones existentes del grupo IBR-DTN [4] para justificar y demostrar la utilización de este protocolo por el demonio DTN y las aplicaciones que se desarrollan encima de esta pila.
- Implementar de una aplicación Android, que sea compatible con los dispositivos de la versión 4 del sistema operativo.
- Realizar diferentes pruebas en dispositivos diversos (teléfonos móviles, tablets) comprobando los requisitos de la aplicación.
- Documentar los resultados obtenidos y todas las conclusiones recopiladas en la elaboración de este proyecto.

1.3. Fases del Proyecto

Este proyecto ha estado dividido en cuatro fases principales que se especifican a continuación:

- **Fase I**: Recapitulación de información acerca del protocolo DTN, del desarrollo en Android y de la implementación de la aplicación IBR-DTN empleada para realizar las comunicaciones utilizando el protocolo *bundle*.
- **Fase II**: Análisis de la información y desarrollo de la idea principal para la implementación de una aplicación piloto en Android para demostrar la utilidad del protocolo DTN.
- **Fase III**: Especificación e implementación de la aplicación Android, desarrollo e instalación del entorno y pruebas de la aplicación.
- **Fase IV**: Documentación del proyecto, defensa del trabajo realizado y de los resultados obtenidos.

1.4. Medios Empleados

Para la realización de este proyecto ha sido necesaria la utilización de un ordenador personal con la memoria y el procesador suficiente para ejecutar el SDK de desarrollo en Android sin inconvenientes en su modo de depuración.

Para las diferentes pruebas realizadas en la memoria, además, ha sido necesaria la utilización de varios terminales móviles que emplearan el sistema operativo Android. Los terminales empleados para las pruebas detalladas en esta memoria son los siguientes:

- Teléfono móvil HTC One M7. Versión de Android: 5.0.2
- Teléfono móvil Sony Xperia Arc. Versión de Android: 4.0.4
- Tablet BQ Edison 2. Versión de Android: 4.2.2
- Tablet Samsung Galaxy Tab Pro 4. Versión de Android: 4.4.2

Para el correcto funcionamiento de la aplicación desarrollada en este proyecto, ha sido necesaria la instalación en estos dispositivos de la aplicación Android IBR-DTN, que contiene el demonio DTN. La versión instalada es la 1.0, dtnd:1.0.1, build:570f2b9.

1.5. Estructura de la Memoria

Para facilitar la lectura de la memoria, se incluye a continuación un breve resumen de cada capítulo de los que está compuesta esta memoria.

El *Capítulo 1* contiene una introducción especificando en qué consiste el proyecto, así como los objetivos principales del mismo. También se incluyen los medios empleados para la realización de este proyecto y las diferentes fases de las que se compone.

En el *Capítulo 2* se han especificado, de forma resumida, las diferentes características y funciones del protocolo DTN. Además, se realiza también un pequeño estudio del sistema operativo Android, principalmente de la versión 4, así como los diferentes elementos utilizados en la aplicación desarrollada para el proyecto.

Es posible encontrar en el *Capítulo 3* los detalles de la implementación del demonio DTN realizado por el grupo IBR-DTN. Se incluye en este apartado la configuración empleada en este proyecto, así como una serie de pruebas realizadas para demostrar la utilización del protocolo DTN por esta aplicación.

El capítulo principal del proyecto es el *Capítulo 4*. En él se explica el desarrollo de la aplicación Android *Around*, que es la aplicación DTN llevada a cabo como fin y demostración del sistema DTN desarrollada como piloto en este proyecto. Se encuentran detallados el diseño, las especificaciones y las pruebas realizadas.

En el *Capítulo 5* se incluyen las conclusiones y los trabajos futuros.

En el último apartado se incluyen una serie de anexos: el presupuesto, el manual de usuario y un diagrama de *Gantt* con la evolución del proyecto. Por último se incluyen las referencias y un glosario de términos.

Capítulo 2: Estado del Arte

2.1. Introducción

En este capítulo se tratará sobre el estado del arte de los dos pilares básicos de este proyecto. Para empezar, se realizará un estudio del protocolo DTN en el que está basado este proyecto. Por otra parte, se realizará un pequeño resumen de las partes más importantes de una aplicación Android, puesto que será la plataforma en la cual se desarrollará el piloto para este proyecto.

2.2. DTN

2.2.1. Introducción

DTN es una RFC desarrollada por el IETF, del año 2007, que está enfocada en definir una arquitectura de comunicaciones para redes que se conectan de forma ocasional y pueden sufrir de cortes en la comunicación [1]. Esta arquitectura está basada en la creación de un Internet Interplanetario, centrándose en los problemas de comunicaciones en el espacio profundo en entornos con alto retardo. Puede utilizarse en varios entornos operacionales, sobre todo aquellos sujetos a cortes e interrupciones, así como en situaciones de alto retardo.

También es una arquitectura interesante para conectar sensores que necesitan conectividad de forma intermitente siguiendo una serie de patrones temporales. Se podría utilizar para redes terrestres inalámbricas que no pueden mantener conectividad *end-to-end*, redes de satélites con retardos moderados y conectividad periódica, redes acuáticas subterráneas con retardos moderados y comunicación poco frecuente debido a factores medioambientales.

Se define una capa orientada a mensajes, *end-to-end*, llamada capa *bundle* [3] que se encuentra por encima de la capa de transporte de los dispositivos en los que se implementa y por debajo de las aplicaciones. Los dispositivos que implementan esta capa se llaman nodos DTN. Esta capa emplea almacenamiento persistente para combatir los cortes de comunicación en la red. Incluye la transferencia salto a salto de forma responsable, utilizando *end-to-end* ACKs de forma opcional. A su vez se incluyen en la capa una serie de herramientas diagnósticas y de mantenimiento para su uso. Centrándose en la interoperabilidad del protocolo, se utiliza un esquema de nombres flexible basado en los URIs (*Uniform Resource Identifiers* [5]) que permite encapsular los nombres y los esquemas de direccionamiento utilizando la misma sintaxis. Emplea un modelo básico de seguridad, que se puede habilitar de forma opcional, pensado para proteger la infraestructura de usos no autorizados.

La capa *bundle* proporciona funcionalidad parecida a la capa de internet para *gateways* descrita originalmente en los diseños de ARPANET/Internet. Se diferencia en

que no se centra en las capas, sino que pretende enviar los mensajes virtuales (más que hacer conmutación de paquetes). Ambas proporcionan interoperabilidad entre los protocolos de capas inferiores y entre estos protocolos a los de las superiores. También las dos proporcionan un servicio de *store-and-forward* (almacenamiento y envío) al servicio siguiente (la capa *bundle*, como se indicó previamente utiliza almacenamiento persistente para esta función).

La arquitectura DTN proporciona un método para interconectar puertas de enlace heterogéneas o *proxys* que emplean enrutamiento de mensajes con el modelo *store-and-forward* (almacenamiento y envío) para superar las interrupciones de las comunicaciones. Es un servicio similar al *e-mail*, pero con nombres evolucionados, enrutamiento, y opciones de seguridad. Los nodos que no puedan soportar todos los requisitos de la arquitectura se podrán utilizar como *proxys* por la capa de aplicación, funcionando como aplicaciones DTN.

El motivo por el cual se define DTN es porque los protocolos de Internet existentes no funcionan bien en algunos entornos debido a algunos de los supuestos con los que trabaja la arquitectura de Internet:

- Debe existir un camino *end-to-end* entre la fuente y el destino mientras dure la comunicación (o sesión).
- Para las comunicaciones fiables, las retransmisiones están basadas en una serie de mensajes enviados por el receptor de la información que se utilizan para reparar posibles errores.
- Las pérdidas *end-to-end* son relativamente pequeñas.
- Todos los *routers* y nodos finales soportan los protocolos TCP/IP.
- Las aplicaciones no necesitan preocuparse sobre los resultados de las comunicaciones.
- Los mecanismos de seguridad en el nodo final son suficientes.
- La conmutación de paquetes es la abstracción más importante entre la interoperabilidad y el funcionamiento adecuado de las comunicaciones.
- Seleccionar una única ruta entre transmisor y receptor es suficiente para tener una comunicación satisfactoria.

La arquitectura DTN se concibe para relajar estas presunciones, basándose en los siguientes principios de diseño:

- Utilizar mensajes de longitud variable (no utilizar *streams* o paquetes de tamaño limitado) para ayudar a aumentar las capacidades de la red, para tener una buena planificación y para poder tomar decisiones de enrutamiento cuando sea posible.
- Utilizar una sintaxis que soporte un gran rango de nombres y de direcciones que se ajusten a los convenios para asegurar la interoperabilidad.
- Utilizar almacenamiento interno a la red que soporte las operaciones de *store-and-forward* sobre múltiples caminos y en largos periodos de tiempo (los caminos *end-to-end* podrían ni existir). No se requiere fiabilidad *end-to-end*.
- Proporcionar mecanismos de seguridad que protejan la infraestructura de tráfico no autorizado, descartando los mensajes tan rápido como sea posible.

- Permitir clases de servicio con alta granularidad, opciones de entrega y un método para expresar la utilidad temporal de los datos que permita a la red entregarlos garantizando las necesidades de las aplicaciones.
- Las aplicaciones deben minimizar el número de intercambios RTT (*Round Trip Time*).
- Las aplicaciones deben hacer frente a los reinicios tras los fallos mientras haya transacciones pendientes por la red.
- Las aplicaciones deben informar a la red de la vida útil así como de la importancia relativa de los datos que van a ser transmitidos.

2.2.2. Descripción de la Arquitectura DTN

2.2.2.1. Switching Virtual de Mensajes usando Store and Forward

Una aplicación que utiliza DTN puede mandar mensajes de longitud arbitraria, llamados ADUs (*Application Data Units*), que pueden limitarse según la implementación. El orden relativo de los ADUs podría no mantenerse. Típicamente se envían y se reciben en unidades completas.

Estos mensajes se transforman por la *bundle layer* en una o más unidades de protocolo llamadas *bundles*, que se envían por los nodos DTN. Los *bundles* tienen un formato definido y contienen dos o más bloques de datos. Cada bloque puede contener datos de la aplicación u otra información utilizada para la entrega del *bundle* en el destino. Los bloques son el equivalente de la cabecera o el *payload* usado en otros protocolos. El término bloque se utiliza en lugar de cabecera porque puede que el bloque no se encuentre al comienzo del *bundle* por algún requisito específico.

Los *bundles* se pueden fragmentar en múltiples unidades (llamados "*bundle fragments*" o fragmentos) durante la transmisión. Los fragmentos son *bundles* en sí mismos y pueden ser refragmentados. Dos o más fragmentos pueden reensamblarse en cualquier lugar de la red formando un nuevo *bundle*.

El origen y el destino de los *bundles* se obtienen con los EIDs (*Endpoint Identifiers)* de longitud variable que identifican el origen y el destino final de los *bundles*. También contienen un EID de *report-to* utilizado cuando operaciones especiales requieren el diagnóstico directo por una entidad arbitraria (diferente del origen por motivos de seguridad). Un EID puede asociarse a uno o varios nodos DTN (direcciones *multicast* o *report-to*).

Las redes IP se basan en operaciones de *store-and-forward*. Existe la asunción de que el almacenamiento no es persistente por una cantidad grande de tiempo, en función de la cola y del retardo de transmisión. En contraste, la arquitectura DTN no espera que los puntos de la red sean fiables o estén disponibles, por lo que se espera que los nodos almacenen los *bundles* durante algún tiempo. La mayoría de los nodos DTN usarán alguna forma de almacenamiento persistente (disco, memoria flash…) para que los *bundles* almacenados puedan sobrevivir a un reinicio del sistema.

Los *bundles* tienen un *timestamp* de origen, un tiempo de vida útil, un designador de clase de servicio, y una longitud. Esta información proporciona enrutamiento a la capa *bundle* con un conocimiento previo del tamaño y de los requisitos de transferencia

de los datos a transmitir. Cuando hay una cantidad significativa de encolamiento en la red, la ventaja de conocer inicialmente esta información puede ser relevante para tomar decisiones de enrutamiento y planificación. Un enfoque alternativo (como la entrega basada en *streams* de paquetes) haría que la planificación fuera más difícil. Aunque los paquetes proporcionan algunas de las características de los *bundles*, las agregaciones mayores proporcionan una forma de aplicar una mejor planificación a la red, así como el control de los *buffers* en unidades de datos más útiles para las aplicaciones.

Un elemento esencial de esta implantación en *bundles* es que tienen un lugar para esperar en la cola hasta que se produce una oportunidad para la comunicación (contacto). Para ello, se asumen los siguientes detalles:

1. El almacenamiento está disponible y bien distribuido alrededor de la red.
2. El almacenamiento es lo suficientemente persistente y robusto para almacenar *bundles* hasta que se puedan enviar.
3. El modelo de *store-and-forward* es la alternativa más interesante a realizar continuos intentos de conexión u otras opciones.

Para que una red soporte eficientemente la arquitectura DTN deben considerarse estas opciones e implementarse. Aun así, la inclusión de almacenamiento persistente, aspecto fundamental, presenta nuevos problemas; especialmente respecto al control de la congestión y la mitigación de los ataques de denegación de servicio. El almacenamiento en los nodos, en esencia, representa un nuevo recurso que debe ser controlado y protegido.

2.2.2.2. Nodos y Endpoints

Un nodo DTN se utiliza para enviar y recibir *bundles*, implementando la capa de *bundle*. Las aplicaciones utilizan nodos DTN para enviar o recibir ADUs transportadas en los *bundles* aunque también usan nodos DTN cuando actúan como *report-to destinations* para el diagnóstico de la información transportada en *bundles*.

Los nodos pueden ser miembros de grupos llamados DTN *endpoints*. Un DTN *endpoint* es un conjunto de nodos DTN. Se considera que un *bundle* se ha entregado correctamente a un DTN *endpoint* cuando un subconjunto mínimo de nodos en el DTN *endpoint* ha recibido el *bundle* sin errores. Este subconjunto se llama MRG (*Minimum Reception Group*) del *endpoint*. Un MRG de un *endpoint* puede ser un nodo (*unicast*), un nodo de un grupo de nodos (*anycast*) o todos los nodos de un grupo (*multicast* y *broadcast*). Un nodo puede estar en el MRG de múltiples *endpoints*.

2.2.2.3. EIDs y Registros

Un EID (*Endpoint Identifier*) es un nombre, utilizando la sintaxis general de una URI, que identifica un "DTN *endpoint*". Utilizando un EID, un nodo es capaz de determinar el MRG del "DTN *endpoint*" nombrado por el EID. Cada nodo tiene que tener al menos un EID que lo identifica de forma única.

Las aplicaciones envían ADUs destinadas a un EID y pueden conseguir que los ADUs enviados a un EID particular les sean entregados. En función de la construcción del EID

utilizado puede emplearse como “comodín” alguna parte del EID, útil para el diagnóstico y enrutamiento.

El deseo de una aplicación de recibir ADUs destinados a un EID particular se denomina registro, y en general se mantiene persistentemente por el nodo DTN. Esto permite que la información de registro de la aplicación sobreviva a un reinicio de la misma o del sistema operativo. No está garantizado el éxito del proceso de registro por parte de la aplicación. Por ejemplo, una aplicación podría requerir registrarse para recibir un EID específico que no puede interpretarse o no está disponible en el nodo DTN.

Esquemas de URI

Cada EID Se expresa sintácticamente como un URI (*Uniform Resource Identifier* [5]). Esta sintaxis se diseñó como una forma de expresar nombres o direcciones para un amplio rango de propósitos, por lo que es útil para construir nombres para los nodos DTN.

Según esta terminología, cada URI comienza con un *scheme name* (nomenclatura IANA). Es una serie de caracteres que se ajustan a la sintaxis definida. Esta parte de la URI se llama SSP (*scheme-specific part*) y puede ser muy general.

Los esquemas de URIs son un concepto clave en la arquitectura DTN. Evolucionaron de un concepto previo: las regiones, que se relacionaban intrínsecamente con la topología de la red. Utilizando URIs se obtiene mucha flexibilidad en la estructura de los EIDs. Como nombres, los EIDs no requieren una organización topológica para el enrutamiento. Esta organización no se prohíbe, y en algunos entornos podría ser ventajosa.

Un único EID puede referirse a un *endpoint* que contenga uno o más nodos DTN. Es la responsabilidad del diseñador definir cómo interpretar el SSP de un EID para determinar si se refiere a un conjunto de nodos multicast, anycast o unicast. Las URIs se construyen basándose en las reglas de la RFC 3986, usando los caracteres US-ASCII [5].

El grupo IBR-DTN ha desarrollado una serie de aplicaciones DTN [4] que pueden ser ilustrativas sobre la utilización de EIDs. Estas aplicaciones utilizan EIDs cómo el EID *dtn://chat.dtn/presence*, el *dtn://broadcast.dtn/streaming* y el dtn://cloud.dtnbone.dtn.

Late Binding

El *binding* consiste en la interpretación del SSP de un EID con el propósito de transportar el mensaje subyacente a un destino. Por ejemplo, *binding* podría requerir el mapeo de un EID al siguiente salto o a una capa inferior para la transmisión.

El *late binding* indica que el *binding* del destino del *bundle* no tiene porqué ocurrir en el origen de ese *bundle*. Como el EID de destino puede ser reinterpretado en cada salto, el *binding* puede ocurrir en el origen, en la transmisión o en el destino. Contrasta con el *binding* que se realiza en las comunicaciones de Internet (*name-to-adress*) donde una consulta DNS en el origen fija la dirección de destino antes del envío de los datos.

Este caso se conocería como *early binding* ya que se hace la traducción antes de enviar los datos por la red.

En una red que está desconectada frecuentemente, esta solución puede ser una ventaja ya que el tiempo de transmisión del mensaje puede exceder el tiempo de validez del *binding*, haciéndolo en el origen imposible o inválido. Utilizando la técnica de *late binding* podemos reducir la cantidad de información administrativa a propagar por la red. También puede limitar los requisitos de sincronización del mapeo a una topología local donde se pueden realizar cambios.

2.2.2.4. Anycast y Multicast

Un EID puede referirse a un *endpoint* que contiene uno o más nodos DTN. Cuando nos referimos a un grupo de más de uno, se denomina *anycast* o *multicast* (*broadcast* como tipo de *multicast*). Para un grupo de entrega *anycast*, el *bundle* se entrega a uno de los nodos del grupo, mientras que si el grupo de entrega es *multicast* debe entregarse a todo ese grupo. Para los grupos *multicast* deben tenerse en cuenta algunas particularidades para DTN. En redes con "poco" retardo, como internet, los nodos se consideran parte de un grupo si han mostrado interés en unirse a él recientemente. En DTN, sin embargo, los nodos pueden querer recibir datos enviados a un grupo durante un intervalo de tiempo anterior al que ellos son capaces de recibirlo. Concretamente, una aplicación expresa el deseo de recibir datos enviados a un EID 'e' en el momento 't'. Antes de esto, durante el intervalo [t0, t1] con t > t1, puede que se hayan generado datos para el grupo 'e'. Para que una aplicación pueda recibir estos datos, deben estar disponibles durante un tiempo suficientemente largo después de que se hayan terminado de enviar al grupo. Por lo tanto, puede ser necesario almacenar los datos en la red para soportar esta funcionalidad.

2.2.2.5. Clases de Prioridad

La arquitectura DTN ofrece medidas (relativas) de prioridad (baja, media, alta) para la entrega de ADUs. Estas prioridades diferencian el tráfico basado en los deseos de la aplicación para la urgencia de la entrega de los ADUs, y son entregados en *bundle blocks* generados por la *bundle layer* basados en información específica de la aplicación.

Se han definido tres clases de prioridad relativas. Estas clases típicamente conllevan algún tipo de prioridad en la planificación entre los *bundles* de la cola de envío:

- ***Bulk***: Los *bulk bundles* se envían con política de *best effort*. Estos *bundles* no se enviarán si hay algún *bundle* de las otras clases hacia el mismo destino y desde el mismo origen.
- **Normal**. Estos se enviarán con mayor prioridad a los *bulk* pero con menor a los *expedited*.
- ***Expedited***: Estos *bundles* se transmiten con la mayor prioridad entre todos los *bundles*.

Las aplicaciones especifican la clase de prioridad y el tiempo de vida para cada ADU enviada. Esta información, junto con la política aplicada en los nodos DTN que selecciona cómo se enrutan y transmiten los mensajes, afecta a la probabilidad y al

retardo de la entrega de un ADU. La clase de prioridad de un *bundle* solo se requiere para relacionar *bundles* del mismo origen. Esto significa que un *bundle* de alta prioridad de una fuente podría no ser entregado antes (o con una QoS superior) que un *bundle* de prioridad media de otro origen. En función de la política de un nodo DTN de planificación o envío, la prioridad puede ser modificada durante la ruta. En algunos nodos DTN los *expedited bundles* pueden tener menor prioridad que *bulk bundles* (debido al origen). Existen muchas variaciones.

2.2.2.6. Opciones de Entrega y Administrative Records

La arquitectura DTN soporta muchas opciones de entrega que pueden ser escogidas por la aplicación cuando se requiere la transmisión de un ADU. Además, la arquitectura define dos tipos de *administrative records*: *status reports* y señales. Estos *records* son *bundles* que proporcionan información sobre la entrega de otros *bundles* y se utilizan en conjunción con las opciones de entrega.

Opciones de Entrega

Se definen ocho opciones de entrega. Las aplicaciones que envían un ADU pueden requerir una combinación de ellas que se transportan con cada uno de los *bundles* generados por la capa *bundle*, generando el ADU que requirió la aplicación:

1. ***Custody Transfer Requested***: Los *bundles* generados se entregan con un enfoque en la fiabilidad usando métodos de *custody transfer*. Los *bundles* se transmiten por la capa usando protocolos de transferencia fiables (si están disponibles). Esta responsabilidad de la entrega fiable de un *bundle* al destino puede ser transmitida a los siguientes elementos de la red por los que viaja el *bundle*.
2. ***Source Node Custody Acceptance Requiered***: Requiere que el nodo DTN origen proporcione *custody transfer* a los *bundles* generados. Si no está disponible en el origen, cuando se requiere esta opción de entrega, la transmisión falla. Algunas aplicaciones lo utilizan para insistir al nodo DTN origen para que se preocupe de los *bundles* generados (por ejemplo, usando almacenamiento persistente).
3. ***Report When Bundle Delivered***: Requiere un único *Bundle Delivery Status Record* generado cuando el ADU se entrega al destino. También se conoce como *return receipt*.
4. ***Report When Bundle Acknowledged by Application***: Requiere que un *Acknowledgement Status Report* sea generado cuando el ADU es recibido por la aplicación destino. Esto sólo ocurre por acción de la aplicación destino, es diferente de *Bundle Delivery Status Report*. Se utiliza en casos donde la aplicación pueda actuar como una *gateway*, y desee indicar el estatus de una operación de protocolo externa a DTN a la aplicación origen.
5. ***Report When Bundle Received***: Requiere que se genere un *Bundle Reception Status Report* cuando cada *bundle* enviado llega a un nodo DTN. Diseñado para el diagnóstico.
6. ***Report When Bundle Custody Accepted***: Requiere generar un *Custody Acceptance Status Report* cuando cada *bundle* enviado es aceptado usando *custody transfer*. Diseñado también como herramienta diagnóstica.

7. ***Report When Bundle Forwarded***: Requiere la generación de un *Bundle Forwarding Status Report* cuando cada *bundle* enviado sale de un nodo DTN. Diseñado como herramienta diagnóstica.
8. ***Report When Bundle Deleted***: Requiere que se genere un *Bundle Deletion Status Report* cuando se elimina un *bundle* (tras haber sido enviado) de un nodo DTN. Diseñado como herramienta diagnóstica.

Las cuatro primeras opciones se crearon para el uso común de las aplicaciones, mientras que las cuatro restantes son para un uso diagnóstico y suelen estar limitadas a entornos sujetos a congestión o a ataques.

Si las medidas de seguridad definidas en la RFC están habilitadas, hay tres opciones de entrega adicionales:

1. ***Confidentiality Requiered***: Se requiere que el ADU sea secreto para cualquier parte de la red excepto para el origen y los miembros del EID destino.
2. ***Authentication Requiered***: Se requiere que todos los campos que no varíen en el camino del *bundle* se hagan fuertemente verificables, es decir, criptográficamente fuertes. Esto protege una gran cantidad de campos, incluyendo el EID origen y destino, así como los datos del *bundle*.
3. ***Error Detection Requiered***: Requiere que las modificaciones a los campos que no varían en el camino de cada *bundle* para cada *bundle* sean detectables con alta probabilidad en el destino.

Administrative Records: Bundle Status Reports y Custody Signals

Los *administrative records* se utilizan para reportar información de estado o errores relacionados con la capa *bundle*. Se definen dos tipos: *Bundle Status Reports* (BSRs) y *Custody Signals*. Los *administrative records* se asemejan a los mensajes del protocolo ICMP en IP. Sin embargo, en ICMP los mensajes son devueltos al origen. En DTN, se redireccionan al *report-to* EID, en el caso de los BSRs, y al EID del nodo que custodia el *bundle* para las *custody signals*, que puede ser diferente del EID origen. Los *administrative records* se envían como *bundles* con EID de origen asignado a uno de los EIDs asociados con el nodo DTN que genera el *administrative record*. En algunos casos, la llegada de un único *bundle*, o un fragmento puede generar múltiples *administrative records*.

Los siguientes BSRs están definidos:

- ***Bundle Reception***: Enviado cuando un *bundle* llega a un nodo DTN. La generación de este mensaje puede estar limitada por la política local.
- ***Custody Acceptance***. Enviado cuando un nodo acepta la custodia de un *bundle* con la opción *Custody Transfer Requested* activada. La generación de este mensaje puede estar limitada por la política local.
- ***Bundle Deletion***. Enviado desde un nodo DTN cuando un *bundle* que contiene la opción *Report When Bundle Delete* se descarta. Esto puede ocurrir por varios motivos, como la expiración. La generación de este mensaje puede estar limitada por la política local pero se requiere en casos en los que la eliminación se realiza por el nodo que está custodiando actualmente el *bundle*.

- ***Bundle Delivery***: Enviado desde el nodo destino cuando un ADU completo tiene activada la opción *Report When Bundle Delivered*.
- ***Acknowledged by application***: Enviado desde el nodo destino cuando un ADU completo tiene activada la opción *Application Acknowledgment* y ha sido procesado por la aplicación. Generalmente conlleva alguna acción específica del lado del receptor.

Además del *status report*, la *custody signal* se define para indicar el estado de un *custody transfer*. Estos se envían al *current-custodian* EID en un *bundle*:

- ***Custody Signal***: Indica que la custodia se ha transferido correctamente. Esta señal es un booleano y puede indicar acierto o fallo en el intento de transferencia.

Los *administrative records* deben referenciar un *bundle* recibido. Esto se consigue con un método de identificación única de *bundles* basado en un *timestamp* de transmisión y número de secuencia.

2.2.2.7. Campos principales de un Bundle

Los *bundles* transportados entre los nodos DTNs obedecen al protocolo *bundle*. El protocolo está diseñado con un bloque primario obligatorio, un *payload* opcional (que contiene los datos del ADU) así como una serie de extensiones opcionales. Los bloques pueden colocarse en cascada en una forma similar a las cabeceras de extensión de IPv6 como se indica en la Figura 1.

Se detallan a continuación los campos más importantes que están presentes en el bloque primario (presentes en cada *bundle*):

- ***Creation Timestamp***: Concatenación del tiempo de creación de los *bundles*, así como un número de secuencia que se incrementa de forma monótona garantizando un ADU único de origen desde una fuente. La creación del *timestamp* se basa en la aplicación *time-of-day* que requiera enviar un ADU (no cuando se envían por la red). Se asume que los nodos DTN tienen posibilidades de sincronización.
- ***Lifetime***: El *time-of-day* en el cual el mensaje deja de ser útil. Si un *bundle* está almacenado en la red (incluyendo el nodo DTN origen) cuando se alcanza el *lifespan*, puede ser descartado. Se expresa como un *offset* relativo al *Creation Time*.
- ***Class of Service Flag***. Indica opciones de entrega y clase de prioridad de un *bundle*.
- ***Source* EID**. EID del origen (primero que envía).
- ***Destination* EID**. EID del destino final.
- ***Report-To Endpoint* ID**. EID identificando dónde los *reports* deberían ser enviados. Esto puede o no identificar el mismo *endpoint* del nodo origen.
- ***Custodian* EID**. EID del *custodian* actual de un *bundle* (si existe).

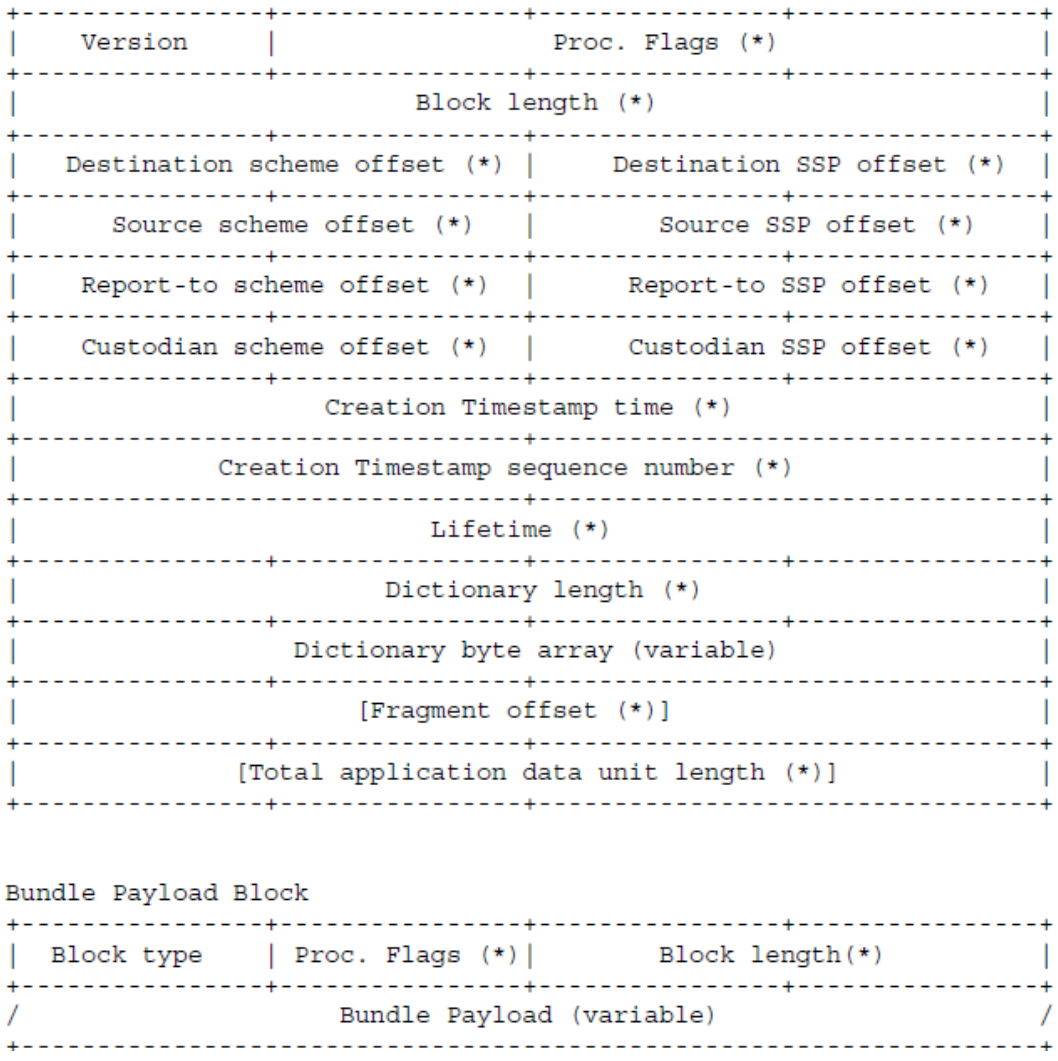

FIGURA 1: FORMATO DE UN BUNDLE [3]

El bloque de *payload*, además de la propia payload, indica información sobre ella como la longitud. Además de estos bloques, cada *bundle* especifica otros campos que están detallados en el protocolo *bundle* [3].

2.2.2.8. Routing and Forwarding (Enrutamiento y Envío)

La arquitectura DTN proporciona un marco para el enrutamiento y el envío a través de la capa *bundle* de mensajes *unicast*, *anycast* y *multicast*. Como los nodos DTN de una red pueden interconectarse usando más de un tipo de tecnología (por debajo de la capa *bundle*), una red DTN se puede describir de forma abstracta con un *multigraph* (un grafo donde los vértices pueden interconectarse con más de una arista). Las aristas del grafo son, generalmente, variantes en el tiempo respecto al retardo y la capacidad, direccionales (por la posibilidad de la conectividad en un sentido).

Cuando una arista tiene capacidad cero, se considera no conectada. Como las aristas en el grafo DTN pueden tener un retardo significativo, es importante distinguir dónde se mide el tiempo al expresar la capacidad de una arista o su retardo. Se adopta como convención expresarlos como funciones temporales donde el tiempo se mide desde el punto en que los datos se insertan a una arista de la red. Por ejemplo, consideramos

una arista con capacidad $C(t)$ y retardo $D(t)$ en un instante t. Si B bits empiezan a llegar a la arista en el instante t, acabarán de llegar en el instante $t + D(t) + (1/C(t))*B$. Se asume que $C(t)$ y $D(t)$ no cambian significativamente en ese intervalo.

Como las aristas pueden tener capacidad positiva o cero, es posible describir un intervalo durante el cual la capacidad es estrictamente positiva, y el retardo y la capacidad se considerarán constantes. Este periodo se denomina "contacto". Además, el producto de la capacidad por el contacto se conoce como "volumen". Si los contactos (y sus volúmenes) se conocen de antemano, se puede realizar enrutamiento inteligente y decisiones de encaminamiento (óptimamente para redes pequeñas). Utilizar el volumen de un contacto de forma óptima requiere la capacidad de dividir grandes ADUs y *bundles* en unidades más pequeñas enrutables. Esto se proporciona por la fragmentación DTN.

Cuando los caminos de entrega a través de un grafo DTN tienen pérdidas o los intervalos de contacto y volúmenes no se conocen de antemano, el enrutamiento computacional puede ser un verdadero reto.

Tipos de *Contacts*

Se pueden dividir los contactos en una serie de categorías, basadas en la capacidad de predicción de su rendimiento y si se requiere alguna acción para crearlos:

- ***Persistent Contacts***: Siempre están disponibles (no se requiere *connection initiation* para instanciar estos contactos). Una conexión permanente a internet como DSL podría ser un ejemplo representativo de esta clase.
- ***On-Demand Contacts***: Requieren alguna acción para instanciarlos pero a partir de ahí son equivalentes a los persistentes hasta que finalizan. Una conexión de tipo "*dial-up*" es un ejemplo representativo (desde el punto de vista de quien inicia la conexión).
- ***Intermittent – Scheduled Contacts***: Se establece un contacto acordado en un instante determinado para una duración específica. Un enlace entre un satélite orbitando cerca de la tierra podría ser un ejemplo. La lista de contactos de un nodo con el satélite puede ser realizada en función del calendario del satélite, su capacidad y su latencia. Para redes con retardo sustancial el instante correcto puede ser dependiente al retardo. Por ejemplo, un único contacto planificado entre la Tierra y Marte no sería en el mismo instante en cada localización, habría un offset que sería el retardo de propagación.
- ***Intermittent – Opportunistic Contacts***: No se planifican, se presentan de forma inesperada. Por ejemplo un avión que no debería pasar por una localización determinada establece una conexión con una torre. También una conexión *bluetooth* entre un dispositivo que se aproxima a un punto que está generando una oferta en unos grandes almacenes para sus clientes.
- ***Intermittent – Predicted Contacts***: Basados en una planificación variable, pero basándonos en la observación de contactos previos es probable que vuelvan a contactar en instantes determinados y con una duración predecible.

2.2.2.9. Fragmentación y Reensamblado

La fragmentación y el reensamblado de DTN se utilizan para mejorar la eficiencia de la transferencia de los *bundles*, asegurando que el volumen del contacto se utiliza totalmente evitando retransmisiones o un envío parcial de *bundles*. Hay dos formas de fragmentación y reensamblado en DTN:

- **Fragmentación Proactiva**: Un nodo DTN puede dividir un bloque de datos de la aplicación en múltiples boques más pequeños y transmitir cada bloque como un *bundle* independiente. En este caso, el destino final es responsable de la extracción de los bloques más pequeños de los *bundles* entrantes y reensamblarlos en el *bundle* completo original, generando al final el ADU. Esta aproximación se llama fragmentación proactiva porque se usa de forma primaria cuando se conoce el volumen del contacto de antemano (o se puede predecir).
- **Fragmentación Reactiva**: Los nodos DTN que comparten una arista en el grafo DTN pueden fragmentar un *bundle* cooperativamente cuando un *bundle* sólo se transfiere parcialmente. En este caso, la capa *bundle* receptora modifica el *bundle* entrante para indicar que es un fragmento y enviarlo normalmente. El *previous hop sender* puede saber (por protocolos de convergencia de capas) que sólo una porción del *bundle* se entregó al siguiente salto y enviar las partes restantes cuando los contactos estén disponibles. Esto se conoce como fragmentación reactiva ya que el proceso de fragmentación sucede tras el intento de transmisión. Por ejemplo, consideremos una estación base G, y dos satélites *store-and-forward* S1 y S2 en baja órbita sobre lados opuestos de la tierra. Mientras G transmite un gran *bundle* a S1, el protocolo de transporte fiable que hay debajo de la capa *bundle* en cada uno indica que la transmisión ha terminado, pero media transferencia se ha completado satisfactoriamente. G puede hacer un fragmento del *bundle* más pequeño con la segunda parte del *bundle* original y enviarlo a S2 cuando esté disponible. Además, S1 (que se encuentra fuera de rango de G), podría formar un nuevo *bundle* con la primera mitad y mandarlo al siguiente salto si es oportuno.

La fragmentación reactiva no es requerida en todas las implementaciones DTN ya que necesita un nivel de soporte de los protocolos inferiores que puede no existir, así como presentar dificultades en temas de seguridad (firmas digitales y autenticación). Cuando un mensaje firmado digitalmente se recibe parcialmente provocará que la mayoría de formatos de autenticación fallen.

Si la seguridad en DTN está activada es posible que sea necesario utilizar fragmentación proactiva para crear *bundles* de un tamaño más conveniente para las firmas digitales. Aunque la fragmentación reactiva no esté implementada, la capacidad de reensamblar los fragmentos en el destino es requerida para considerar que un nodo soporta fragmentación DTN. Además, para contactos con volumen pequeño comparado con el tamaño típico de *bundle*, podría usarse una aproximación de entrega incremental para prevenir *data delivery livelock*. La fragmentación reactiva es una posible solución pero otras capas de transporte de protocolo podrían manejar este asunto también.

2.2.2.10. Fiabilidad y Custody Transfer

El servicio más básico que proporciona la capa *bundle* es la entrega de un mensaje *unicast* (sin recibir confirmación) priorizado (pero no garantizado). También proporciona dos opciones para aumentar la fiabilidad de la entrega: *end-to-end ACKs* y *custody transfer*. Las aplicaciones que deseen implementar sus propios mecanismos de fiabilidad en los mensajes *end-to-end* pueden usar el ACK. La *custody transfer* en DTN sólo especifica una posibilidad de retransmisión.

La transmisión de *bundles* con la opción de *custody transfer* habilitada supone el movimiento de la responsabilidad de la transmisión fiable de los *bundles* de un ADU entre los diferentes nodos DTN de la red. Para la entrega *unicast*, esto supone mover los *bundles* por la ruta más corta (en medidas de métrica) al destino final, retransmitiendo cuando sea necesario. Los nodos que reciben los *bundles* por el camino (que aceptan la responsabilidad de la entrega fiable) se llaman *custodians*. El movimiento de un *bundle* (y la responsabilidad de su entrega) de un nodo a otro se llama *custody transfer*. Es análogo a la transacción del *commit* de una base de datos.

Custody transfer permite a la fuente delegar la responsabilidad de la transmisión y recuperar los recursos relacionados con la retransmisión de forma rápida tras enviar el primer *bundle*. No todos los nodos DTN aceptan *custody transfer* por lo que no es un mecanismo real salto a salto. Algunos nodos pueden tener un almacenamiento suficiente para hacer custodia pero pueden decidir no ofrecer ese servicio si tienen congestión o no tienen la energía suficiente.

La existencia de nodos que hacen custodia puede modificar el enrutamiento DTN. En ocasiones puede ser beneficioso mover un *bundle* a un *custodian* lo más rápido posible aunque el *custodian* esté lejos (en términos de distancia, tiempo o métrica) del destino final del *bundle* o de otro nodo al que tenga que ir.

Diseñar el sistema con esta funcionalidad conlleva la construcción de más de un grafo de enrutamiento. *Custody transfer* en DTN no sólo proporciona un método de seguimiento de los *bundles* que requiere un manejo especial e identificar los nodos DTN que participan en la *custody transfer*, sino que también proporciona un mecanismo (débil) de aumentar la fiabilidad de la entrega de los mensajes.

De forma general, *custody transfer* confía en la capa *bundle* fiable de los protocolos de transporte de las redes en las que opera para proporcionar transferencia fiable de un *bundle* entre un nodo y el siguiente. Sin embargo, cuando se necesita la *custody transfer* la capa bundle proporciona un *timeout* adicional y mecanismos de retransmisión y un ACK asociado (de la capa *bundle*) entre los *custodians*. Cuando una aplicación no necesita *custody transfer* este *timeout* y el mecanismo de retransmisión asociado no se implementan, y la entrega correcta se fía únicamente en los protocolos subyacentes.

Cuando un nodo acepta la custodia de un *bundle* que contiene la opción *Custody Transfer Requested* activada, una señal de *Custody Transfer Accepted* se envía por la capa *bundle* al *Custodian EID* actual que se encuentra en el *bundle block* primario.

Además, la *Custodian EID* actual se actualiza para que contenga uno de los EIDs de los nodos de envío (siguiente salto, *unicast*) antes de que se envíe al siguiente salto.

Cuando la aplicación requiere que se entregue un ADU con *custody transfer*, la petición es recomendable aunque en algunas circunstancias el origen del *bundle* por el que se requería *custody transfer* puede no proporcionar el servicio. También el *bundle* podría atravesar múltiples nodos DTN hasta que consigue un *custodian*. Los *bundles* en esta situación se marcan con su campo *Current Custodian EID* a "*null endpoint*". Cuando las aplicaciones desean que el origen sea el *custodian* de un *bundle*, podrían proporcionar la opción de entrega *Source Node Custody Acceptance Requiered*. Esto podría ser útil para las aplicaciones que quieren una cadena continua de custodia o que deseen finalizar antes de asegurarse de que sus datos están de forma segura en un *custodian*.

En una red DTN donde uno o más saltos entre *custodians* son estrictamente de una dirección (no pueden invertirse), el mecanismo DTN de *custody transfer* se verá afectado en esos saltos debido a la falta de capacidad para recibir una *custody signal* u otra información de vuelta por el camino, por lo que el *bundle* expirará en la entrada del camino de un único sentido. Esto no significa que el *bundle* se pierda necesariamente, los nodos del otro lado del salto podrían continuar transfiriendo la custodia, y el *bundle* se podría entregar correctamente en el destino. Sin embargo, en esta circunstancia el origen que requirió recibir BSRs de expiración para este *bundle* recibirá un reporte de expiración para el *bundle* y posiblemente determinará (incorrectamente) que el *bundle* se ha descartado y no entregado. Aunque este problema no se puede solucionar completamente en esta situación, un mecanismo proporcionado para ayudar a mejorar el conocimiento de la información incorrecta que pueda reportarse cuando el *bundle* expire tras haberse transferido en un camino de un único sentido. Esto se consigue con un nodo al comienzo del camino de un único sentido que avise de la existencia de un camino conocido de un único sentido usando una variante del *status report* de un *bundle*. Estos tipos de reportes se proporcionan si el *bundle* los requiere utilizando la opción *Report When Bundle Forwarded*.

2.2.2.11. Soporte para *proxies* y *gateways* de la capa de aplicación

Una de las aspiraciones de DTN es proporcionar un método común para interconectar las *gateways* y los *proxies* de la capa de aplicación. En los casos donde existan aplicaciones de Internet que puedan tolerar retardos, se pueden construir *proxies* locales para beneficiarse de las capacidades de comunicación proporcionadas por DTN. Haciendo los *proxies* compatibles con DTN se reduce la carga en el autor del *proxy* de preocupación sobre cómo implementar el enrutamiento y el control de la fiabilidad, así como permitir a las aplicaciones existentes TCP/IP a operar sin modificación sobre una red basada en DTN.

Cuando se utiliza DTN para proporcionar la encapsulación de un túnel para otros protocolos utilizándose para construir redes formadas por *gateways* de la capa de aplicación. La capacidad de hacer ACKs de la aplicación se diseñó para estos casos. Esto proporciona una forma común para que *gateways* remotas de la capa de aplicación puedan señalizar el éxito o el fallo de operaciones que no sean del protocolo DTN,

iniciadas debidas a los ADUs DTN recibidos. Sin esta habilidad, los indicadores tendrían que ser implementados por las aplicaciones por si mismas sin formas estándar.

2.2.2.12. Timestamps y Sincronización Temporal

La arquitectura DTN depende del tiempo de sincronización entre los nodos DTN (soportados por protocolos que no son DTN, externos) por cuatro propósitos primarios: identificación de *bundle* y fragmentos; enrutamiento con contactos predichos o planeados; expiración del tiempo computacional de los *bundles* y expiración del registro de la aplicación.

La identificación de los *bundles* y la expiración se soportan incluyendo un *timestamp* de creación y un campo explícito de expiración (expresado en segundos tras el *timestamp* origen) en cada *bundle*. Los *timestamps* de los *bundles* que van llegando están disponibles para las aplicaciones que los consumen en ADUs recibidos por alguna función de la interfaz del sistema. Cada conjunto de *bundles* correspondiente a un ADU es necesario que contenga un *timestamp* único para el EID de envío. El EID, *timestamp* y offset de datos / longitud de información, identifican de forma única a un *bundle*. La identificación única del *bundle* se utiliza para múltiples propósitos, incluyendo *custody transfer* y reensamblado de los fragmentos de un *bundle*. Cuando una aplicación expresa su deseo de recibir un ADU destinado a un EID particular, el registro solamente se mantiene por un periodo de tiempo finito, y puede ser especificado por la aplicación. Para un registro *multicast*, la aplicación puede especificar un rango temporal o un intervalo de interés para su registro. En este caso, el tráfico enviado al EID durante el intervalo especificado, será entregado en su momento a la aplicación (a no ser que el tráfico expire debido al *expiration time* proporcionado por la aplicación en la fuente o cualquier otro motivo que facilite la entrega).

2.2.2.13. Congestión y Control de Flujo en la capa *Bundle*

Aún no existe un consenso amplio en este aspecto. Existen reticencias sobre la eficiencia y la eficacia de los mecanismos de control de congestión y de flujo sobre estos entornos con un retardo variable (y generalmente alto) especialmente usando el mecanismo de *custody transfer* que puede requerir que los nodos retengan *bundles* durante un largo periodo de tiempo.

En DTN se define control del flujo como los medios que aseguren que la tasa media a la que transmite el nodo origen a un nodo receptor no exceda de la tasa media a la que el nodo receptor está preparado para recibir datos de ese origen.

Se define como control de congestión en DTN a los medios que aseguran que la tasa agregada a la que todas las fuentes de tráfico inyectan datos a la red no excede la tasa máxima agregada a la que la red puede entregar los datos a los nodos destino en el tiempo. Si el control de flujo se propaga hacia atrás desde nodos congestionados a las fuentes del tráfico, entonces el mecanismo de control de flujo se puede utilizar como una solución parcial del problema de congestión en sí mismo.

Las decisiones de control de flujo en DTN deben hacerse en la capa de *bundle*, basándose en la información de los recursos (almacenamiento persistente primario) disponibles en el nodo donde está el *bundle*. Cuando los recursos de almacenamiento se

vuelven escasos, el nodo DTN sólo tiene un cierto grado de libertad para controlar la situación. Siempre puede descartar los *bundles* que han expirado (esto debería hacerse regularmente por los nodos en cualquier caso). Si el nodo aceptaba custodia de *bundles* también puede dejar de hacerlo. Si hay recursos de almacenamiento disponibles en otro lugar de la red también podría usarlos para almacenar los *bundles*. También puede descartar los *bundles* que no han expirado, pero para los que no ha aceptado la custodia. El nodo debe evitar descartar los *bundles* para los que aceptó custodia, haciendo esto únicamente como último recurso.

Además de estos mecanismos, un nodo DTN debe ser capaz de utilizar la información de los protocolos subyacentes que afecte a la utilización de los recursos. Por ejemplo, un nodo DTN que reciba un *bundle* DTN usando TCP/IP podría frenar intencionalmente la tasa de recepción haciendo las operaciones de lectura de forma menos frecuentes para reducir la carga. Esto es posible con el propio mecanismo de control de flujo de TCP.

2.2.2.14. Security

La posibilidad de la escasez de recursos en algunas redes tolerantes al retardo dictamina que debe realizarse algún mecanismo de autenticación y de control de acceso a la propia red en algunas circunstancias. No es aceptable que un usuario no autorizado inunde la red con tráfico de forma fácil, pudiendo denegar el servicio a los usuarios autorizados. En muchos casos tampoco es aceptable que tráfico no autorizado se envíe a algunos puntos de la red (como enlaces críticos para la red). Tomando estas consideraciones se establecen algunos objetivos para la componente de seguridad de la arquitectura DTN:

- Prevenir inmediatamente que las aplicaciones no autorizadas transporten o almacenen sus datos en la red DTN.
- Prevenir que aplicaciones no autorizadas tomen el control de la infraestructura DTN.
- Prevenir que cualquier aplicación envíe *bundles* a una tasa o clase de servicio para la que no tienen autorización.
- Descartar inmediatamente los *bundles* que hayan sido dañados o modificados indebidamente en tránsito.
- Detectar y desautorizar las entidades comprometidas.

Muchos de los protocolos existentes de autenticación y control de acceso diseñados para operar en redes con bajo retardo, pueden no funcionar bien en DTN. En particular, actualizar listas de control de acceso y revocar credenciales puede ser especialmente difícil. Además, las soluciones que requieren un acceso frecuente a algún servidor centralizado para completar una transacción de autenticación o autorización no son útiles para DTN. Como consecuencia de estas dificultades se incluyen retardos en la comunicación, retardos en detectar y recuperar un sistema comprometido, así como retardos en completar transacciones debido a accesos inapropiados.

Para ayudar a satisfacer estos requisitos de seguridad la arquitectura DTN adopta un estándar [19] que se implementará opcionalmente en la arquitectura de seguridad que utiliza autenticación salto a salto y *end-to-end* y mecanismos de integridad. El

propósito de uso de las dos aproximaciones es poder controlar el acceso a los datos que se envían y al almacenamiento de forma separada a la integridad de los datos de aplicación. Mientras que los mecanismos *end-to-end* proporcionan autenticación para el usuario (podrían ser muchos usuarios), los mecanismos salto a salto sirven para autenticar nodos DTN como legítimos transceptores de los *bundles* entre ellos. Es posible que únicamente una subred de los nodos forme parte de los mecanismos de seguridad.

En concordancia con los objetivos listados previamente, los nodos DTN pueden descartar tráfico tan pronto como un chequeo de autenticación o de control de acceso falla. Esto consigue eliminar cualquier tráfico no deseado y evitar el envío a algún punto sensible de la red pero también lleva el beneficio asociado de evitar los ataques de denegación de servicio, siendo más difícil de llevar a cabo que en los *routers* de internet; sin embargo, el coste computacional de estos mecanismos es muy grande.

2.2.3. Control de Estado

Un aspecto importante de cualquier arquitectura de red es el control de su estado. Este apartado describe cómo se realiza el control de estado de la capa *bundle*.

2.2.3.1. Estado de Registro de la Aplicación

En entornos con un gran retardo (o retardo variable) una interfaz de aplicación asíncrona es la más apropiada. Estas interfaces típicamente incluyen métodos para que las aplicaciones registren hechos cuando ocurran algunos eventos (por ejemplo cuando llegan los ADUs). Estos registros crean información de estado, llamada registro de estado de la aplicación.

Los registros de estado de la aplicación se crean típicamente por petición explícita de la aplicación y se eliminan cuando se especifica de forma explícita, aunque pueden ser eliminados por algún *timer* indicado por la aplicación. En la mayoría de los casos debe haber una provisión para almacenar las condiciones de fin o reinicio de la aplicación y del sistema operativo. En los casos dónde las aplicaciones no han reiniciado automáticamente la aplicación pero el registro de estado de la aplicación es persistente, se puede proporcionar algún método para indicar al sistema qué hacer cuando se lance algún evento (reinicio de aplicación, ignorar un evento…).

Para iniciar el registro y establecer el estado de registro de la aplicación, la aplicación especifica un *Endpoit ID* para el que desea recibir ADUs, junto con valores temporales opcionales indicando cuanto tiempo debería permanecer el registro activo. Esta operación es análoga a la operación de `bind()` en el API de los sockets comunes. Para registrar grupos se puede especificar un intervalo de tiempo. Este intervalo indica el rango de creación de tiempos de ADUs enviados a un EID específico.

2.2.3.2. Estado de Custody Transfer

El estado de una *custody transfer* incluye información requerida para conocer qué *bundles* han sido custodiados en un nodo así como el estado del protocolo relacionado con la transferencia de custodia en uno o más. El estado relacionado con la cuenta se crea cuando se recibe un *bundle*. Un estado de retransmisión de *custody transfer* se crea cuando se inicia la transferencia de custodia por el envío de un *bundle* con la opción

de *custody transfer requested delivery* activada. El estado de retransmisión y el estado de *accounting* pueden liberarse cuando se reciben señales de *Custody Transfer Succeeded,* indicando que la custodia se ha movido. Además, el tiempo de expiración del *bundle* (posiblemente mitigado por política local) proporciona un vínculo superior en el momento en el que el estado se purga del sistema en el evento que no se ha purgado explícitamente, debido a la recepción de una señal.

2.2.3.3. Estado de *Routing* y *Forwarding* de un bundle

Como en la arquitectura de internet, se distingue entre enrutamiento y envío (*routing and forwarding*). El enrutamiento se refiere a la ejecución de un algoritmo (posiblemente distribuido) para encontrar un camino de enrutamiento acorde a alguna función objetiva. *Forwarding* (envío) se refiere al acto de mover un *bundle* de un nodo DTN a otro. El enrutamiento utiliza un estado de enrutamiento (el RIB o la información base de enrutamiento), mientras que el envío hace uso de un estado derivado del enrutamiento, y se mantiene como estado de envío (el FIB o base de envío de información). La estructura de la FIB y las reglas de mantenimiento son elecciones de la implementación. En algunas DTNs, el intercambio de la información utilizado para actualizar el estado de un RIB puede ocurrir por caminos de la red diferentes de los que se intercambia la información.

El mantenimiento del estado de un RIB depende del tipo de algoritmo de enrutamiento utilizado. Un algoritmo de enrutamiento puede considerar una clase de servicio específica y la localización de los *custodians* potenciales, y esta información se incrementará el tamaño del RIB.

Los *bundles* pueden ocupar colas en un nodo durante un tiempo considerable de tiempo. Para entregas *unicast* o *anycast*, la cantidad de tiempo es probable que sea un intervalo entre el momento en que llega un *bundle* al nodo y cuando se envía al siguiente salto. Para entrega *multicast* esto podría ser mayor, hasta el tiempo de expiración del *bundle*. Esta situación ocurre cuando la entrega *multicast* se utiliza como una forma de que los nodos que se unen a un grupo obtengan información previa enviada al grupo. En estos casos algunos nodos pueden actuar como “archivadores” que proporcionan copias de los *bundles* a nuevos participantes que se han entregado a otros.

2.2.3.4. Estado relacionado con la Seguridad

La seguridad de DTN descrita en la RFC, cuando se usa, requiere un mantenimiento de estado en todos los nodos DTN que la usan. Todos estos nodos se les requiere almacenar su propia información privada (incluyendo su propia política y material de autenticación) y un bloque de información usado para verificar credenciales. Además, en la mayoría de los casos, los nodos DTN cachearán información pública (posiblemente credenciales) de los vecinos. Toda la información cacheada tiene tiempo de expiración y los nodos son responsables de adquirir y distribuir actualizaciones de información pública y credenciales antes de que el tiempo de expiración acabe (para evitar una disrupción del servicio).

Además de la autenticación *end-to-end* y salto a salto, el control de acceso puede usarse en DTN por uno o más mecanismos como las capacidades o listas de control de

acceso (ACLs). Los ACLs representarían otro bloque de estado presente en un nodo que desea reforzar la política de seguridad. Los ACLs se inicializan típicamente en el tiempo de configuración del nodo y puede ser actualizado dinámicamente por *bundles* DTN o por alguna técnica fuera de banda. Las habilidades o credenciales pueden ser revocadas requiriendo el mantenimiento de la lista de revocación (*black list*) para comprobar material inválidamente autenticado que haya sido distribuido.

Algunas DTNs pueden implementar barreras de seguridad seleccionando nodos en la red donde las credenciales *end-to-end* pueden ser comprobadas además del chequeo de las credenciales salto a salto. La información pública usada para verificar la autenticación DTN será típicamente cacheada en estos puntos.

2.2.3.5. Política y Estado de Configuración

Los nodos DTN contendrán una cantidad de información de políticas y de configuración. Esta información podría alterar el comportamiento del envío de los *bundles*. Por ejemplo, los tipos de algoritmos criptográficos y los procedimientos de control de acceso que usar si la seguridad de DTN está habilitada, si los nodos son *custodians*, qué tipos de capa de convergencia y protocolos de enrutamiento se usan, como los *bundles* con diferente prioridad deben planificarse, dónde y por cuánto tiempo deben almacenarse los *bundles*, qué reportes de estado se generan y a qué tasa, etc…

2.2.4. Estructura de las Aplicaciones

La entrega de los *bundles* DTN está preparada para operar en redes tolerantes al retardo sobre un gran rango de tipos de redes. Esto no significa que sea necesario que haya grandes retardos en la red, quiere decir que puede haber retardos muy significativos (incluyendo periodos de desconexión entre el emisor y los receptores). Los protocolos DTN son tolerantes al retardo, por lo que las aplicaciones que los usan también deben ser tolerantes al retardo para funcionar correctamente en entornos sujetos a un gran retardo o interrupciones.

Las primitivas de comunicación proporcionadas por la arquitectura DTN están basadas en comunicaciones asíncronas, orientadas a mensaje que difieren de la típica comunicación de petición y respuesta. Generalmente, las aplicaciones deberían intentar incluir información suficiente en un ADU para que pueda tratarse como una unidad independiente de trabajo por la red y los receptores. El objetivo es minimizar los intercambios síncronos entre las aplicaciones que están separadas por una red muy grande y con una gran posibilidad de retardos altamente variables. Un único mensaje de *file transfer request*, por ejemplo, podría incluir información de autenticación, información de localización del archivo, y las operaciones requeridas sobre él (uniendo todo en un *bundle*). Comparando este estilo de operaciones con la clásica transferencia por FTP se aprecia que el modelo de los *bundles* puede completarse en un RTT mientras que un *put* de un archivo FTP puede necesitar hasta 8 RTTs para comenzar el envío [20].

Las aplicaciones tolerantes al retardo deben considerar factores adicionales más allá de las implicaciones de caminos con largo retardo. Por ejemplo, una aplicación podría terminar (voluntariamente o no) entre el instante en que envía un mensaje y el instante en el que esperaría una respuesta. Si se anticipa esta posibilidad la aplicación

podría ser instanciada de nuevo, con información de estado almacenada en almacenamiento persistente. Este es aún un problema de la implementación, pero también una consideración de diseño de la aplicación. Algunas consideraciones de diseño de aplicaciones tolerantes al retardo pueden resultar en aplicaciones que funcionen razonablemente bien en entornos con bajo retardo y que no sufran extraordinariamente en entornos con alto o muy variable retardo.

2.2.5. Convergencia de Capas para el Uso de Protocolos

No todos los protocolos subyacentes en diferentes familias de protocolos proporcionan la misma funcionalidad exactamente, por lo que deben realizarse adaptaciones o una configuración por protocolo o por familia de protocolos. Esta adaptación se realiza con una serie de capas de convergencia entre la capa *bundle* y los protocolos subyacentes. Las capas de convergencia controlan los detalles específicos de protocolo para cada protocolo subyacente y presentan una interfaz consistente para la capa *bundle*.

La complejidad de una capa de convergencia puede variar sustancialmente de otras, depende del tipo de protocolo subyacente que adapta. Por ejemplo, una capa convergente para TCP/IP para usarse en internet podría únicamente tener que añadir los *bundles* a los *streams* TCPs mientras que una capa de convergencia para algunas redes donde no haya un protocolo de transporte fiable debe ser sustancialmente más complejo (para implementar fiabilidad, fragmentación, control de flujo…) si se requiere ofrecer una entrega fiable por la capa *bundle*.

2.2.6. Conclusiones y Aplicaciones DTN en Este Proyecto

Tras realizar un estudio pormenorizado del protocolo DTN y de la complejidad de sus diferentes elementos, cabe destacar que la utilización de DTN no se encuentra demasiado extendida en la actualidad. Sin embargo, la juventud del protocolo y las virtudes previamente mencionadas, hace pensar en la potencialidad de este protocolo den el futuro para comunicaciones en redes que sean inestables o de gran retardo.

El piloto de aplicación Android sobre DTN que se ha llevado a cabo para este proyecto, no puede abarcar la completitud y complejidad del protocolo completa. Este piloto utilizará el API Java desarrollada por el grupo IBR-DTN. El proyecto piloto será tratado en capítulos posteriores, pero es importante destacar algunos elementos DTN que serán utilizados en la aplicación piloto de Android. Estos elementos serán los EIDs y registros, la distribución multicast, enrutamiento y envío, fragmentación y reensamblado, fiabilidad y *custody transfer*.

2.3. Android

2.3.1. Introducción

Android es una pila de software en código abierto que incluye el sistema operativo, middleware, y aplicaciones clave además de una serie de APIs para desarrollar aplicaciones destinadas originalmente para dispositivos móviles [4].

El sistema operativo móvil, está basado en el kernel de *Linux*, desarrollado actualmente por Google. Android Inc. se fundó en 2003 en California para desarrollar dispositivos móviles más enfocados en las preferencias personales de los usuarios. La compañía fue adquirida por Google en 2005, lo que mostró las intenciones de Google en introducirse en el mercado de las comunicaciones móviles, desde el punto de vista del dispositivo [6].

Android es un sistema operativo versátil, que se utiliza en la actualidad principalmente en dispositivos móviles como *smartphones* o *tablets*, pero que se abre mercado en nuevos mercados como las televisiones inteligentes, los *wearables*, las consolas de videojuegos, las cámaras digitales, los ordenadores personales e incluso los automóviles.

Los dispositivos móviles de hoy en día son herramientas poderosas que incorporan cámaras, reproductores multimedia, sistemas GPS y pantallas táctiles. La evolución de la tecnología ha eliminado la visión del teléfono móvil como un dispositivo para realizar llamadas. El desarrollo de software para estos dispositivos se ha convertido en una función principal para personalizarlos según las necesidades del usuario.

Android tuvo mucha competencia inicial con el sistema operativo de los *smartphones* de Nokia (Symbian), más adelante con iOS de Apple, el sistema operativo de BlackBerry y Windows Phone. Observando los datos de cuota de mercado actual [17] se puede afirmar que el sistema operativo Android es el más utilizado del mundo en dispositivos móviles, con una cuota de mercado muy superior a su potencial competidor actual (iOS).

En la actualidad, el sistema operativo Android tiene como versión más actualizada la 5.1 (*Lollipop* API 22). Sin embargo, como se aprecia en la Figura 2, la cuota de mercado actual no justifica aún el desarrollo de aplicaciones en Android 5.0 aunque sí se muestra conveniente por su continuo crecimiento.

Version	Codename	API	Distribution
2.2	Froyo	8	0.3%
2.3.3 - 2.3.7	Gingerbread	10	5.6%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.1%
4.1.x	Jelly Bean	16	14.7%
4.2.x		17	17.5%
4.3		18	5.2%
4.4	KitKat	19	39.2%
5.0	Lollipop	21	11.6%
5.1		22	0.8%

Data collected during a 7-day period ending on June 1, 2015.
Any versions with less than 0.1% distribution are not shown.

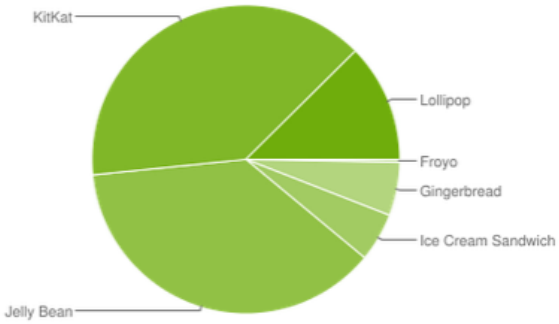

FIGURA 2: CUOTA DE MERCADO DE LAS DISTRIBUCIONES ANDROID [17]

El desarrollo de la aplicación que abarca este proyecto se realizará en Android 4 (versiones superiores a *Jelly Bean*). La cuota de mercado a junio de 2015 es del 76.6% agregando los valores desde la versión de API 16 a la 19, por lo que la versión a analizar y realizar el trabajo de este piloto se realizará enfocada en ella y no en versiones posteriores.

2.3.2. La Pila de Software de Android

La pila de *software* de Android está compuesta por un *kernel* de Linux y una colección de librerías de C/C++ [4]. La pila de Android se representa con más detalles en la Figura 3.

Los principales elementos de la pila de *software* de Android son:

- ***Kernel Linux***: Servicios del Core (librerías *hardware*, control de procesos y memoria, seguridad, red y control de batería) son proporcionados por el *kernel* 2.6 de *Linux*. También proporciona una abstracción entre el *hardware* y el resto de la pila.
- **Librerías**: Se encuentran encima del *kernel*. Android incluye librerías de C/C++ como libc, ssl; librerías multimedia; librerías para pantallas; librerías gráficas como SGL y OpenGL para gráficos 2D y 3D; librerías básicas de SQL; SSL y WebKit para el navegador web integrado y para seguridad en Internet.
- ***Run time Android***: El sistema en tiempo de ejecución es el elemento que controla principalmente el funcionamiento de Android (más que la implementación Linux). Además de las librerías del *Core* y la Máquina Virtual *Dalvik*, el run time es el motor que hace funcionar a las aplicaciones.
  - **Librerías del Core:** Estas librerías proporcionan las funcionalidades disponibles en las librerías Java, además de algunas específicas para Android.

- **Dalvik VM:** *Dalvik* es una Máquina Virtual optimizada para asegurar que un dispositivo puede ejecutar múltiples instancias simultáneamente de forma eficiente. Se basa en el *kernel Linux* para el manejo de hilos y de la memoria a bajo nivel. A partir de la versión de Android 5.0 se ha reemplazado completamente esta máquina virtual por ART (Android *Runtime*), el nuevo entorno de ejecución de aplicaciones Android [16].

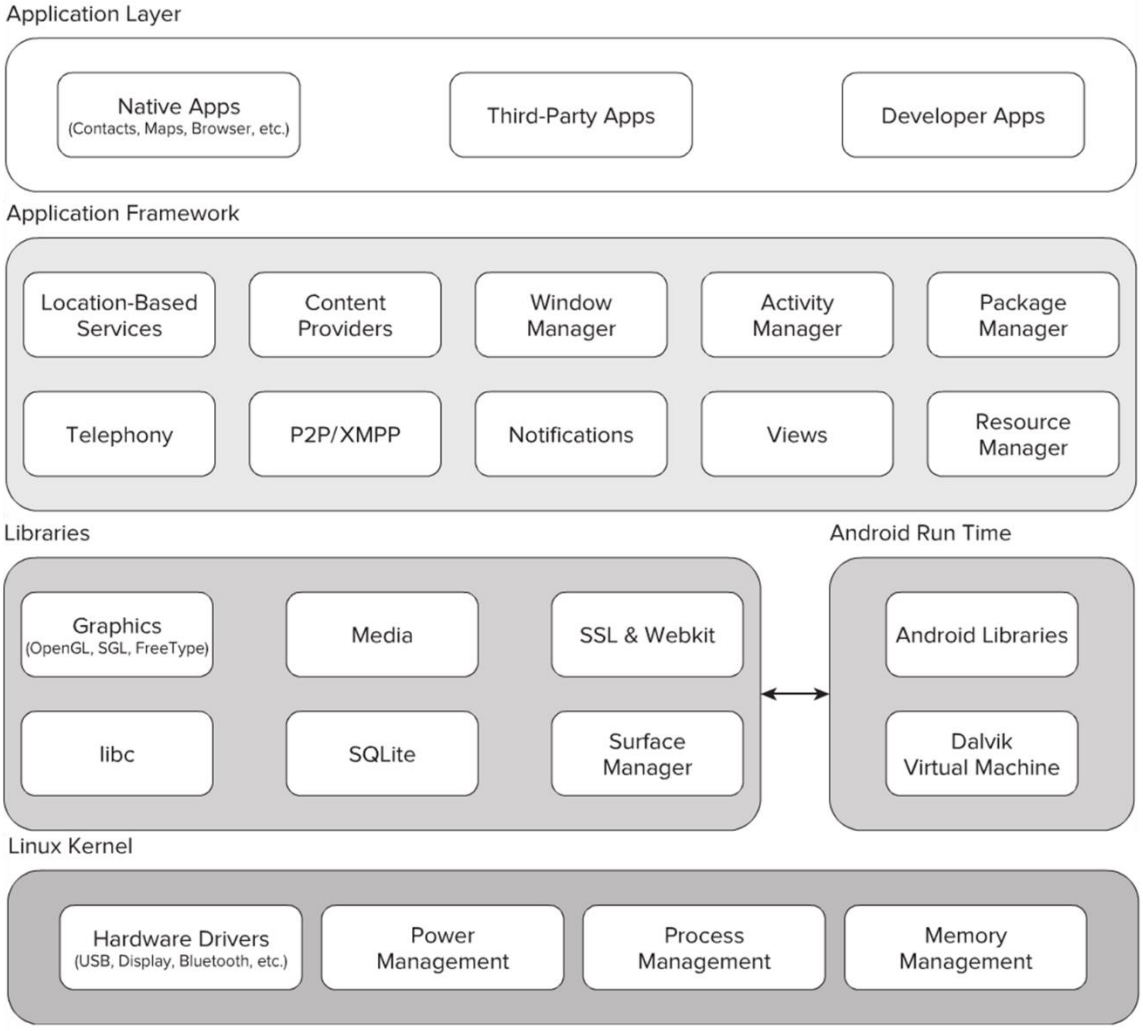

FIGURA 3: PILA ANDROID [4]

- **Framework de Aplicaciones**: Proporciona las clases utilizadas para la creación de aplicaciones Android. Proporciona una abstracción para el acceso al *hardware* y controla la interfaz de usuario y los recursos de aplicaciones.
- **Capa de Aplicación**: Todas las aplicaciones, tanto las nativas como las de terceros, están construidas en la capa de aplicación con los mismos APIs.

2.3.3. Tipos de Aplicaciones

Las aplicaciones Android pueden enmarcarse dentro de cuatro tipos diferentes.

Una aplicación tipo ***foreground*** es aquella que solamente es útil cuando aparece en pantalla y se suspende cuando no está visible. En este tipo de aplicaciones hay que ser especialmente cauteloso con el ciclo de vida de las *activities*, para que pasen de una forma "suave" de la pantalla principal al segundo plano. Las aplicaciones tienen poco control sobre sus ciclos de vida, y si no tienen servicios ejecutándose es posible que sean eliminadas por Android. Por este motivo es necesario almacenar el estado de las aplicaciones cuando van a pasar a segundo plano. Ej: Mapas, Juegos tradicionales.

Una aplicación con interacción limitada se dice que es una aplicación en ***background***. Pasa la mayoría de su tiempo de vida oculta, excepto cuando necesita ser configurada. Es posible crear servicios completamente invisibles pero en la práctica es mejor proporcionar algún tipo de control de usuario. Ej: La aplicación que nos muestra en pantalla las llamadas.

Las aplicaciones **intermitentes** son las que hacen la mayoría de su trabajo en segundo plano aunque está siempre esperando algún tipo de interactividad. Estás aplicaciones se ejecutarán de forma silenciosa pero notificarán a los usuarios cuando sea necesario. Estas aplicaciones suelen estar formadas por *activities* visibles y servicios en segundo plano invisibles. Es necesario que estén pendientes de su estado cuando interactúan con el usuario. Por ejemplo, no deberían notificar al usuario si éste está interactuando con su interfaz gráfica. Ej: Media Player.

Algunas aplicaciones se representan únicamente como un ***widget*** en la pantalla principal. Ej: *Widget* mostrando el tiempo atmosférico.

Algunas aplicaciones muy complejas son difíciles de encuadrar en una única categoría ya que incluyen elementos de varias. Cuando se crea una aplicación es necesario considerar cómo se usará más probablemente y diseñarla acorde a ello.

2.3.4. Elementos de la Aplicación Android

2.3.4.1. Componentes de una Aplicación Android

Activities

Es la presentación de la aplicación. Cada pantalla de la aplicación será una extensión de la clase *Activity*. Las *activities* usan vistas para crear interfaces gráficas de usuario que muestran información y responden a las acciones del usuario.

El estado de la *activity* ayuda a determinar la prioridad de la aplicación a la que pertenece. La prioridad de la aplicación influye en la probabilidad de que se termine la aplicación así como sus activities. El estado de cada activity viene determinado por la posición que ocupa en la pila de *activities*. Esta pila, representada en la Figura 4 es de tipo LIFO y se incluyen en ella todas las *activities* ejecutándose. Cuando arranca una nueva *activity* la que se encuentra en la pantalla principal se mueve a la parte superior de la pila. Si el usuario utiliza el botón Atrás o se cierra la aplicación de la pantalla principal, la siguiente *activity* en la pila pasa a la pantalla principal activándose.

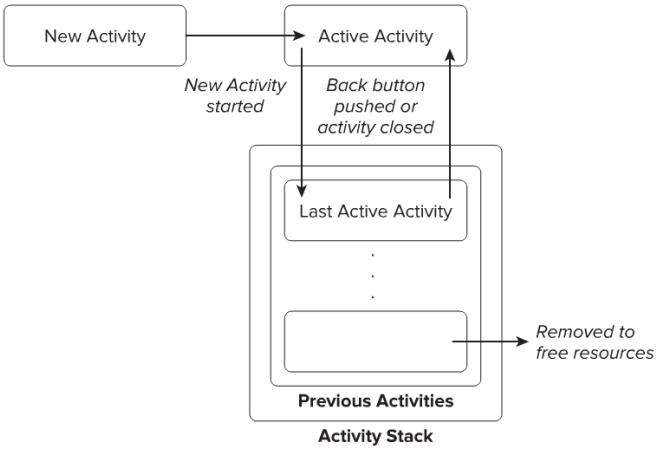

FIGURA 4: PILA DE LA ACTIVITY [4]

Los estados de una *activity* son los siguientes y se representan en ciclo de vida de una *activity* en la Figura 5:

- **Activa**. Cuando está en la cima de la pila. Es la que se encuentra visible e interactuando con el usuario. Android intentará que siga viva a toda costa, terminando activities inferiores en la pila si es necesario, para que esta tenga los recursos que necesita. Cuando otra activity se activa, ésta se pausa.
- **Pausada**. Es una actividad visible pero que no es la que interactúa con el usuario. Se llega a este estado si una *activity* transparente o que no esté a pantalla completa se activa frente a ella. Android las terminará en casos extremos, para obtener recursos para las *activities* activas.
- **Parada**. Cuando las activities se ocultan totalmente se dice que están paradas. La *activity* permanecerá en memoria, reteniendo la información de estado, pero se convierte en una candidata para ser terminada si el sistema requiere memoria. Cuando se para una aplicación es importante guardar los datos y la UI (Interfaz de Usuario) actual.
- **Inactiva**. Cuando se cierra una aplicación se dice que está inactiva. Estas aplicaciones se retiran de la pila y necesitan reinicializarse para que puedan ser vistas y utilizadas de nuevo.

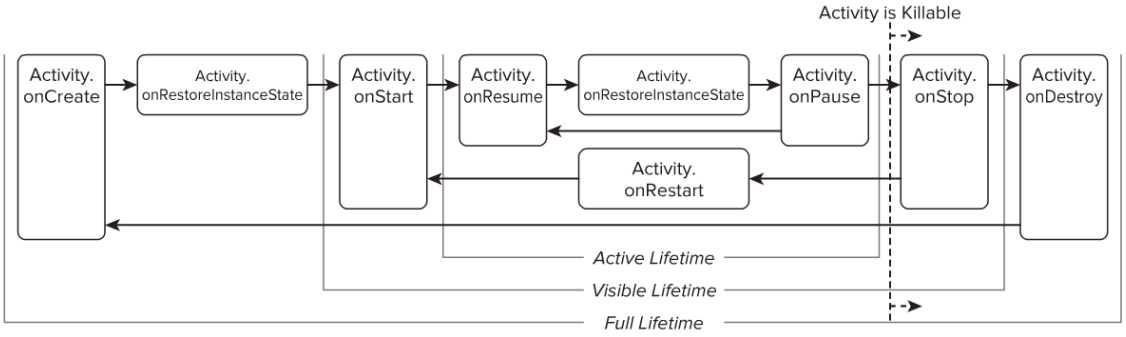

FIGURA 5: ESTADOS DE UNA ACTIVITY [4]

Elementos Visuales de Una Activity

❖ Views

Todos los componentes visuales de Android heredan de la clase *View*, y se llaman *views*. La clase *ViewGroup* es una extensión de *View* diseñada para contener múltiples *views*. Se utilizan para emplear componentes atómicos reusables o para controlar el *layout* de las *views* hijas (en estos casos se llamarán *layouts*).

❖ Layouts

El SDK de Android proporciona una serie de *layouts* simples que pueden ayudarnos a construir nuestra UI. Hay que elegir la combinación correcta de *layouts* para que nuestra interfaz sea fácil de entender y usar. Los tipos de *layouts* que encontramos en Android son:

- ***FrameLayout***. Es el más simple. Coloca cada *view* hijo en la esquina superior izquierda. Cada nuevo hijo se colocará encima del anterior.
- ***LinearLayout***. Alinea cada *view* hijo en líneas verticales u horizontales.
- ***RelativeLayout***. Es el más flexible. Permite definir la posición de cada *view* hijo en relación con otros y con los bordes de la pantalla.
- ***TableLayout***. Permite usar una cuadrícula de filas y columnas.
- ***GalleryLayout***. Despliega una fila de elementos en una lista horizontal con *scroll*.

❖ Fragments

Los *Fragments* [10] permiten dividir las *activities* en componentes encapsulados reutilizables, cada uno de ellos con su propio ciclo de vida e Interfaz de Usuario. Su principal ventaja es la facilidad con la que se pueden crear diseños de IU flexibles que se pueden adaptar a un rango muy grande de tamaños de pantalla, tanto de *smartphones* como de *tablets*. Cada *fragment* es un módulo independiente asociado a la *activity* en la que se encuentra. Pueden ser reutilizados por múltiples *activities*, así como utilizarse en una gran variedad de combinaciones, para utilizarse en las interfaces de usuario de las *tablets* con multipanel e interfaces dinámicas. No es necesario dividir las *activities* en *fragments* pero al hacerlo mejoramos drásticamente la flexibilidad de la IU y nos facilita la adaptación a las nuevas configuraciones de dispositivo para mejorar la experiencia de los usuarios. Los *fragments* aparecen en la versión 3.0 de Android.

❖ Widgets

Los widgets son componentes visuales de la aplicación que se añaden a la pantalla principal.

Servicios

Son los trabajadores invisibles de la aplicación. Los servicios trabajan en segundo plano y se encargan de actualizar los *content providers*, lanzar *intents* y notificaciones. Son un medio perfecto de manejar los eventos y las *activities* que están invisibles,

inactivas o se han cerrado. Los servicios se pueden usar para acciones que no dependan directamente de interacciones con los usuarios.

Content Providers

Se utilizan para controlar y compartir los conjuntos de datos de aplicación. Pueden configurarse para que otras aplicaciones accedan, así como puedes utilizar los *content providers* de terceros para acceder a sus datos almacenados. Android dispone de algunos de serie como los archivos multimedia y la lista de contactos.

Broadcast Receivers

Estos elementos recogen los mensajes de *broadcast* que envían el sistema u otras aplicaciones. Escuchan estos mensajes pero se configuran para que filtren por los criterios que interesen a la aplicación. Despertarán a la aplicación para que respondan a unos mensajes determinados si es necesario.

2.3.4.2. Intents e Intent Filters

Los *intents* e *intent filters* permiten lanzar los componentes de una aplicación Android descritos previamente.

Los *intents* permiten realizar paso de mensajes entre aplicaciones. Con ellos se pueden mandar mensajes de *broadcast* al sistema o hacia una *activity* o servicio determinado. Este mecanismo transforma el dispositivo en un sistema único interconectado compuesto por una colección de componentes independientes. Android realiza *broadcast* con *Intents* para anunciar eventos como cambios de la conexión de Internet, del nivel de carga de la batería, de una llamada, SMS recibido…

Si se requiere de un *intent* para llevar a cabo una acción en un conjunto de datos, en Android se utilizan los *intent filters* para declarar las acciones y los datos que soportarán estos intents.

2.3.4.3. Notificaciones al Usuario

Existen varios tipos de eventos para notificar al usuario. Las notificaciones y los *toasts* se emplean para informar a los usuarios sobre un evento concreto en una aplicación, sin forzarles a realizar una acción sobre ese evento.

Toast

Los *toasts* son interfaces de diálogo en forma de cajas que únicamente están visibles unos segundos antes de desvanecerse. No quitan visibilidad y no interrumpen las aplicaciones activas. Sirven para informar a los usuarios sin forzarles a abrir una *activity* o leer una notificación. Se utilizan sobre todo para avisar al usuario de eventos que ocurren en segundo plano.

Notificaciones

Las notificaciones permiten señalizar a los usuarios sin interrumpir la *activity* ejecutándose. Son la mejor técnica para captar la atención del usuario desde un servicio o un *broadcast receiver*. Las notificaciones pueden crear iconos en la barra de estado, mostrar información adicional al desplegar la barra de estado, encender luces o LEDs del dispositivo, activar la vibración del teléfono o activar alertas acústicas.

2.3.4.4. Adaptadores

Los adaptadores son unas clases que crean un "puente" entre los datos y las *views* y se utilizan en las interfaces de usuario. El adaptador es responsable de crear los *views* hijos usados para representar cada ítem dentro del *view* padre proporcionando acceso a los datos que estarán por debajo.

Android proporciona una serie de adaptadores que pueden modificar la apariencia y la funcionalidad de los controles a los que están asociados. Los adaptadores nativos que incluye Android son:

- ***ArrayAdapter***. Usa objetos genéricos para unir una *view* a los objetos especificados en esa clase.
- ***SimpleCursorAdapter***. Relaciona una *view* específica con un *layout* relacionado con un cursor que proviene de un *content provider*. El adaptador crea una nueva view por cada cursor y relaciona el *layout* con él.

2.3.4.5. Interacción con las Tablas

Con ayuda de los c*ursores* y los *content values* es posible modificar las tablas almacenadas en una aplicación Android.

Cursores

Las *queries* se devuelven con objetos de tipo *Cursor*. Los cursores son punteros a los resultados en la tabla. Existen una serie de funciones que permiten moverse fácilmente en la tabla con los cursores como moverse a la primera fila, a la siguiente fila, a la anterior, a la siguiente columna…

Content Values

Los *content values* se utilizan para insertar filas en las tablas. Cada objeto representa los datos de cada columna representados en una fila de la tabla.

2.3.4.6. Alarmas

Las alarmas son una forma independiente de la aplicación de lanzar *intents* a horas o intervalos determinados.

Las alarmas se programan lejos de la visión de las aplicaciones, por lo que se pueden utilizar para disparar eventos de las aplicaciones incluso cuando ya se han cerrado.

Pueden ser una herramienta útil combinada con un *broadcast receiver*, permitiendo que las alarmas arranquen *intents* de forma *broadcast*, servicios o incluso abrir *activities* sin que la aplicación se arranque hasta que se le requiera.

2.3.4.7. WakeLocks

Los *WakeLocks* son un servicio de control de la batería que permite a la aplicación controlar el estado de batería del dispositivo en que se encuentran. Se pueden utilizar para mantener la CPU funcionando, evitar que se reduzca la intensidad de luz de la pantalla, que se apague y evitar que se oculte el teclado.

2.3.4.8. Android Manifest

El *Android Manifest* es un archivo XML que debe estar presente en todas las aplicaciones Android [8]. El manifiesto proporciona información esencial sobre la aplicación al sistema Android, información necesaria por el sistema antes de ejecutar el propio código de la aplicación.

El manifiesto contiene el nombre del paquete Java asociado con la aplicación, que sirve como identificador único de la aplicación Android. Describe los diferentes componentes de la aplicación (*activities*, servicios, *broadcast receivers* y *content providers*). Indica qué clase define cada uno de estos componentes así como sus funcionalidades (como los *intents* que puede manejar).

En este archivo se especifican los diferentes permisos que la aplicación requiere para acceder a partes protegidas del API e interaccionar con otras aplicaciones. También se declaran los permisos que se requieren de los usuarios para interactuar con los componentes de la aplicación.

En el manifiesto se incluye además el nivel mínimo del API Android requerido por la aplicación para funcionar, así como el nivel máximo y el nivel objetivo para el que la aplicación fue desarrollada.

Es necesario también incluir en este fichero las diferentes librerías externas que la aplicación requiere para su correcto funcionamiento. Se emplea el *Android Manifest* para linkear estas librerías.

2.3.5. Permisos y Seguridad

Los permisos son un mecanismo de seguridad a nivel de aplicación que permite restringir el acceso a componentes de la aplicación. Se utilizan para evitar que aplicaciones maliciosas accedan a información sensible o hagan uso excesivo o no autorizado de los recursos *hardware* o de los canales de comunicaciones.

Cuando se instala una aplicación, los permisos requeridos en el manifiesto se analizan y se conceden (o deniegan) comprobando con una autoridad segura y con el *feedback* de los usuarios.

Todos los permisos de las aplicaciones Android se comprueban cuando se realiza la instalación. Cuando la aplicación se haya instalado, el usuario no podrá visualizar o reevaluar los permisos.

Es posible especificar el nivel de acceso que tiene un servicio: normal, *dangerous* (peligroso), *signature* (firma), *signatureOrSystem*.

Los permisos para los componentes de la aplicación se incluyen en su manifiesto. Algunos ejemplos útiles son los siguientes:

- ***Activities***. Se pueden añadir permisos para limitar la capacidad de otras aplicaciones de ejecutar una *activity*.
- ***Broadcast Receivers***. Se puede controlar qué aplicaciones pueden enviar *intents* por *broadcast* a mi aplicación.
- ***Content Providers***. Se puede limitar el acceso de lectura y escritura a ellos.

- **Servicios**. Se puede limitar la capacidad de otras aplicaciones de arrancar un servicio.

La mayoría de la seguridad de Android recae sobre el *kernel Linux*. Los recursos se asocian con los propietarios de las aplicaciones, por lo que no son accesibles para otros. Android proporciona servicio de *Broadcast* para *Intents*, *Servicios* y *Content Providers* para poder saltar esta barrera y permitir la compartición sin rebajar el nivel de seguridad.

Cada paquete Android tiene una ID única de *Linux* que se asigna durante la instalación. Tiene el efecto de asociar los recursos a quien los crea, para que no afecte a otras aplicaciones.

Por la seguridad del kernel, hay que utilizar *content providers* y mandar *intents* por *broadcast*. Estos mecanismos abren un túnel por el que la información puede pasarse entre las aplicaciones. Los permisos de Android dan seguridad para controlar el tráfico permitido entre ellas.

2.3.6. Funcionalidades y APIs Más Relevantes

2.3.6.1. Almacenamiento de Datos de la Aplicación

Existen diversas técnicas que permiten que Android almacene los datos de una forma rápida, eficiente y robusta.

Cuando se almacenan estados de la Interfaz de Usuario, preferencias de usuario o aplicaciones, se quiere un mecanismo ligero para almacenar una serie de datos compartidos. Mediante las preferencias compartidas se permite almacenar grupos de objetos y valores de forma primitiva.

Las *activities* incluyen controladores de eventos especializadas en almacenar el estado actual de la Interfaz de Usuario cuando la aplicación pasa a segundo plano.

Pese a estas opciones, en ocasiones escribir y leer desde ficheros es la mejor solución. Android permite crear y cargar ficheros en los sistemas externos o internos de memoria.

De manera alternativa al almacenamiento en ficheros, Android ofrece el uso de las librerías de *SQLite*. Las aplicaciones pueden crear sus propias bases de datos con completo control sobre ellas. Se utilizan para almacenar datos complejos y estructurados de las aplicaciones. Las BBDD son privadas por defecto, accesibles únicamente por la aplicación que las creó.

Para poder acceder a todos estos tipos de datos se emplean los *content providers*. Son interfaces genéricas, bien definidas para utilizar y compartir datos. Se pueden utilizar *content providers* para compartir datos entre las aplicaciones. Cualquier aplicación con los permisos apropiados puede añadir, eliminar o actualizar datos de cualquier otra aplicación que se encuentren en *content providers* (incluidas las nativas de Android). Algunas bases de datos que se almacenan de esta forma son los contactos del teléfono y los archivos multimedia.

2.3.6.2. Localización y Mapas

Los servicios basados en localización describen a varias tecnologías que se utilizan para conocer la ubicación geográfica del dispositivo. Android nos proporciona dos abstracciones para acceder a información de localización:

- ***Location Manager***. Proporciona enlaces a servicios basados en localización. Con él puedes obtener tu localización actual, controlar el movimiento, activar alertas de proximidad a puntos o áreas específicas y encontrar *Location Providers* disponibles.
- ***Location Providers***. Cada uno de ellos representa una tecnología diferente utilizada para conocer la ubicación del dispositivo.

Estos servicios son dependientes de algún tipo de *hardware* del dispositivo, como el GPS.

El geocodificador permite traducir entre direcciones y las coordenadas de latitud y longitud que se utilizan en un mapa. Estas consultas se hacen contra un servidor por lo que las aplicaciones que quieran utilizarlo deberán permitir el uso a la conexión a Internet. La geocodificación se puede realizar en los dos sentidos.

2.3.6.3. Cámara

La popularidad de las cámaras digitales incorporadas en los dispositivos móviles, ha conseguido que su precio se disminuya drásticamente, así como su tamaño. Es difícil en la actualidad encontrar un teléfono Android sin cámara.

Para hacer fotografías se pueden utilizar *intents*. En Android se capturan las imágenes en dos formatos a la vez:

- ***Thumbnail***. Por defecto cada fotografía realizada se almacenará con un *thumbnail* (miniatura). Utilizado para la organización y el reconocimiento de imágenes.
- **Imagen Completa**. Se almacenará en la ubicación indicada, si no en la carpeta por defecto.

2.3.6.4. Reconocimiento de Voz

Desde la versión 1.5 se soporta el reconocimiento de voz en Android. Cuando el usuario termina de hablar, el audio resultante se analizará y se procesará por un motor de reconocimiento del habla.

Desde la versión 1.6 se introdujo un motor de conversión de texto a audio. Se puede utilizar para que el dispositivo "hable" a partir de una entrada de texto introducida. Se puede utilizar para hablar con otros usuarios o para usuarios con discapacidad.

Debido a las restricciones de espacio de los dispositivos, los paquetes de lenguaje no están preinstalados en todos los dispositivos.

2.3.7. Requisitos de una Aplicación Android

Las aplicaciones Android deben programarse empleando una serie de mínimos. Es necesario asegurarse de que las aplicaciones:

- **Tienen un buen comportamiento**. Asegurarse de que estén suspendidas cuando no se encuentren en la pantalla principal.
- **Pasan de forma suave desde el segundo plano a la pantalla principal**. Es importante que aparezcan de forma rápida y suave. Esto se puede asegurar guardando el estado de la aplicación y encolando las actualizaciones para que los usuarios no aprecien la diferencia entre abrir la aplicación por primera vez o que aparezca desde el segundo plano. Los usuarios deberían recuperar la interfaz gráfica donde se quedó cuando cerraron la aplicación.
- **Son "educadas"**. La aplicación no debería interrumpir a otra activity que se está ejecutando en ese momento. El buen uso de las notificaciones llevará a las aplicaciones a un comportamiento óptimo entre ellas. Se pueden utilizar también sonidos, luces LED de las que disponga el dispositivo, vibración, etc.
- **Presentan una UI consistente**. Es posible que nuestra aplicación se utilice junto a otras simultáneamente, por lo que la UI tiene que ser fácil de usar.
- **Respondan**. La frustración de una "pantalla congelada" es muy grande, y en los dispositivos móviles aún más. Con la posibilidad de retrasos que causan las conexiones de datos es importante que la aplicación use hilos y servicios en segundo plano para que las aplicaciones respondan correctamente y para evitar que otras aplicaciones respondan ante ellas abruptamente. Android utiliza el *Activity Manager* y el *Window Manager* para controlar estos comportamientos. Si cualquiera de estos dos servicios detecta una aplicación que no responde, desplegará un mensaje hacia al usuario del tipo "Forzar Cierre". Esta alerta no desaparece hasta que pulsas al botón de forzar cierre o la aplicación vuelve a responder. Android controla dos condiciones para ver si una aplicación responde o no:
  - La aplicación responde a cualquier acción del usuario, como pulsar una tecla o tocar la pantalla en menos de 5 segundos.
  - Devolver una señal de *broadcast* desde la aplicación en menos de 10 segundos.

Los usuarios tienen que ser responsables con las aplicaciones que instalan y los permisos que aceptan. El modelo de seguridad de Android restringe el acceso a ciertos servicios y funcionalidades forzando a las aplicaciones a declarar los permisos que requieren. Durante la instalación se muestran a los usuarios los permisos que requieren las aplicaciones para ser instaladas. Esto no libera de responsabilidad al programador. Hay que asegurarse de que la aplicación sea segura además de por nuestra propia seguridad, para evitar que la seguridad del dispositivo se comprometa.

2.3.8. Vender, Promocionar y Distribuir Aplicaciones

Tras crear una nueva aplicación, el siguiente paso es compartirla con el mundo. Las aplicaciones Android se distribuyen en archivos de paquetes (.apk). Para poderse instalar en un dispositivo, los paquetes necesitan estar firmados. Antes de distribuir la aplicación, es necesario compilarla y firmarla usando una clave privada para demostrar a los usuarios que la aplicación proviene realmente de donde se indica. Es importante remarcar que los nombres de los paquetes de aplicación deben llevar un identificador único para cada aplicación. Así, cada variación de la aplicación llevará un nombre de paquete único. No es el nombre del archivo apk lo que debe ser único, sino el identificador de la aplicación.

Una de las ventajas del ecosistema abierto de Android es la libertad de publicar y distribuir las aplicaciones cómo y cuándo queramos. El canal más común de distribución es *Google Play* (*Android Market* en versión 2), pero existen otros canales y mercados alternativos para realizarlo.

La tienda de Google "*Google Play*" es el punto más grande y más popular de distribución de aplicaciones Android. Funciona como un mercado en el cual se pueden distribuir y adquirir aplicaciones. Hay pocos controles que restrinjan cómo se promocionan, distribuyen y venden las aplicaciones. Estas restricciones se encuentran en el *Google Play Developer Distribution Agreement* (DDA) [12] y el *Google Play Developer Program Policies* (DPP) [13] .

En el mercado de Android no se realiza un proceso de revisión antes de que las aplicaciones se publiquen (como ocurre en la App Store de Apple o en la tienda de *Windows Phone*). Esto ocurre tanto con las nuevas aplicaciones, como con sus actualizaciones, lo que permite publicar y actualizar las aplicaciones cuando queramos, sin necesidad de esperar a una aprobación. Si se sospecha que una aplicación ha roto los acuerdos indicados anteriormente, se revisa, y si se comprueba este hecho, la aplicación se suspende y el desarrollador es notificado. Si se produce un caso de malware extremo, la tienda Google Play tiene la capacidad de desinstalar estas aplicaciones maliciosas directamente de los dispositivos.

Google Play proporciona todas las herramientas y mecanismos necesarios para distribuir, actualizar, vender (nacional e internacionalmente) y promocionar. Una vez que publicamos nuestra nueva aplicación, podremos consultarla, junto al resto de nuestras aplicaciones en la herramienta del desarrollador. Podemos consultar cuántos usuarios la han instalado y diversas estadísticas sobre la aplicación.

El *feedback* directo con los usuarios es algo difícil de evaluar, puesto que puede ser poco fiable (comentarios negativos a propósito desde la competencia) y contradictorio. Por este motivo se generan una serie de estadísticas que nos ayudan a evaluar mejor estos comentarios. Estas estadísticas incluyen el comportamiento de los usuarios e incluso lo comparan con otras aplicaciones de la misma categoría. Incluyen porcentaje de usuarios que la utilizan divididos por plataforma, dispositivo hardware, país e idioma.

Esta información puede ser muy útil para decidir dónde distribuir las aplicaciones, así como las versiones de Android a las que queremos dar soporte. También puede ayudarnos a encontrar qué áreas de nuestra aplicación podemos mejorar, o no funcionan correctamente.

Android permite ganar dinero con las aplicaciones por diferentes mecanismos. Si nuestro medio de distribución es Google Play hay tres alternativas disponibles:

- **Aplicaciones de Pago**. Los usuarios pagan un pago previo antes de descargar e instalar la aplicación.
- **Aplicaciones gratuitas con pago por servicios**. La descarga e instalación de la aplicación es gratuita pero se paga por algunos servicios virtuales de la aplicación, por las actualizaciones u otros servicios de valor añadido.
- **Publicidad en las aplicaciones**. Se distribuye la aplicación gratuitamente pero se recibe un cobro por la publicidad desplegada en la aplicación.

Si se utilizan cualquiera de los dos primeros métodos las ganancias se reparten entre el desarrollador y *Google Play*. En este momento los beneficios se reparten ofreciendo un 70% a los desarrolladores.

Si se utiliza publicidad los beneficios dependerán del tipo de anuncios que se expongan en la aplicación. Es importante que los anuncios sean lo menos intrusivos posible y que no degraden la experiencia del usuario con la aplicación. En ocasiones, los usuarios prefieren pagar previamente por la aplicación o por servicios de valor añadido, antes que tener banners con anuncios que desvirtúen el contenido de la aplicación.

2.3.9. Conclusiones y Aplicaciones Android para el Proyecto

El 78.9% de los *smartphones* vendidos en el primer trimestre de 2015 utiliza el sistema operativo Android [17]. Esta afirmación lleva a pensar que el mercado potencial de clientes de aplicaciones Android es muy grande, mucho mayor que el de sus competidores. Por lo que cualquier aplicación innovadora lanzada para este sistema operativo puede tener una repercusión de usuarios muy grande, atacando a una cuota de mercado en crecimiento. Además, la potencia y simplicidad de su lenguaje hace que el aprendizaje del desarrollo de nuevas aplicaciones de este tipo sea una tarea interesante y amena para un nuevo potencial desarrollador.

El sistema operativo continúa evolucionando a gran velocidad, aumentando su incursión en nuevos nichos de mercado (televisiones, *wearables*, vehículos…) lo cual hace de Android una plataforma muy atractiva en la que realizar el desarrollo de una nueva aplicación móvil.

En el piloto de aplicación Android sobre DTN realizado para este proyecto se emplearán todos los elementos de la aplicación mencionados previamente, así como APIs específicas para mapas, control de la cámara, almacenamiento de datos en SQLite. Los *intents* en la aplicación cobrarán mucha importancia para poder realizar paso de mensajes con el master de DTN.

Capítulo 3: IBR-DTN

3.1. Introducción

El proyecto IBR-DTN [2] consiste en una implementación del Bundle Protocol (RFC 5050 [3]) para DTN realizado por el Instituto IBR (*Institut für Betriebssysteme und Rechnerverbund*) de la Universidad Alemana *Technische Universität Braunschweig*. El proyecto fue originalmente pensado para sistemas embebidos pero dispone de APIs en varios lenguajes de programación que pueden ser utilizados para realizar aplicaciones DTN [14].

Para el ámbito que abarca este Proyecto Fin de Carrera, se ha utilizado la implementación Android de IBR-DTN [11]. El grupo IBR-DTN dispone de una serie de apps publicadas en *Google Play*, así como en repositorios de *GitHub* para obtener el código de desarrollo de estas aplicaciones [15].

En este capítulo se expondrá la configuración necesaria para poder desarrollar una aplicación utilizando la implementación de DTN realizada por el grupo IBR-DTN.

3.2. IBR-DTN Master

El demonio DTN del grupo IBR puede encontrarse en Google Play [9] de forma gratuita. La versión utilizada para la realización del proyecto es la 1.0. Publicada en Google Play el día 24 de Febrero de 2015. Esta aplicación consiste en un demonio basado en el *Bundle Protocol* (RFC 5050 [3]). Funciona como un API para que las aplicaciones DTN puedan comunicarse entre ellas.

Todas las aplicaciones que utilicen la implementación de DTN del grupo IBR-DTN necesitan tener activado el demonio DTN para poder ser utilizadas. Este paso será igualmente necesario para la utilización de la aplicación que se ha realizado para el piloto de demostración de este proyecto. Por este motivo, es necesario dedicar un apartado a la configuración y el estudio de esta aplicación.

3.2.1. Soporte de la Aplicación IBR-DTN

Esta aplicación, según se indica en la web del grupo IBR-DTN [14] contiene la funcionalidad de la RFC 5050 [3] así como una serie de características, las más relevantes que se listan a continuación:

- Protocolo de Seguridad de los *bundles* especificado en la RFC 6257 [23].
- API basada en *sockets*.
- Soporte entre *bundles*.
- Soporte IPv6
- Capa de convergencia TCP/IP y UDP/IP.

- Diferentes modelos de *routing*: Con conexiones estáticas, envío tras el descubrimiento, envío masivo mediante inundación (*flooding*).
- Diferentes modelos de almacenamiento: Almacenamiento basado en memoria y almacenamiento persistente en el dispositivo.

3.2.2. Configuración y Parámetros De La Aplicación IBR-DTN Master

La configuración de la aplicación es bastante básica e intuitiva. En este apartado se explicarán los principales parámetros y cómo se han configurado para llevar a cabo este proyecto.

En la Figura 6 se pueden observar los primeros parámetros de la aplicación. En la parte superior aparece una barra de herramientas junto al título, en la que aparecen cuatro botones. El primer botón enciende o apaga el demonio. El segundo muestra los vecinos DTN que se encuentran en nuestra red. El tercero muestra estadísticas de uso, almacenamiento y envío de *bundles*. El último botón oculta las dos últimas opciones de configuración que son la clave local y el log de registro de eventos DTN.

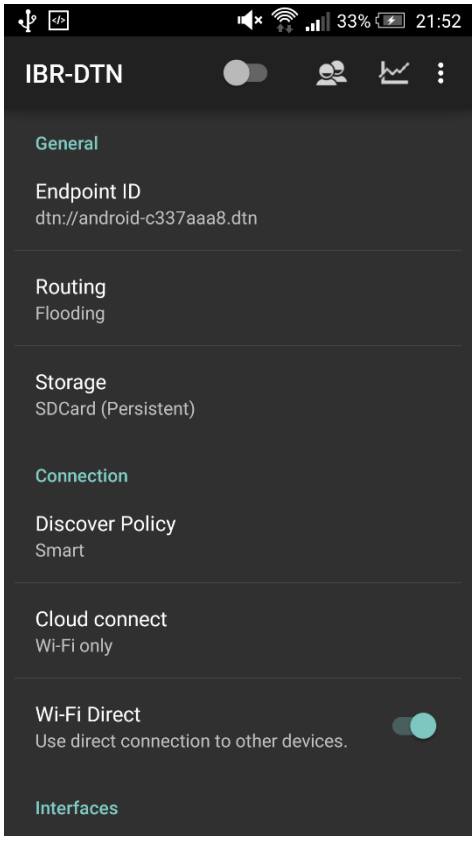

FIGURA 6: CONFIGURACIÓN IBR-DTN MASTER (I)

Aparecen también en la Figura 6 las primeras opciones de ajuste de la aplicación en una *ListView*. Se irán añadiendo la Figura 7 y la Figura 8 que mostrarán de forma continuada el resto de elementos de configuración. Estos elementos se detallarán a continuación para cada uno de los apartados.

- General:
  - ***Endpoint ID***: Identificador DTN del *Endpoint* (terminal móvil en este caso). Para el ejemplo tenemos el *Endpoint* dtn://android-c337aaa8.dtn
  - ***Routing***: Tipo de enrutamiento que realiza la aplicación. Existen cinco posibilidades de configuración: *Disabled*, *Direct-Delivery*, *Flooding*, *Epidemic Routing* y *PRoPHET Routing*. La configuración elegida es *Flooding*.
  - ***Storage***: Tipo de almacenamiento de los *bundles*. Puede ser en la tarjeta SD (de forma Persistente o no persistente) o en memoria. El almacenamiento escogido es en la tarjeta SD de forma persistente.

- Connection:
  - ***Discover Policy*:** Política de descubrimiento de vecinos. Puede estar apagada, en modo inteligente, o siempre encendida. Para optimizar recursos se decide emplear el modo inteligente (*Smart*).
  - ***Cloud connect*:** Posibilidades de conexión a la *cloud DTN* del grupo IBR-DTN. Permite estar apagado, únicamente conectarse por *Wi-Fi* o siempre encendido. Por motivos de utilización de red lo configuramos para utilizarse únicamente por *Wi-Fi*.
  - ***Wi-Fi Direct*:** Permite la conexión directa a otros dispositivos a través de *Wi-Fi*. Esta opción se mantendrá habilitada.

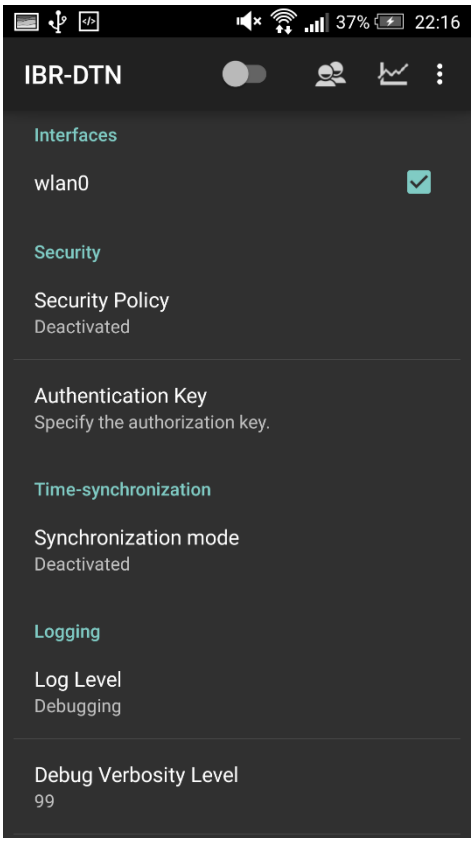

FIGURA 7: CONFIGURACIÓN IBR-DTN MASTER (II)

- **Interfaces**: Lista de interfaces disponibles en el dispositivo para transmitir *bundles*. En este caso la única disponible es la interfaz *Wi-Fi* wlan0.

- Security:
  - ***Security Policy*:** Indica si se requiere autenticación (BAB) o no. En nuestro caso trabajaremos con la autenticación desactivada.
  - ***Authentication Key*:** Clave de autenticación. En nuestro caso no utilizaremos clave de autenticación.

- Time-syncronization:
  - ***Synchronization mode*:** Elemento de sincronización maestro o esclavo. En nuestro caso se encuentra deshabilitado.

- Logging:
  - ***Log Level*:** Permite configurar el nivel de log. La utilizaremos en modo *debug*.
  - ***Debug Verbosity Level*:** Nivel de verbosidad del log cuando se encuentra a *debug*. Utilizamos el nivel 99.
  - ***Log to file*:** Permite almacenar el log externamente en un fichero. Para nuestro ejemplo se mantendrá desactivado.

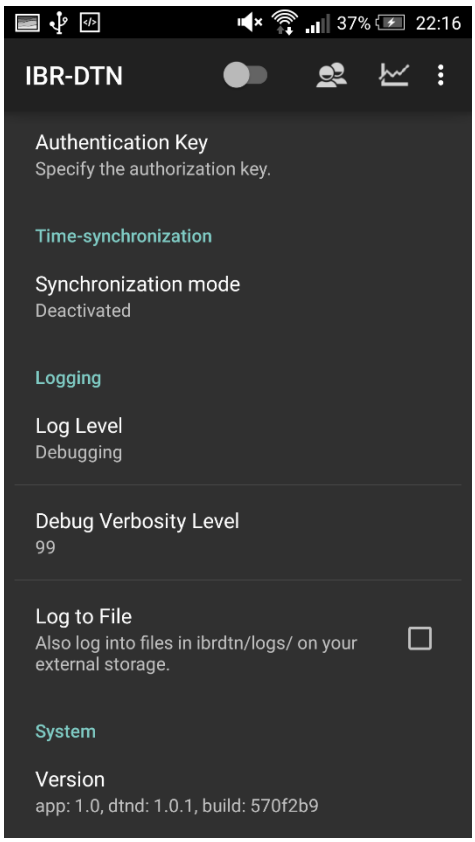

FIGURA 8: CONFIGURACIÓN IBR-DTN MASTER (III)

- System: Detalles de la versión y el build de la aplicación.

3.2.3. Aplicaciones IBR-DTN

Como se ha comentado previamente, el grupo IBR-DTN dispone de una serie de aplicaciones para poder probar las capacidades del demonio DTN que han desarrollado

en Android [4]. Las tres aplicaciones de ejemplo muestran la posibilidad de transferencia de información en diferentes ámbitos empleando el protocolo de envío de *bundles* DTN [3]. Es necesario que el demonio IBR-DTN esté instalado, ejecutándose y activado para que estas aplicaciones puedan transmitir y recibir datos correctamente.

- ***Whisper***: Aplicación de chat utilizando el protocolo DTN. Permite enviar mensajes de texto a los contactos que tengan activado el demonio DTN a nuestro alrededor.
- ***Talkie***: Es una aplicación de tipo chat mediante notas de voz. Los mensajes enviados se transmiten mediante *broadcast* a los dispositivos cercanos conectados al demonio IBR-DTN. Esta aplicación puede combinarse con *Whisper* para enviar mensajes directamente a un contacto específico.
- ***ShareBox***: Esta aplicación puede emplearse para compartir ficheros con los dispositivos que tienes alrededor, utilizando el demonio IBR-DTN.

El código de estas aplicaciones de ejemplo de la utilización de IBR-DTN puede encontrarse también en el repositorio de código de *GitHub* del grupo IBR-DTN [15].

3.2.4. Desarrollo de Aplicaciones IBR-DTN en Android

Es necesario seguir una serie de pautas para que la aplicación que se va a desarrollar como piloto DTN para este proyecto pueda utilizar el demonio IBR-DTN para entablar las comunicaciones necesarias [11].

Inicialmente, es imprescindible configurar el *Android Manifest* para que se dé permiso a que se registren los *bunldes* que se envíen o sean recibidos por la aplicación.

```
<uses-permission android:name="de.tubs.ibr.dtn.permission.DTN_COMMUNICATION" />
```

Además, es necesario incluir en el *Manifest* una serie de *intent filters* para que la aplicación sea listada en el demonio IBR-DTN. Estos *intents* declaran la *activity* como una *activity* DTN.

```
<activity android:name=".MainActivity" android:label="@string/app_name" android:launchMode="standard" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="de.tubs.ibr.dtn.intent.DTNAPP" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <intent-filter>
                <action android:name="de.tubs.ibr.dtn.intent.ENDPOINT_INTERACT" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
</activity>
```

También debe incluirse en el archivo `AndroidManifest.xml` un *intent receiver* que registre una serie de eventos que mandará el demonio IBR-DTN. Los *intents* que se pueden emplear en estas aplicaciones basadas en la implementación de IBR-DTN son:

- **de.tubs.ibr.dtn.intent.STATE**: Señaliza el estado del demonio a la aplicación. Con este *intent* puede despertarse a la aplicación cuando se active el demonio.

- **de.tubs.ibr.dtn.intent.REGISTRATION**: El intent *REGISTRATION* se lanza cuando se realiza un registro de la aplicación con éxito.
- **de.tubs.ibr.dtn.intent.RECEIVE**: Cuando se recibe un nuevo *bundle* por parte de la aplicación se llama a este *intent*.
- **de.tubs.ibr.dtn.intent.STATUS_REPORT**: Este intent se lanzará cuando se reciba un *status report* sobre un *bundle* enviado por nuestra aplicación.

```
<receiver     android:name=".DtnServiceReceiver"     android:enabled="true"     android:exported="true"
android:permission="de.tubs.ibr.dtn.permission.DTN_SERVER" >
            <!-- Add intent filter for notification of incoming bundles -->
            <intent-filter>
                <action android:name="de.tubs.ibr.dtn.intent.RECEIVE" />
                <category android:name="com.pfc.dtn.around" />
            </intent-filter>
            <intent-filter>
                <action android:name="de.tubs.ibr.dtn.intent.STATE" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
            <!-- Add intent filter for status reports generated by the DTN service -->
            <intent-filter>
                <action android:name="de.tubs.ibr.dtn.intent.STATUS_REPORT" />
                <category android:name="com.pfc.dtn.around" />
            </intent-filter>
            <intent-filter>
                <action android:name="de.tubs.ibr.dtn.intent.REGISTRATION" />
                <category android:name="com.pfc.dtn.around" />
            </intent-filter>
</receiver>
```

Una vez asignados todos los permisos necesarios, y habiendo habilitado los *intents* necesarios para entablar comunicaciones a través de IBR-DTN en Android puede comenzarse el desarrollo de una nueva aplicación para poner en práctica lo aprendido con este protocolo. En la web de IBR-DTN [11] podemos encontrar un binario (.jar) con el API de IBR-DTN, que será necesario para poder utilizar las clases y métodos del código del IBR-DTN *master* necesarias para desarrollar la aplicación Android.

3.2.5. Testing

Para demostrar la validez de la solución y poder afirmar que realmente esta aplicación está utilizando el protocolo DTN para enviar los mensajes (y que por lo tanto la realizada para el piloto del proyecto).

En los siguientes apartados se analizan las tramas observadas en una captura de tráfico realizada desde uno de los dos elementos presentes en la comunicación mediante la aplicación "*Whisper*" que utiliza el demonio IBR-DTN. Para realizar estas pruebas se ha utilizado una red *Wi-Fi* privada.

En estas pruebas se han empleado los siguientes dispositivos como usuarios:

- **Teléfono Móvil**: Teléfono Sony Ericsson Xperia Arc (LT15i) con Android 4.0.4. Dirección MAC de Wi-Fi: 30:17:c8:6d:52:b7. En este dispositivo se añadió una aplicación de tipo Sniffer de tráfico para capturar las tramas enviadas y recibidas por el dispositivo. Para utilizar esta aplicación fue necesario asignar a dicho dispositivo permisos de superusuario (root). Dirección IP: 192.168.1.13
- **Tablet**: BQ Edison 2 con Android 4.1.1. Dirección MAC de Wi-Fi: 00:22:f4:9f:02:c7. Dirección IP: 192.168.1.14

3.2.5.1. Multicast Listener Report Message

Estas tramas de la Figura 9 se observan en el momento en que cualquiera de los dos dispositivos se conecta con el demonio IBR-DTN. Es un mensaje con dirección de destino *multicast*, por lo que parece que muestra la presencia del dispositivo en la red DTN.

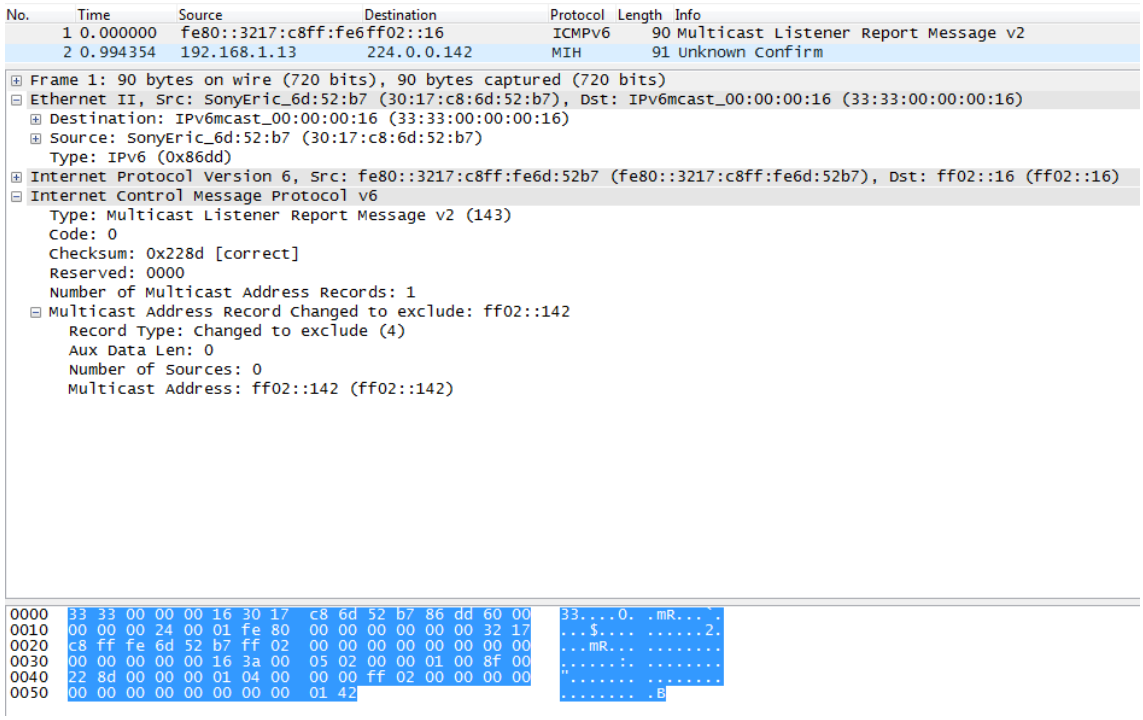

FIGURA 9: MULTICAST LISTENER REPORT MESSAGE

3.2.5.2. Media Interpreter Handover

Esta trama se manda desde cualquiera de los dispositivos a la dirección 224.0.0.142 que es el *Discovery Agent* presente en la red de IBR-DTN para poder controlar la presencia de los dispositivos en esta red. Se envían mensajes de este tipo desde cualquier dispositivo que utiliza el demonio DTN cada segundo. Este envío es configurable desde la interfaz del demonio. Se observa en la Figura 10 el envío de la dirección MAC del dispositivo móvil en la trama. Existe un mensaje de Respuesta a la petición.

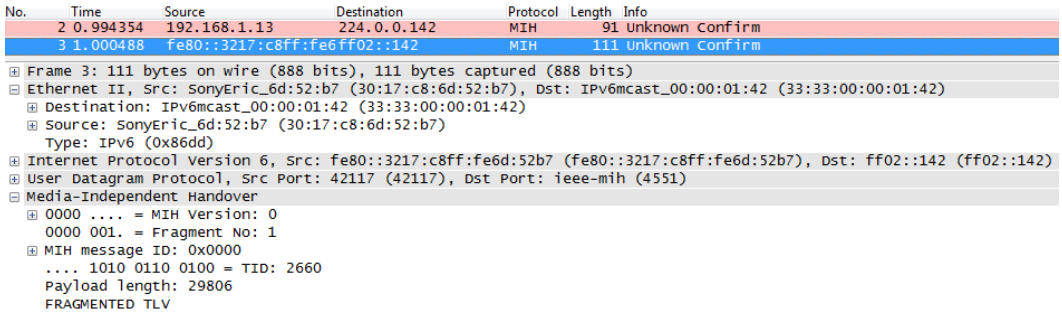

FIGURA 10: MEDIA INTERPRETER HANDOVER (PETICIÓN)

3.2.5.3. Peticiones ARP

Se observan entre las tramas las peticiones típicas de direcciones MAC utilizando el protocolo ARP como puede verse en la Figura 11.

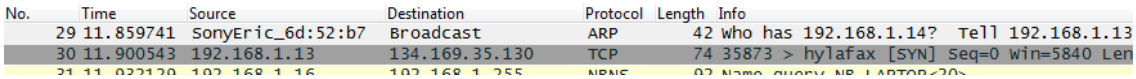

FIGURA 11: PETICIONES ARP

3.2.5.4. TCP Three Way Handshake (en Ambos Sentidos)

Los mensajes DTN utilizan el protocolo de transporte TCP, por lo que para iniciar una sesión es necesario según este protocolo realizar la negociación de la conexión, conocida como Three Way Handshake. Como se observa en las imágenes siguientes, este intercambio se produce en ambos sentidos. Esto es un hecho peculiar puesto que TCP establece una conexión *Full Duplex* (lo que indica que únicamente es necesario originar la conexión desde un sentido para poder realizar la comunicación entre ambos) [21]. Esta es una característica impuesta por la solución de IBR-DTN, ya que no se encuentra especificada en el estándar de DTN [1]. Se muestra con detalle en la Figura 12.

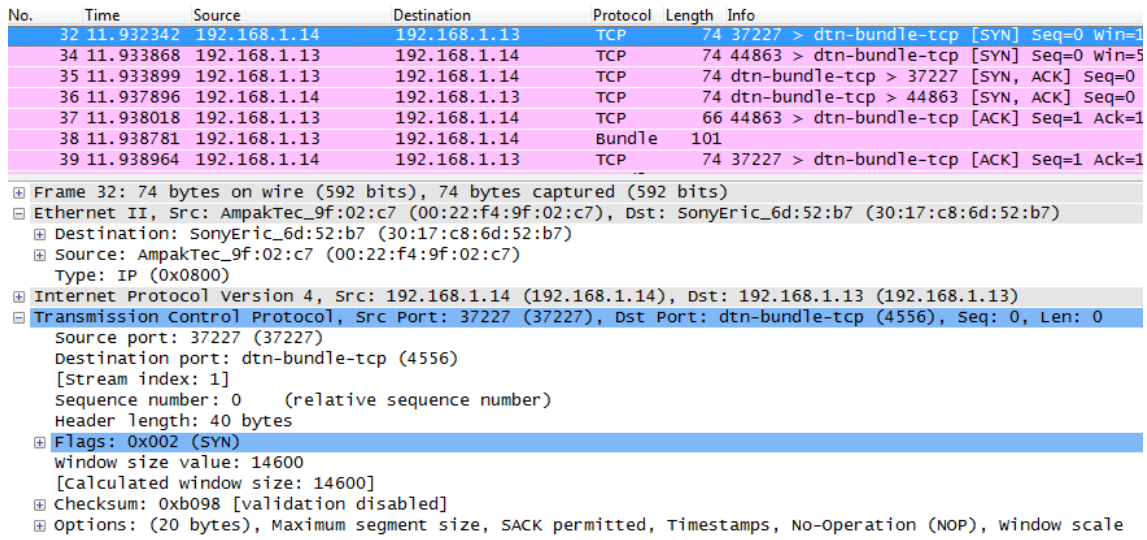

FIGURA 12: SEGUIMIENTO DEL ESTABLECIMIENTO DE LA CONEXIÓN

3.2.5.5. Primeros Bundles DTN: Capa de Convergencia

Se observan en las dos imágenes siguientes los primeros *bundles* DTN enviados entre los dispositivos. Parece que estos *bundles* muestran la presencia de los vecinos en la red DTN. Se observa el EID local del *bundle* que transmitió el dispositivo en la Figura 13 y el EID local del otro dispositivo en la Figura 14. Esta negociación es necesaria por la capa de convergencia TCP que se utiliza para enviar los *bundles* a través del protocolo de transporte DTN.

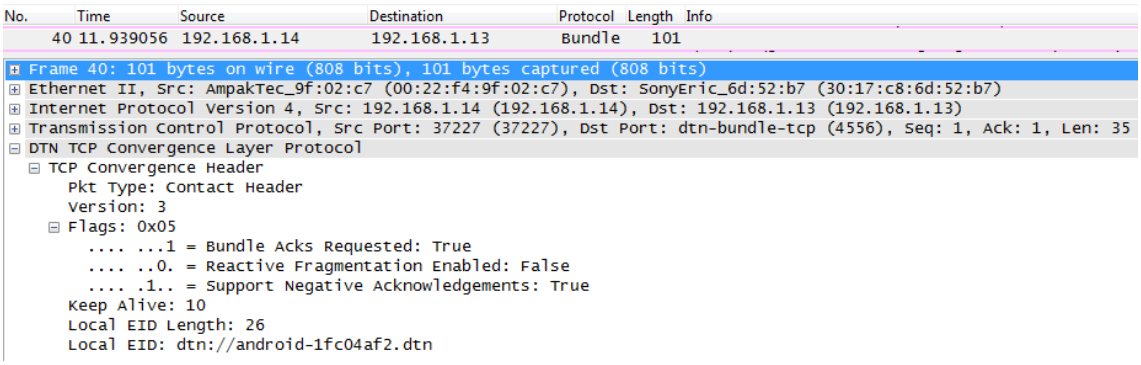

FIGURA 13: PRIMER BUNDLE DE TABLET A MÓVIL

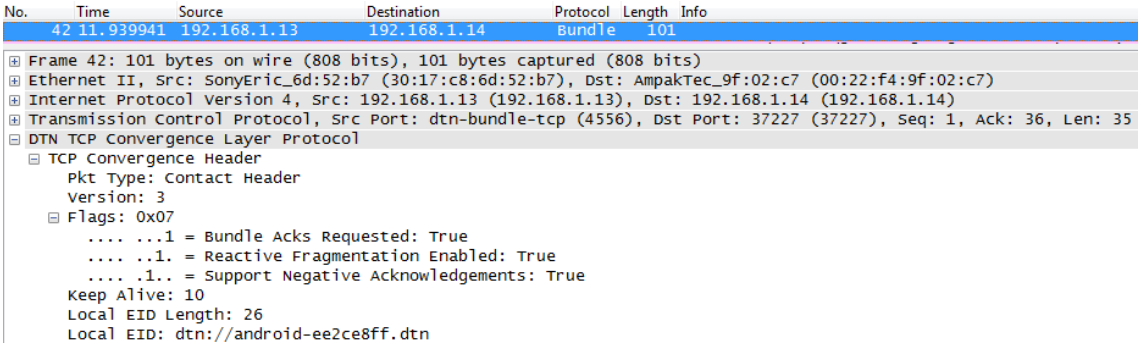

FIGURA 14: PRIMER BUNDLE DE MÓVIL A TABLET

3.2.5.6. Enrutamiento de los Dispositivos

Se observan una serie de mensajes que indican el enrutamiento entre los dos nodos DTN. Se han detallado las opciones de la cabecera *bundle* DTN para conocer más detalles en la Figura 15.

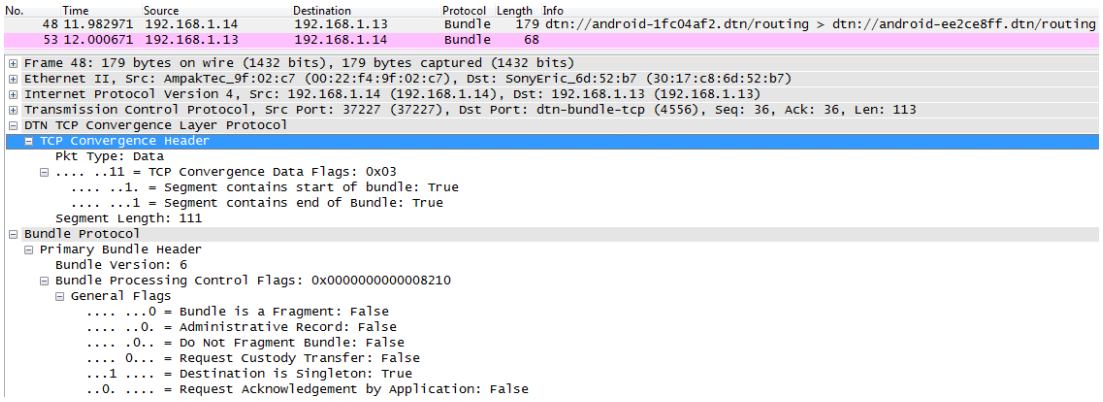

FIGURA 15: ROUTING BUNDLE

En la primera parte de la cabecera detallada en la Figura 16 se observan algunos elementos interesantes. Lo primero que se observa es la capa de convergencia entre la capa *bundle* y la TCP. Indica que ya hay datos DTN en este paquete. Se observa la prioridad: *Expedited* (alta prioridad). También se ve que no hay fragmentación, así como hay un único destinatario del *bundle*.

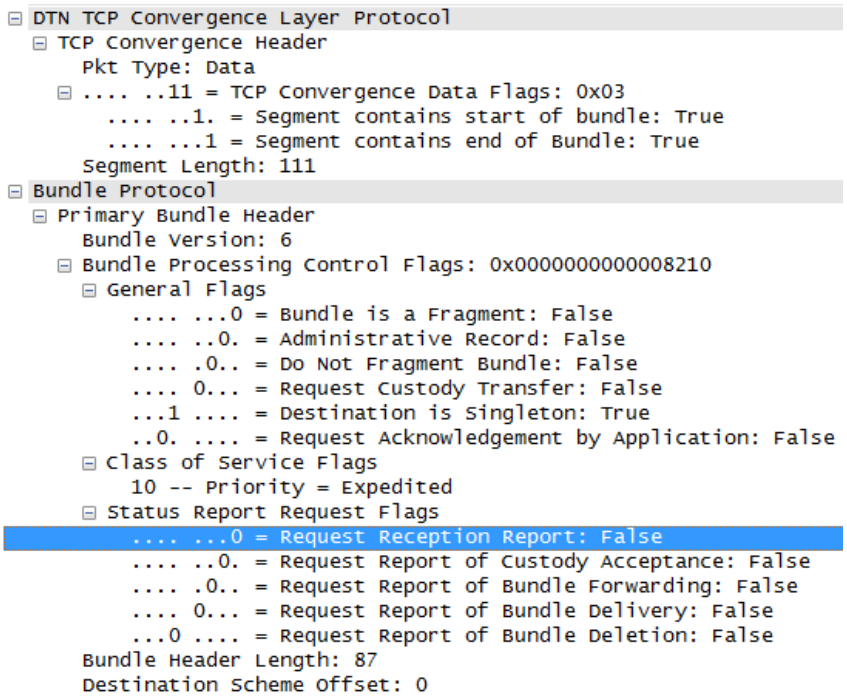

FIGURA 16: CABERA BUNDLE (I)

Se observa en la Figura 17 el EID DTN, con la dirección DTN en formato de URI, especificando a una dirección de enrutamiento (*routing*)

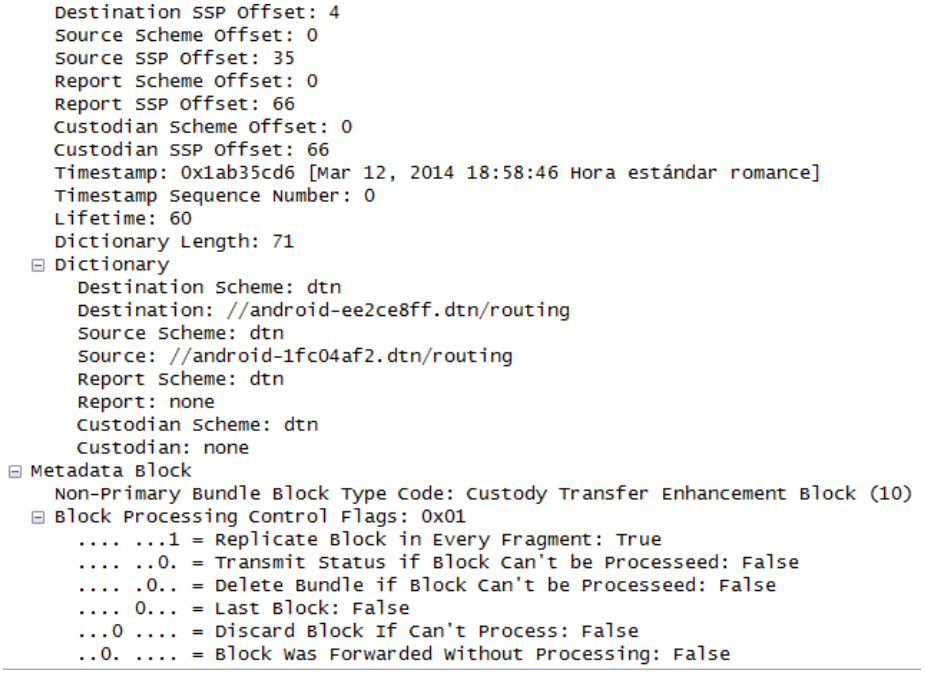

FIGURA 17: CABECERA BUNDLE (II)

Se puede comprobar en la Figura 18 que es el último bloque y que únicamente servirá para enrutar los *bundles* posteriores.

```
        .0.. .... = Block Contains an EID-reference Field: False
      Block Length: 3
      Block Processing Control Flags: ERROR: Replicate must be clear for CTEB
      CTEB Custody ID: 25160
      CTEB Creator Custodian EID:
      CTEB Is NOT Valid (Bundle Custodian [dtn:none] != CTEB Custodian [])
  ⊟ Metadata Block
      Non-Primary Bundle Block Type Code: Unknown (199)
    ⊟ Block Processing Control Flags: 0x01
        .... ...1 = Replicate Block in Every Fragment: True
        .... ..0. = Transmit Status if Block Can't be Processeed: False
        .... .0.. = Delete Bundle if Block Can't be Processeed: False
        .... 0... = Last Block: False
        ...0 .... = Discard Block If Can't Process: False
        ..0. .... = Block Was Forwarded Without Processing: False
        .0.. .... = Block Contains an EID-reference Field: False
      Block Length: 2
  ⊟ Payload Header
      Header Type: 1
    ⊟ Block Processing Control Flags: 0x08
        .... ...0 = Replicate Block in Every Fragment: False
        .... ..0. = Transmit Status if Block Can't be Processeed: False
        .... .0.. = Delete Bundle if Block Can't be Processeed: False
        .... 1... = Last Block: True
        ...0 .... = Discard Block If Can't Process: False
        ..0. .... = Block Was Forwarded Without Processing: False
        .0.. .... = Block Contains an EID-reference Field: False
      Payload Length: 6
```

FIGURA 18: CABECERA BUNDLE (III)

Se produce un ACK de respuesta al mensaje que se observa en la Figura 19.

```
No.   Time         Source          Destination     Protocol  Length  Info
   48 11.982971  192.168.1.14    192.168.1.13    Bundle    179 dtn://android-1fc04af2.dtn/routing > dtn://android-ee2ce8ff.dtn/routing
   53 12.000671  192.168.1.13    192.168.1.14    Bundle     68
⊞ Frame 53: 68 bytes on wire (544 bits), 68 bytes captured (544 bits)
⊞ Ethernet II, Src: SonyEric_6d:52:b7 (30:17:c8:6d:52:b7), Dst: AmpakTec_9f:02:c7 (00:22:f4:9f:02:c7)
⊞ Internet Protocol Version 4, Src: 192.168.1.13 (192.168.1.13), Dst: 192.168.1.14 (192.168.1.14)
⊞ Transmission Control Protocol, Src Port: dtn-bundle-tcp (4556), Dst Port: 37227 (37227), Seq: 36, Ack: 149, Len: 2
⊟ DTN TCP Convergence Layer Protocol
  ⊟ TCP Convergence Header
      Pkt Type: Ack
      Ack Length: 111
```

FIGURA 19: ACK AL ROUTING BUNDLE

Curiosamente, desde la capa TCP se envía otro ACK al destino. Esto es debido a que cómo utilizan el protocolo TCP, la trama anterior (aunque sea un ACK a nivel de capa bundle) tiene que tener su ACK.

Se aprecia en la Figura 20 el ACK TCP al bundle de ACK:

```
No.   Time         Source          Destination     Protocol  Length  Info
   55 12.036377  192.168.1.14    192.168.1.13    TCP         66 37227 > dtn-bundle-tcp [ACK] Seq=149 Ack=38 Win=14656 Len=0 TSval=6777625 TSecr=34668611
⊞ Frame 55: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
⊞ Ethernet II, Src: AmpakTec_9f:02:c7 (00:22:f4:9f:02:c7), Dst: SonyEric_6d:52:b7 (30:17:c8:6d:52:b7)
⊞ Internet Protocol Version 4, Src: 192.168.1.14 (192.168.1.14), Dst: 192.168.1.13 (192.168.1.13)
⊟ Transmission Control Protocol, Src Port: 37227 (37227), Dst Port: dtn-bundle-tcp (4556), Seq: 149, Ack: 38, Len: 0
    Source port: 37227 (37227)
    Destination port: dtn-bundle-tcp (4556)
    [Stream index: 1]
    Sequence number: 149    (relative sequence number)
    Acknowledgment number: 38    (relative ack number)
    Header length: 32 bytes
  ⊞ Flags: 0x010 (ACK)
    Window size value: 229
    [Calculated window size: 14656]
    [Window size scaling factor: 64]
  ⊞ Checksum: 0x74bf [validation disabled]
  ⊞ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ⊟ [SEQ/ACK analysis]
      [This is an ACK to the segment in frame: 53]
      [The RTT to ACK the segment was: 0.035706000 seconds]
```

FIGURA 20: TCP ACK AL ROUTING BUNDLE ACK

3.2.5.7. Keep Alives y ACKs (en Ambos Sentidos)

Se pueden ver una serie de mensajes *Keep Alive* de forma periódica desde ambos dispositivos para mantener activa la conexión. Estos mensajes *Keep Alive* tienen un ACK

asociado de la otra parte desde el protocolo TCP por el mismo motivo que ocurría anteriormente como se visualiza en la Figura 21.

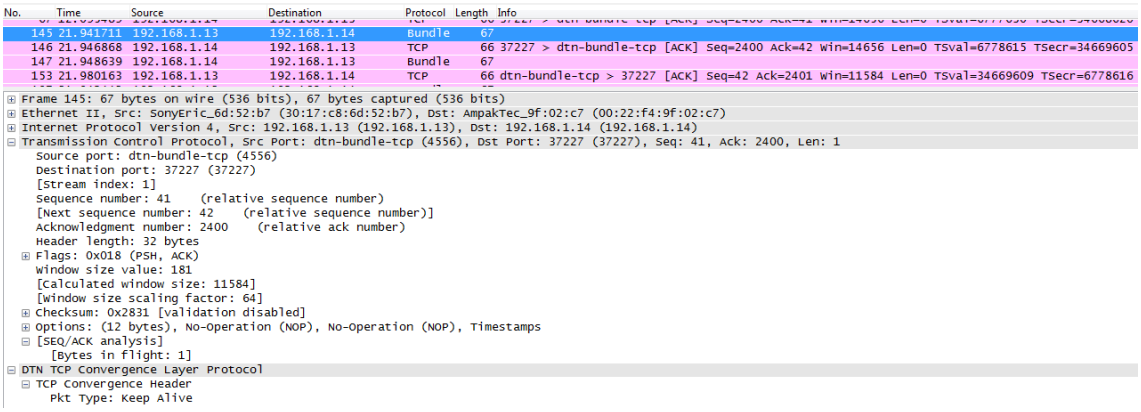

FIGURA 21: KEEP ALIVE

3.2.5.8. Mensaje del Chat

En este punto se pasa a observar el envío de un mensaje desde la aplicación de chat "*Whisper*" en la Figura 22.

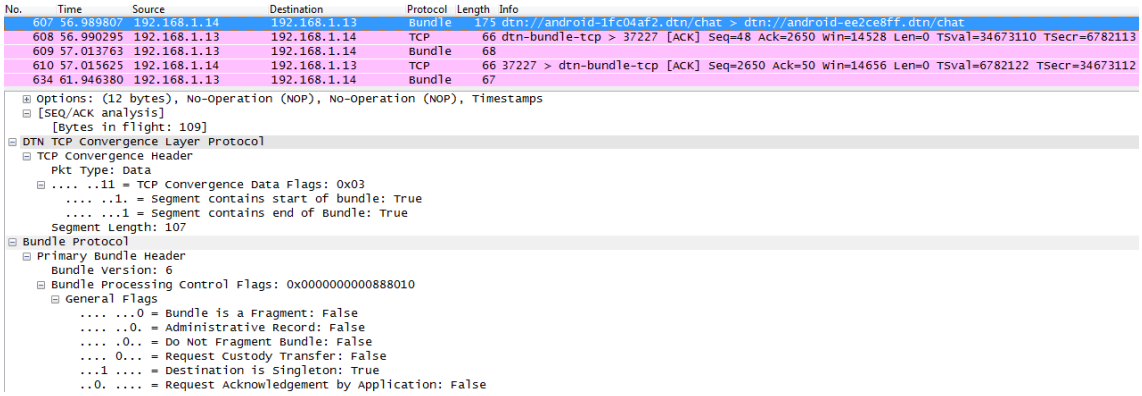

FIGURA 22: MENSAJE DE CHAT: ANSWERING TEST 1

En el detalle de las cabeceras del protocolo *bundle*, representadas en la Figura 23, se pueden observar algunos detalles. La capa de convergencia indica que se transmiten datos y que todo el *bundle* va en un mensaje.

Las cabeceras de protocolo muestran algunos detalles interesantes como que no hay problema con la fragmentación (aunque este *bundle* no es un fragmento). La prioridad es nivel *bulk* (política *best effort*, la más baja).

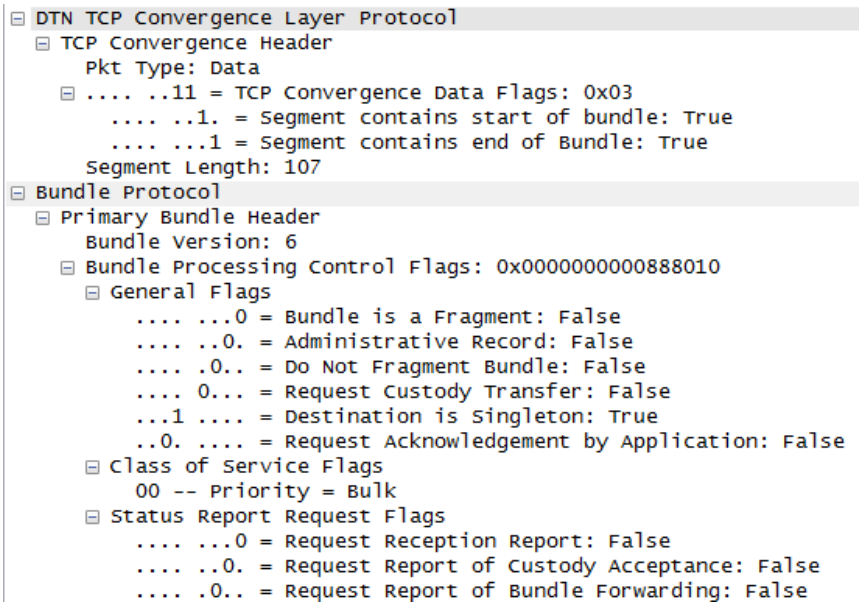

FIGURA 23: MENSAJE (I)

En la Figura 24 se observan las URIs DTN del chat que son las mismas que en el mensaje de *routing* pero terminadas con "chat".

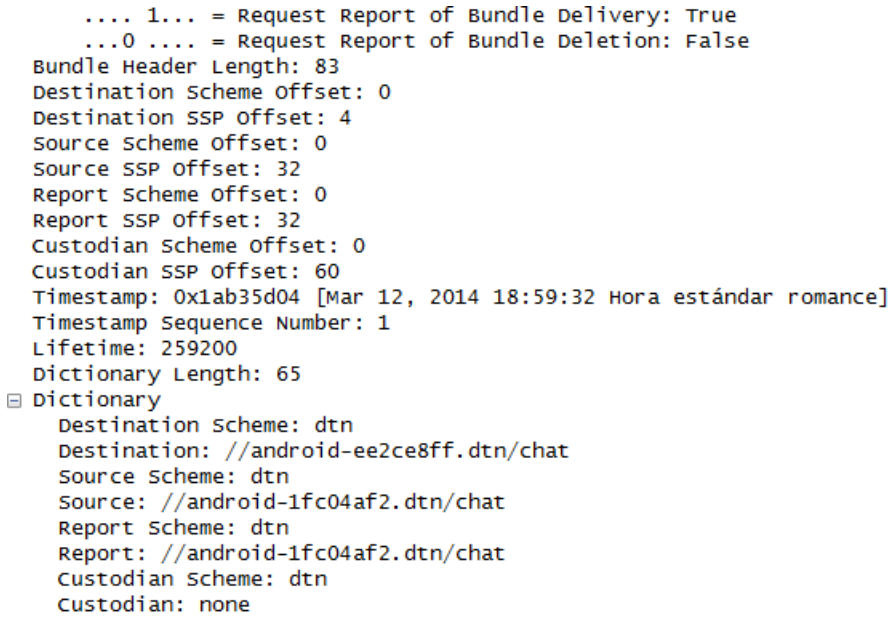

FIGURA 24: MENSAJE (II)

No se realiza *custody transfer* y es el último de los *bundles* a enviar como podemos apreciar en la Figura 25.

```
    Timestamp: 0x1ab35d04 [Mar 12, 2014 18:59:32 Hora estándar romance]
    Timestamp Sequence Number: 1
    Lifetime: 259200
    Dictionary Length: 65
  ⊟ Dictionary
      Destination Scheme: dtn
      Destination: //android-ee2ce8ff.dtn/chat
      Source Scheme: dtn
      Source: //android-1fc04af2.dtn/chat
      Report Scheme: dtn
      Report: //android-1fc04af2.dtn/chat
      Custodian Scheme: dtn
      Custodian: none
⊟ Payload Header
    Header Type: 1
  ⊟ Block Processing Control Flags: 0x08
      .... ...0 = Replicate Block in Every Fragment: False
      .... ..0. = Transmit Status if Block Can't be Processeed: False
      .... .0.. = Delete Bundle if Block Can't be Processeed: False
      .... 1... = Last Block: True
      ...0 .... = Discard Block If Can't Process: False
      ..0. .... = Block Was Forwarded Without Processing: False
      .0.. .... = Block Contains an EID-reference Field: False
    Payload Length: 16
```

FIGURA 25: MENSAJE(III)

3.2.5.9. ACK al Mensaje

Como pasó previamente, el origen recibe un ACK de TCP al *bundle* enviado.

```
No.   Time        Source          Destination     Protocol Length Info
  607 56.98980?   192.168.1.14    192.168.1.13    Bundle   175    dtn://android-1fc04af2.dtn/chat > dtn://android-ee2ce8ff.dtn/chat
  608 56.990295   192.168.1.13    192.168.1.14    TCP      66     dtn-bundle-tcp > 37227 [ACK] Seq=48 Ack=2650 Win=14528 Len=0 TSval=34673110 TSecr=6782113
  609 57.013763   192.168.1.13    192.168.1.14    Bundle   68
  610 57.015625   192.168.1.14    192.168.1.13    TCP      66 37227 > dtn-bundle-tcp [ACK] Seq=2650 Ack=50 Win=14656 Len=0 TSval=6782122 TSecr=34673112
⊞ Frame 608: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
⊞ Ethernet II, Src: SonyEric_6d:52:b7 (30:17:c8:6d:52:b7), Dst: AmpakTec_9f:02:c7 (00:22:f4:9f:02:c7)
⊞ Internet Protocol Version 4, Src: 192.168.1.13 (192.168.1.13), Dst: 192.168.1.14 (192.168.1.14)
⊟ Transmission Control Protocol, Src Port: dtn-bundle-tcp (4556), Dst Port: 37227 (37227), Seq: 48, Ack: 2650, Len: 0
    Source port: dtn-bundle-tcp (4556)
    Destination port: 37227 (37227)
    [Stream index: 1]
    Sequence number: 48    (relative sequence number)
    Acknowledgment number: 2650    (relative ack number)
    Header length: 32 bytes
  ⊞ Flags: 0x010 (ACK)
    Window size value: 227
    [Calculated window size: 14528]
    [Window size scaling factor: 64]
  ⊞ Checksum: 0x47d7 [validation disabled]
  ⊞ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  ⊟ [SEQ/ACK analysis]
      [This is an ACK to the segment in frame: 607]
      [The RTT to ACK the segment was: 0.000488000 seconds]
```

FIGURA 26: TCP ACK AL MENSAJE

3.2.5.10. Bundle ACK

Se recibe además del ACK TCP un ACK de la capa *bundle* como se indica en la Figura 27.

```
No.   Time        Source          Destination     Protocol Length Info
  607 56.98980?   192.168.1.14    192.168.1.13    Bundle   175    dtn://android-1fc04af2.dtn/chat > dtn://android-ee2ce8ff.dtn/chat
  608 56.990295   192.168.1.13    192.168.1.14    TCP      66     dtn-bundle-tcp > 37227 [ACK] Seq=48 Ack=2650 Win=14528 Len=0 TSval=34673110 TSecr=6782113
  609 57.013763   192.168.1.13    192.168.1.14    Bundle   68
  610 57.015625   192.168.1.14    192.168.1.13    TCP      66 37227 > dtn-bundle-tcp [ACK] Seq=2650 Ack=50 Win=14656 Len=0 TSval=6782122 TSecr=34673112
    [Bytes in flight: 2]
⊟ DTN TCP Convergence Layer Protocol
  ⊟ TCP Convergence Header
      Pkt Type: Ack
      Ack Length: 107
```

FIGURA 27: ACK BUNDLE AL MENSAJE

3.2.5.11. TCP ACK al Bundle ACK

Por último, en la Figura 28 se representa un ACK TCP al ACK enviado desde la capa *bundle*.

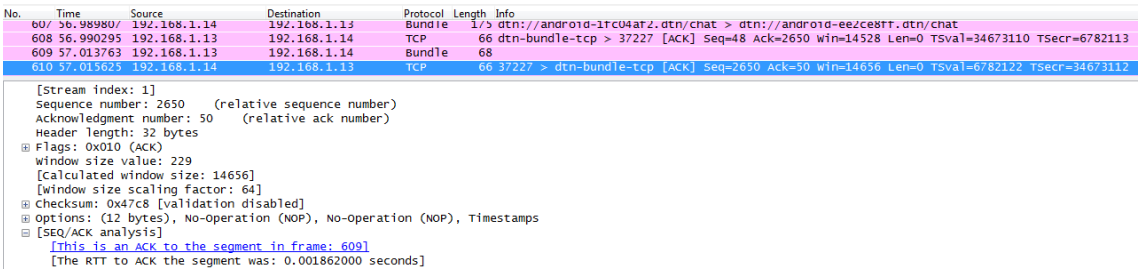

FIGURA 28: TCP ACK AL BUNDLE ACK

3.2.5.12. Parada del Demonio DTN

Para finalizar este pequeño test sobre la aplicación DTN del grupo IBR-DTN se procede al apagado de los demonios DTN. Se observan las tramas típicas TCP del fin de la sesión.

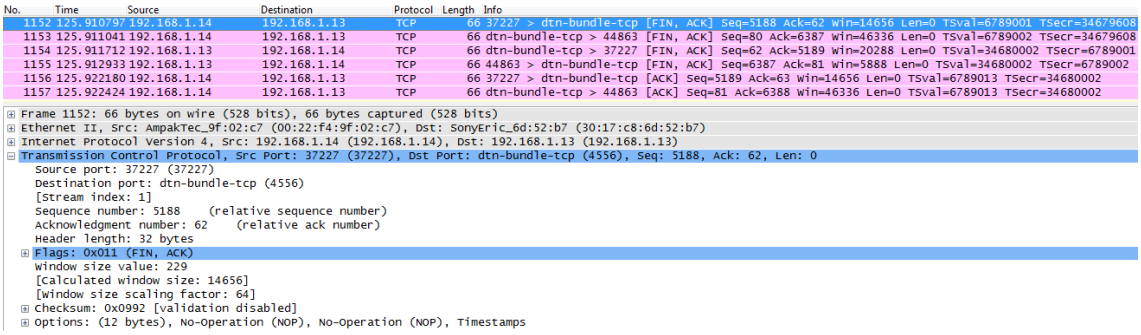

FIGURA 29: PROTOCOLO DE FIN TCP

3.2.6. Conclusiones

La implementación estudiada del protocolo DTN, realizada por el grupo IBR-DTN es válida para cumplir con los objetivos del proyecto. Se ha comprobado que esta aplicación emplea el protocolo DTN para realizar la transmisión y recepción de la información.

Tras el estudio de varias implementaciones de DTN como DTN2, ION-DTN O JDTN [22], se ha decidido que esta es la más completa y la que conlleva una instalación más sencilla para llevar a cabo el proyecto de una aplicación piloto Android que emplee el protocolo de comunicación DTN. Por estos motivos, se ha escogido la implementación de IBR-DTN para llevar a cabo este proyecto.

Capítulo 4: Diseño e Implementación de *Around*

4.1. Introducción

En este capítulo se explicará la aplicación piloto desarrollada en Android para justificar la utilización del protocolo DTN en aplicaciones de monitorización y control de eventos.

La aplicación consiste en un sistema de compartición de información en zonas posiblemente acotadas geográficamente. La aplicación permite compartir una nota informativa, geolocalizada, con el resto de usuarios de la aplicación, incluyendo información adicional como imágenes.

4.2. Diseño e Implementación del Sistema

En este apartado estudiaremos el sistema en sus diferentes partes, así como su integración final que conformará la aplicación Android para el piloto de DTN que estamos desarrollando.

4.2.1. Especificaciones del Sistema

Como paso previo a la implementación de la aplicación Android se establecerán una serie de especificaciones que la aplicación piloto deberá cumplir.

1. La aplicación deberá estar desarrollada para ser utilizada como aplicación Android.
2. La aplicación debe ser capaz de conocer la posición geográfica continua del dispositivo en el que se hospeda.
3. Los usuarios deben poder enviar una serie de notas como puntos de interés sobre las ubicaciones en las que se encuentran o en una ubicación concreta deseada.
4. Los puntos de interés pueden contener una imagen, un título y una descripción breve del mismo.
5. El usuario debe poder delimitar una zona geográfica en la que desea recibir notas.
6. Las notas serán enviadas mediante *bundles* empleando el estándar reflejado en la RFC 5050 [3], y la implementación del grupo IBR-DTN del protocolo.
7. La aplicación será desarrollada para la versión de Android API 19 (4.4 *KitKat*) pero se soportará una versión mínima de Android API 19 (4.1 *Jelly Bean*).
8. La aplicación será compatible con el demonio IBR-DTN en su versión 1.0, dtnd: 1.0.1, *build*: 570f2b9, disponible en Google Play [9].

4.2.2. Interfaz Gráfica del Sistema

La aplicación contará con tres *activities*. La aplicación se abrirá utilizando la `ActivityMain` que contendrá el mapa con las localizaciones de las notas, así como el menú y los botones de navegación de la aplicación.

Desde el menú se podrá acceder a la `AboutActivity` que contiene información acerca de la versión, autor y funcionalidad básica sobre la aplicación.

Al hacer *click* en cualquier parte del mapa o sobre el botón de añadir nota en mi posición actual, aparecerá la tercer *activity*. `AddNoteActivity` se empleará para añadir los diferentes elementos de una nueva nota. A saber, el nombre, la descripción y la imagen asociada.

4.2.3. Diagrama de Clases

En este apartado se muestra un esquema de las diferentes clases de las que se compone la aplicación en la Figura 30. Por orden de importancia las clases son las siguientes:

- **`MainActivity`**: La *activity* principal desde la que se arranca la aplicación. Esta clase tiene comunicación con casi todas las demás clases de la aplicación. En esta clase se genera el *layout* principal, se manda la información a través de DTN y se reciben los *bundles* para representar las notas en el mapa.
- **`Note`**: Esta clase representa una Nota. Los atributos de esta clase son el identificador de usuario, el título, la descripción, las coordenadas (latitud y longitud) y la ubicación de la imagen en el dispositivo. Una Nota se almacena como un objeto JSON.
- **`Database`**: Base de Datos SQLite que almacena las Notas de un usuario. En esta clase es posible crear la Base de Datos, insertar una Nota, eliminar una Nota o recuperar todas las Notas de la Base de Datos.
- **`MapUtils`**: Clase que contiene una serie de métodos auxiliares para trabajar con el mapa de *Google Maps*. Estos métodos se utilizarán para eliminar un marcador, colocarlos en el mapa, hacer zoom en el mapa, añadir una nota nueva e insertarla a la base de datos.
- **`CoordinatesService`**: Servicio empleado para la localización del terminal en el espacio. Se utiliza para obtener las coordenadas geográficas a través del GPS (si está activado) o a través de la conexión de datos.
- **`PingService`**: Servicio necesario para la comunicación DTN de la aplicación. Este servicio es el encargado de enviar los *bundles* al demonio DTN. Además, crea las nuevas sesiones y las elimina.
- **`DTNServiceReceiver`**: Clase utilizada para la recepción de los *bundles* a través del demonio DTN. Es capaz de identificar entre dos tipos de *bundles* recibidos: *Bundles* con información y reportes de estado.
- **`AddNoteActivity`**: Esta *activity* genera el *layout* del informe para recopilar todos los datos necesarios para añadir una nota. Además, se encarga de transmitir esa nota a la `MainActivity`.

- **`AboutActivity`**: *Activity* que contiene un *layout* de texto con información acerca de la aplicación.
- **`FloatingActionButton`**: Esta clase se utiliza para la representación de los botones flotantes de la aplicación. Además, se encarga de la interacción con ellos.

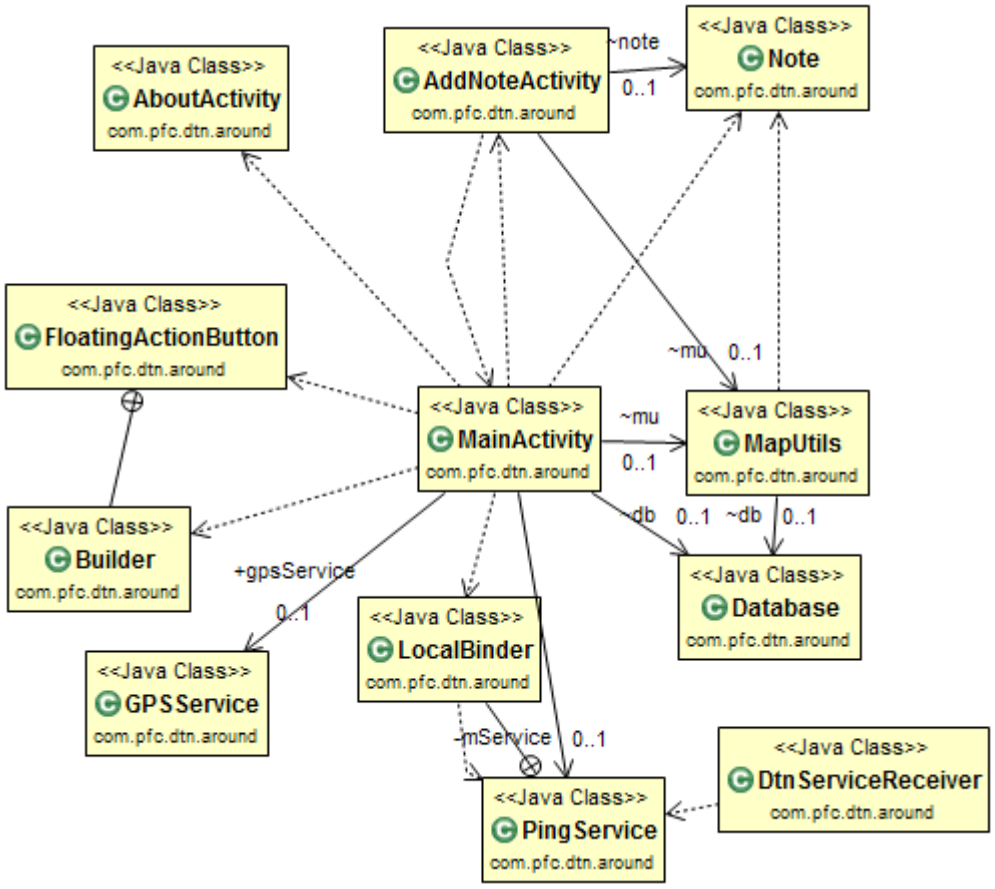

FIGURA 30: DIAGRAMA DE RELACIÓN DE CLASES

4.2.4. Diagramas de Flujo

En este apartado se recogerá el diseño del sistema en función de las diferentes clases especificadas en el apartado anterior.

4.2.4.1. Note

Una nota tiene una serie de atributos importantes, pero los métodos solamente serán para obtener o modificar esos atributos (métodos de *`get`* y *`set`*), además del constructor de la clase `Note`.

Los atributos de esta clase son de los más importantes que hay en la aplicación. Por este motivo se listan a continuación:

- **`userID`**: Identificador único del usuario que creó la nota.
- **`Title`**: Título de la nota.
- **`Description`**: Descripción de la nota.
- **`Coords`**: Coordenadas en formato `LatLng` de la nota.

- **`imagePath`**: Ruta de directorios del dispositivo en el que se encuentra la imagen asociada a la nota, si existe.

4.2.4.2. MainActivity

Clase de la *activity* principal de la aplicación.

Método de Creación de Activity (onCreate())

El método principal de esta clase será el método para crear la *activity*. El método `onCreate()`, además de crear la *activity* dibujará el layout, obteniendo para ello todos los elementos necesarios. Este método generará el mapa y lo rellenará con los marcadores existentes (en caso de tener notas almacenadas previamente). Ver la Figura 31 para más detalles.

En los marcadores del mapa se colocarán además *listeners* que generarán un evento cuando se hace click en un marcador. En este caso mostrarán un *infowindow* con el título y la descripción de la nota insertada. En estos *infowindows* también se añadirán *listeners* para poder eliminar las notas. Al hacer click en la *infowindow* aparecerá un *popup* que nos permitirá eliminar el marcador y la nota asociada a él.

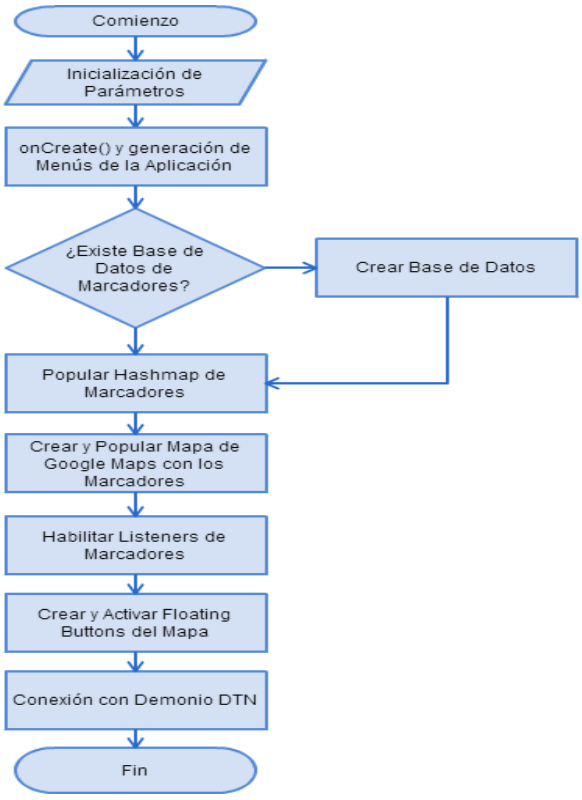

FIGURA 31: DIAGRAMA DE FLUJO DE CREACIÓN DE MAINACTIVITY

En este método se generarán también los diferentes botones de acción para los mapas. Estos botones tendrán asociadas al hacer click la acción de añadir una nueva nota en la localización del dispositivo, ajustar la pantalla para ver todas las notas en el mapa y ocultar las notas del mapa.

Por último, se establecerá la conexión con el demonio DTN (IBR-DTN Master) para permitir enviar y recibir *bundles* empleando el RFC 5050.

Método onMapClick()

Este método se encarga de lanzar una nueva *activity* `AddNoteActivity` para añadir una nueva activity. Esta activity se arranca empleando un *Intent* y empleando el método `startActivityForResult()` que invocará al método `onActivityResult()` con los datos de la nueva Nota, para realizar su inserción. Detallado en la Figura 32.

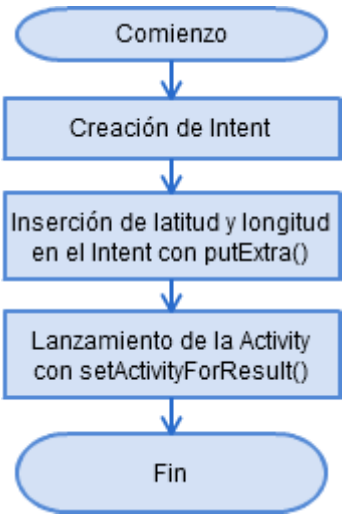

FIGURA 32: DIAGRAMA DE FLUJO DE ONMAPCLICK()

Método de Alerta de Configuración GPS (showSettingsAlert())

El principal objetivo de este método será crear un *alert dialog* que nos invoque a la pantalla de configuración si el GPS no está activado. Si el GPS Está activado este *alert dialog* no se mostrará. En este método se configuran las acciones de los botones de aceptar y cancelar, explicado en el diagrama de flujo de la Figura 33.

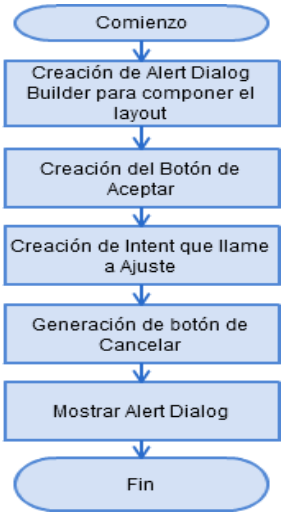

FIGURA 33: DIAGRAMA DE FLUJO DE ALERTA DE CONFIGURACIÓN GPS

Método de Borrado de Base de Datos (cleanDatabase())

Con la ayuda de este método, se invoca a la base de datos para eliminarla completamente. Para confirmar esta acción se muestra un *alert dialog* que nos notifica el evento y nos pide la aceptación del borrado. Mostrado con detalles en la Figura 34.

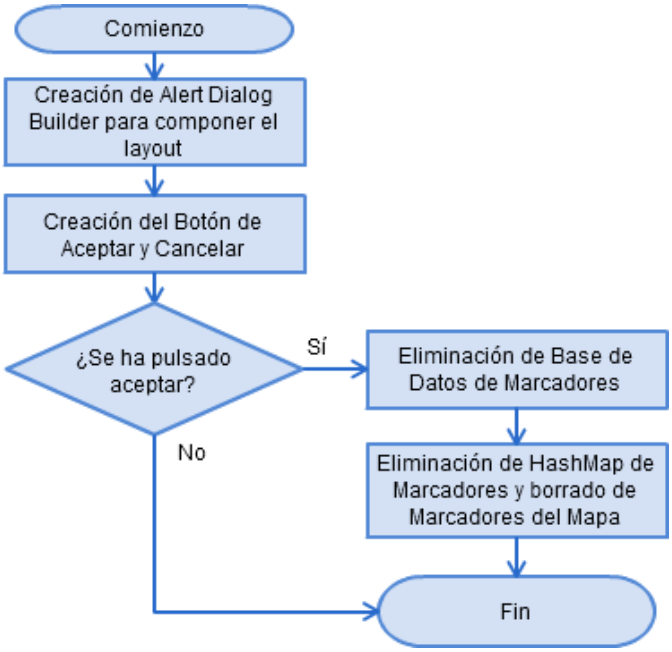

FIGURA 34: DIAGRAMA DE FLUJO DE CLEANDATABASE()

Método de Establecimiento de Conexión DTN serviceConnection()

Establece la conexión del servicio DTN. Utiliza un *binder* local para conectarse al servicio de la aplicación IBR-DTN. También se utiliza para la desconexión del servicio como se especifica en la Figura 35.

FIGURA 35: DIAGRAMA DE FUJO DE SERVICECONNECTION()

Método onDestroy()

Empleamos el método `onDestroy()` para desconectar el servicio DTN en caso de que la aplicación se cierre, y el servicio esté abierto, representado en la Figura 36.

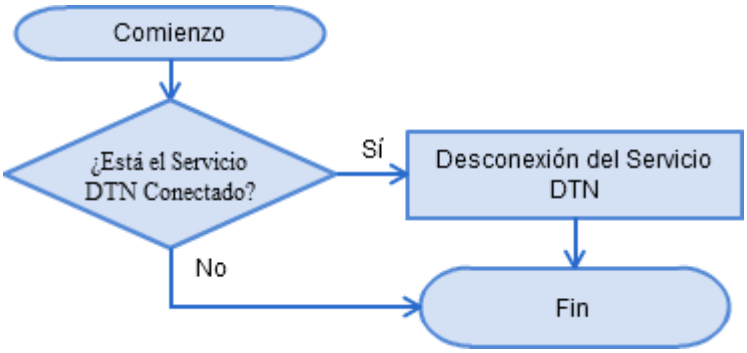

FIGURA 36: DIAGRAMA DE FLUJO DEL MÉTODO ONDESTROY()

Método onResume()

Este método se ejecuta cuando la aplicación vuelve a abrirse desde el segundo plano. En este método se recoge el restablecimiento de la conexión DTN como se muestra en la Figura 37.

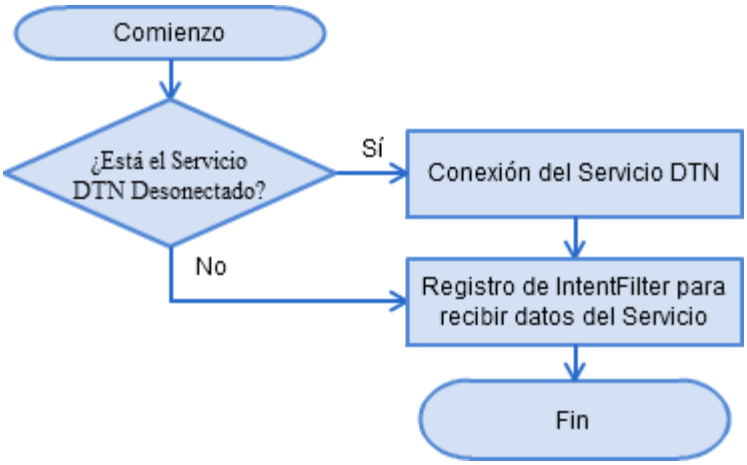

FIGURA 37: DIAGRAMA DE FLUJO DEL MÉTODO ONRESUME()

Método BroadcastReceiver()

Este método recogerá los *intents* recibidos que contengan un *bundle* DTN. En este método se comprobará si el *intent* contiene un S*tring* extra denominado *payload*. Este *String* contendrá toda la información y la imagen (si existe) de la nota recibida. Tras comprobar si el *intent* contiene una nota, es necesario antes de almacenarla saber si esa nota se encuentra dentro del radio deseado para la recepción. Si es así esta nota se almacena en un objeto de tipo `JSONObject` para almacenarlo en la base de datos y se añade el nuevo marcador en el mapa. El detalle se encuentra en la Figura 38.

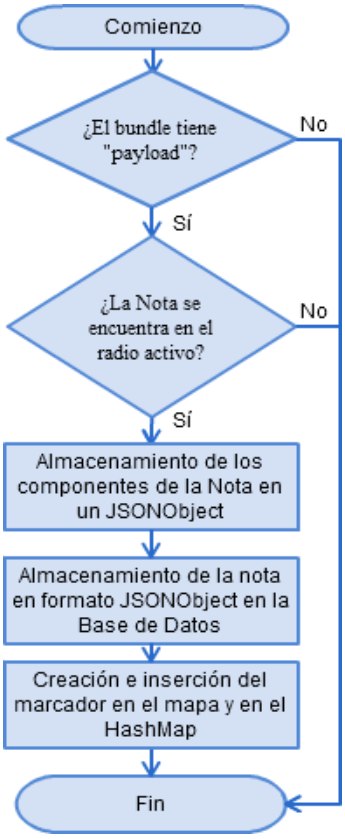

FIGURA 38: DIAGRAMA DE FLUJO DEL MÉTODO BROADCASTRECEIVER()

Método onActivityResult()

En la Figura 39 se encuentra explicado el método método encargado de realizar el procesado del resultado de los *intents* enviados previamente para arrancar una *activity* empleando el método `startActivityForResult()`. En este caso, se recogerá el resultado tras realizar la petición de añadir una nota nueva. Si es una nota válida se almacenará en la base de datos y se representará en el mapa como ocurría en el caso de la recepción de una nota en un *bundle* DTN. Por último, se envía la nota mediante un *intent* al demonio DTN para que la transmita por broadcast a los vecinos disponibles.

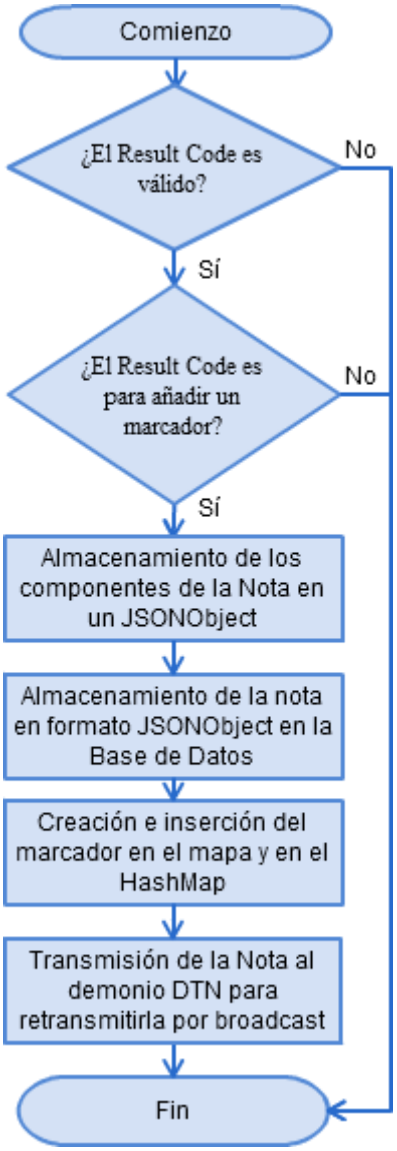

FIGURA 39: DIAGRAMA DE FLUJO DEL MÉTODO ONACTIVITYRESULT()

Método Ping()

Este método se emplea para transmitir la Nota como un *payload* de un *bundle* DTN. En este método se crea un intent y se le añade la Nota para transmitir el *bundle* DTN a través del máster DTN del grupo IBR-DTN. El *intent* se emplea para arrancar el servicio y enviar al EID de la aplicación (`dtn://broadcast.dtn/around`) la Nota en el *bundle* creado para ello. Se puede observar el procedimiento en la Figura 40.

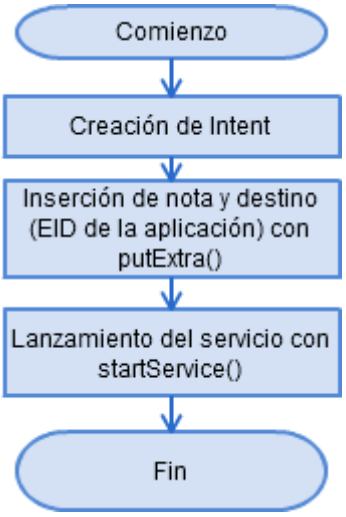

FIGURA 40: DIAGRAMA DE FLUJO DEL MÉTODO PING()

4.2.4.3. DataBase

Base de datos que contendrá una tabla con todas las Notas del usuario.

Método Constructor (Database())

Este método crea la base de datos SQLite denominada `NotesDatabase.db` en el dispositivo como se muestra en la Figura 41.

FIGURA 41: DIAGRAMA DE FLUJO DEL MÉTODO CONSTRUCTOR DATABASE()

Método onCreate()

En este método se declara la estructura de la tabla de marcadores. Para nuestra tabla se usará una clave compuesta por un *timestamp* de inserción y el nombre de usuario, el id de usuario de la Nota, el *timestamp*, las coordenadas, el título, la

descripción y la ubicación de la imagen. Con estos parámetros se lanza una *query* para crear una tabla en la base de datos como es indicado en la Figura 42.

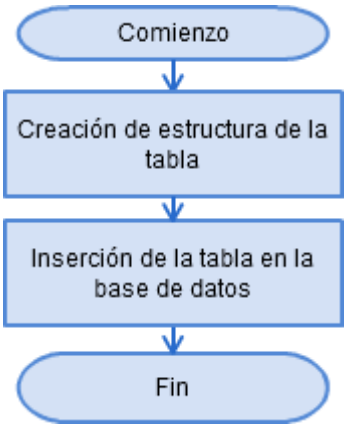

FIGURA 42: DIAGRAMA DE FLUJO DEL MÉTODO ONCREATE()

Método insertData()

Este método recibe una nota y la inserta dentro de la base de datos. Para ello se utiliza un *content value* en el que se almacenan los datos a insertar. Después, se emplea el método `insert` de la librería SQLite para insertar la nueva nota en la base de datos como se indica en la Figura 43.

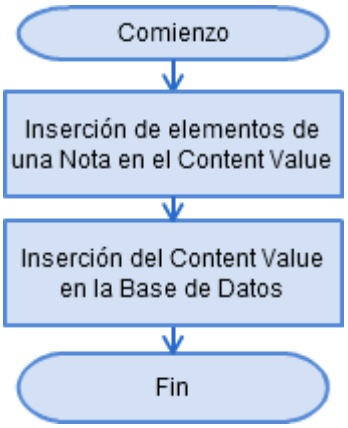

FIGURA 43: DIAGRAMA DE FLUJO DEL MÉTODO INSERTDATA()

Método fetchData()

En este método se recorre toda la estructura de la tabla, utilizando los *cursores*. Se van almacenando todas las Notas recogidas de la tabla en un *Array*. Se representan más detalles en la Figura 44.

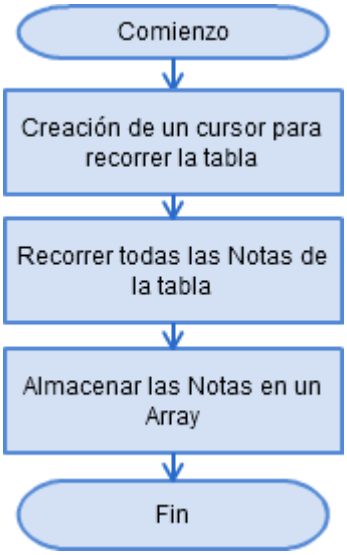

FIGURA 44: DIAGRAMA DE FLUJO DEL MÉTODO FETCHDATA()

Método removeElement()

Para eliminar un elemento de la tabla se emplea el método representado en la Figura 45. Para ello se realiza una *query* a la tabla eliminando el elemento seleccionado.

FIGURA 45: DIAGRAMA DE FLUJO DEL MÉTODO REMOVEELEMENT()

4.2.4.4. CoordinatesService

Servicio GPS que se encarga de actualizar en segundo plano las coordenadas geográficas en las que se encuentra ubicado el dispositivo. Implementa `LocationListener` para poder realizar este seguimiento.

Método Constructor CoordinatesService()

Este método crea el *Location Manager* y obtiene la localización actual del dispositivo como se muestra con más detalle en el diagrama de la Figura 46.

FIGURA 46: DIAGRAMA DE FLUJO DE COORDINATESSERVICE()

Método getLocation()

Con el método `getLocation()` se obtienen las coordenadas geográficas del dispositivo. Se emplea *Criteria* que decide cuál de los dos elementos del dispositivo que sirven para obtener coordenadas (conexión a red o conexión GPS) es la más apropiada en función de la activación y la cobertura. Con ello se obtienen las coordenadas actuales como se muestra en el esquema de la Figura 47.

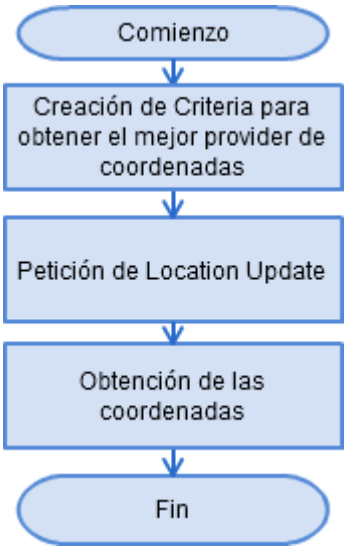

FIGURA 47: DIAGRAMA DE FLUJO DEL MÉTODO GETLOCATION()

4.2.4.5. MapUtils

La clase `MapUtils` contiene métodos auxiliares para interactuar con el mapa y los marcadores de *Google Maps*.

Método onInfoWindowClicked()

Cuando un usuario pulsa en una *infowindow* de los marcadores, se permite la posibilidad de eliminar ese marcador. Se desplegará un *alert dialog* que solicitará al usuario la confirmación para eliminar la Nota del mapa. Se representan los detalles en la Figura 48.

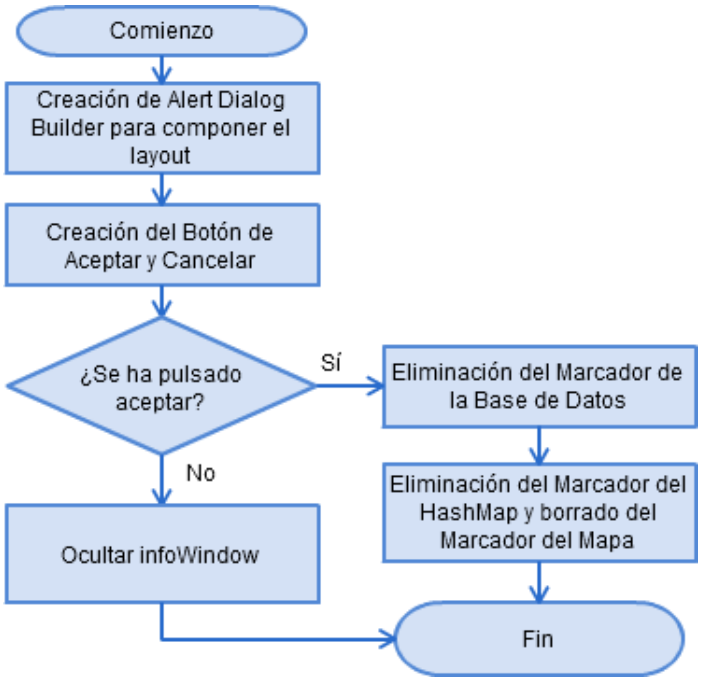

FIGURA 48: DIAGRAMA DE FLUJO DEL MÉTODO ONINFOWINDOWCLICKED()

Método removeAllMarkers()

Empleando el método `removeAllMarkers()` se consiguen eliminar todas las notas del *Hashmap*, de la base de datos y del mapa como se indica en la Figura 49.

FIGURA 49: DIAGRAMA DE FLUJO DEL MÉTODO REMOVEALLMARKERS()

Método populateMap()

Con la ayuda de este método se rellenan el Hashmap de marcadores y se representan las Notas en el mapa. Los datos son obtenidos de la base de datos de marcadores, al iniciar la aplicación. En el diagrama de la Figura 50 se muestra una representación del método.

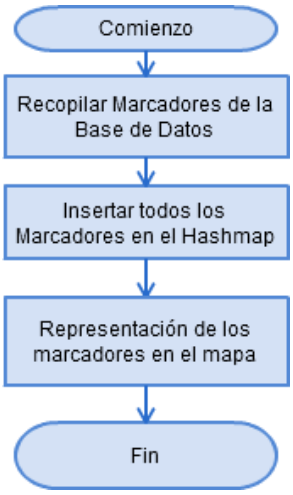

FIGURA 50: DIAGRAMA DE FLUJO DEL MÉTODO POPULATEMAP()

Método putMarkerOnMap()

Se usa este método para colocar una nota nueva en el mapa como se detalla en la Figura 51. El marcador empleado para ser representado en el mapa será la imagen (si dispone de ella) o un marcador que representa si la nota es del usuario (rojo) o ha sido recibida por otro (verde).

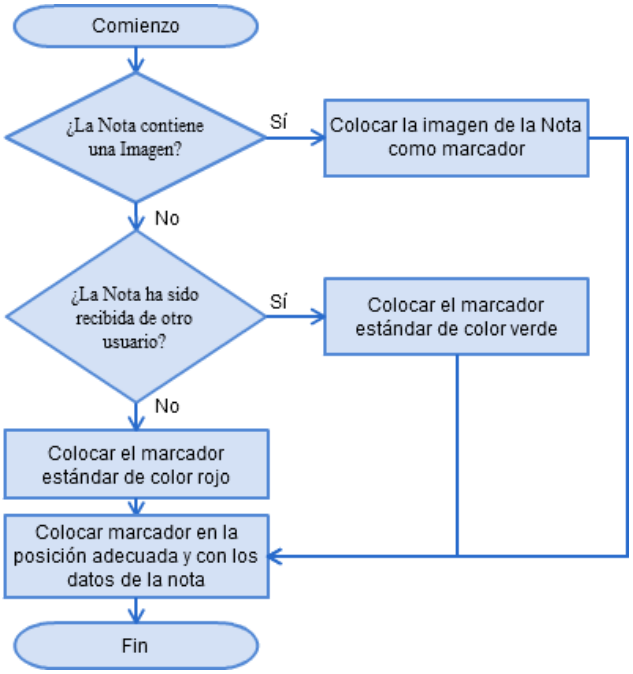

FIGURA 51: DIAGRAMA DE FLUJO DEL MÉTODO PUTMARKERONMAP()

4.2.4.6. AddNoteActivity

Esta clase genera una *activity* para añadir una nueva Nota.

Método onCreate()

Este método crea la nueva *activity*. En esta *activity* se crea un *layout* con dos entradas de texto una para insertar el título de la Nota (que es obligatorio) y otra para insertar una descripción opcional. Además, se añaden dos botones para insertar una imagen (desde la galería o tomando una nueva foto desde la cámara). Para obtener la imagen se lanza el *intent* adecuado para ello. Tras tener la imagen, cuando se pulsa aceptar, se comprueba que se haya insertado un título para la imagen y si es así se arranca un *intent* con los datos de la nueva Nota para volver a la *activity* principal y verla representada en el mapa. Se detalla en el esquema de la Figura 52.

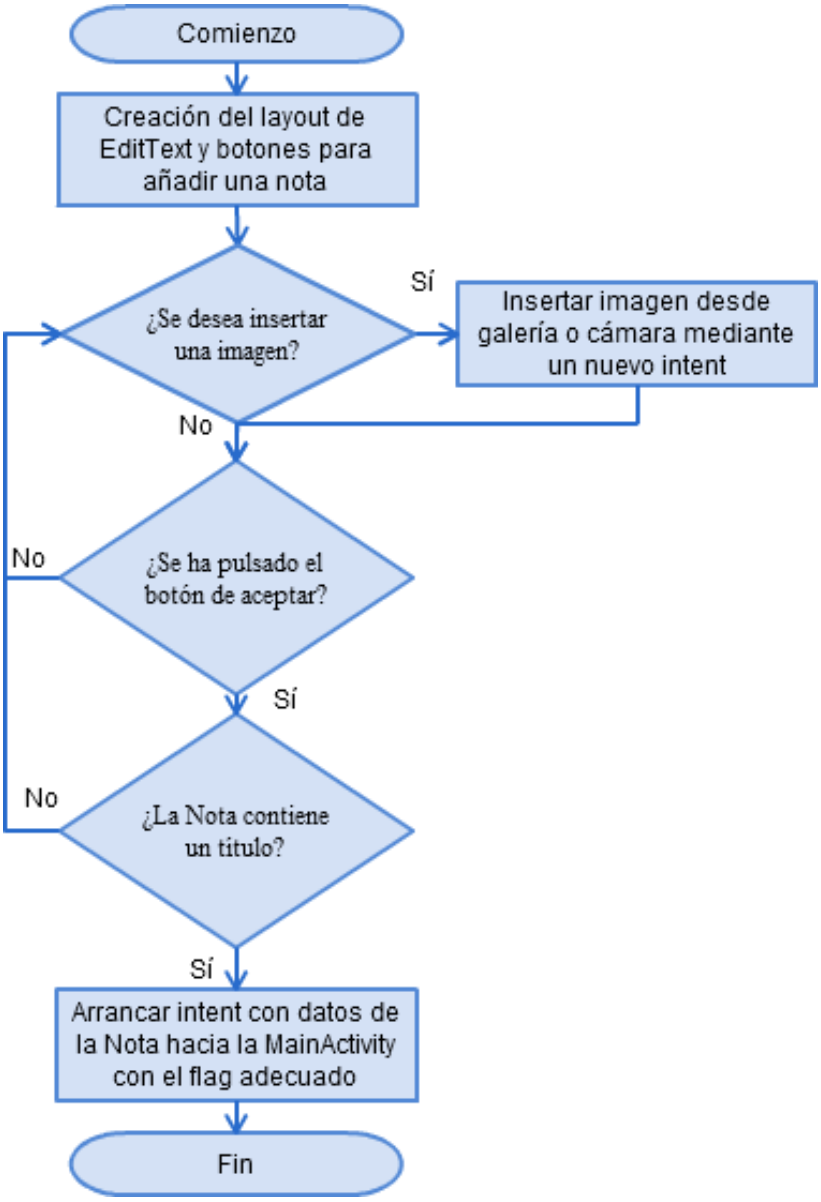

FIGURA 52: DIAGRAMA DE FLUJO DEL MÉTODO ONCREATE()

Método onActivityResult()

Este método se emplea para obtener la imagen como resultado del lanzamiento del *intent* para capturar una imagen con la cámara o recogerla de la galería de imágenes del usuario. Esta imagen se representará en una *ImageView* en la *activity*. Además, con ayuda de su URI, se obtiene el directorio de la imagen en el dispositivo para almacenarlo en la Nota como se indica en la Figura 53.

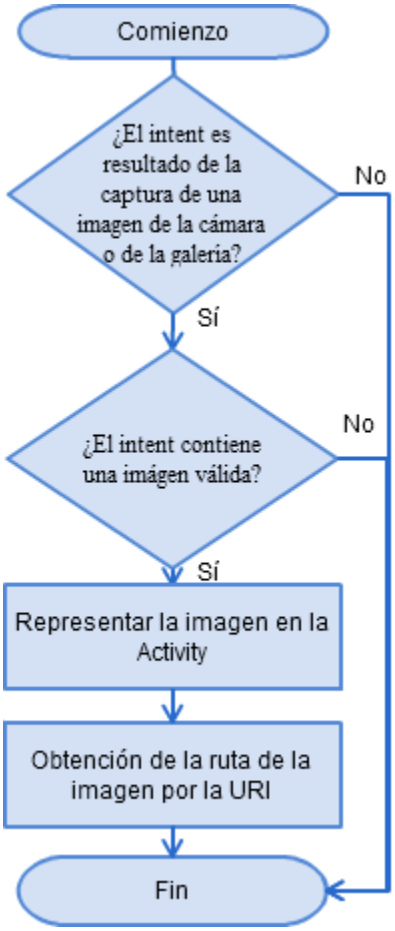

FIGURA 53: DIAGRAMA DE FLUJO DE ONACTIVITYRESULT()

4.2.4.7. DTNServiceReceiver

La clase `DTNServiceReceiver` extiende a `BroadcastReceiver` para recoger los *intents* enviados por otras aplicaciones. En este caso se procesarán los *intents* recuperados del demonio IBR-DTN.

Método onReceive()

Este método procesa los *intents* recibidos y comprueba si son de alguno de los dos tipos soportados para esta aplicación DTN. Un *intent* puede ser un *bundle* recibido o un *Status Report*. Tras recuperar el intent, se lanza hacia el servicio de *Ping*, que procesará el contenido del *intent*, indicando a este servicio el tipo de *intent* recibido (*bundle* o

status report). Se incluirá todo el contenido del *intent* al arrancar el servicio de ping como se detalla en la Figura 54.

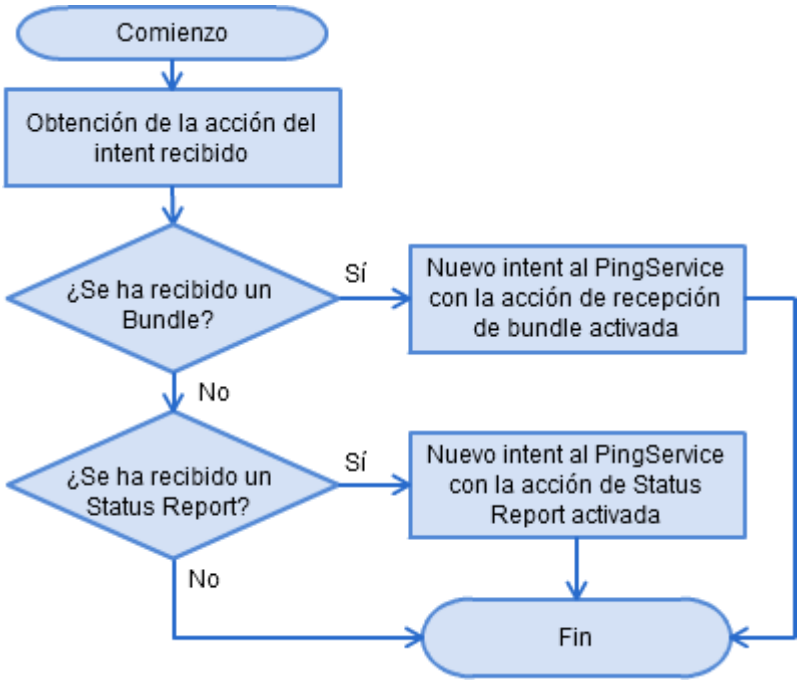

FIGURA 54: DIAGRAMA DE FLUJO DEL MÉTODO ONRECEIVE()

4.2.4.8. PingService

Este servicio procesará los *bundles* DTN recibidos mediante el `DTNServiceReceiver`. Uno de los atributos principales de esta clase es el EID de la aplicación (`dtn://broadcast.dtn/around`). A continuación se detallan los métodos más relevantes de esta clase.

Método doPing()

En este método se enviará el b*undle*. Tras crearlo, se le añaden todos los parámetros como el destino (*broadcast*), tiempo de vida, petición de confirmación de recepción activada y la Nota como *payload*. Se muestra el diagrama de flujo de este método en la Figura 55.

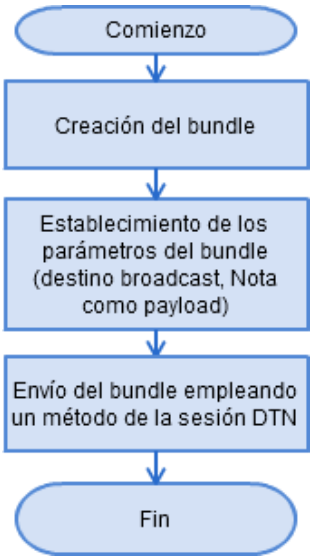

FIGURA 55: DIAGRAMA DE FLUJO DEL MÉTODO DOPING()

Método onHandleIntent()

En este método se realiza la acción tras haber recibido uno de los tres tipos de *intents*: petición de envío de un *bundle*, recepción de un *bundle*, recepción de *status report*.

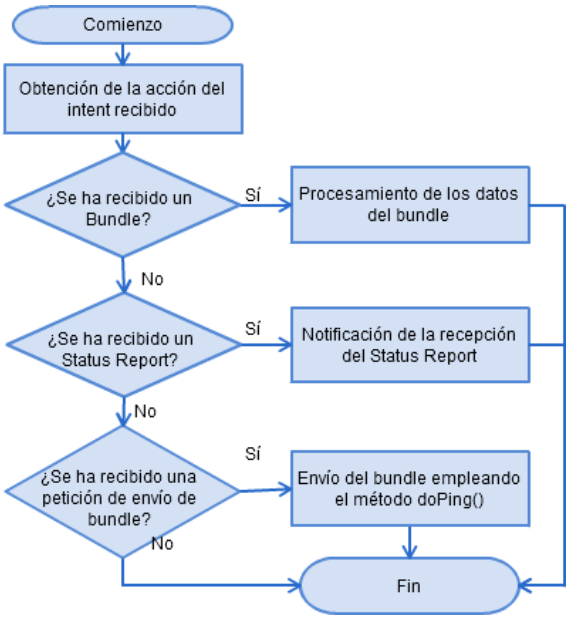

FIGURA 56: DIAGRAMA DE FLUJO DEL MÉTODO ONHANDLEINTENT()

4.2.4.9. AboutActivity

Esta clase contiene la *activity* que representa información acerca de la aplicación.

Método onCreate()

El método `onCreate()` genera el *layout* de la *activity* que contiene información acerca de la aplicación con la versión y el autor, representado en la Figura 57.

FIGURA 57: DIAGRAMA DE FLUJO DEL MÉTODO ONCREATE()

4.3. Pruebas

En este apartado se mostrarán una serie de pruebas sencillas empleando capturas de pantalla de la aplicación, que mostrarán los diferentes menús y elementos interactivos para poder enviar y recibir Notas mediante el uso del protocolo DTN. En el siguiente enlace [18] se encuentra un video demostrativo de algunas de las pruebas realizadas y detalladas en este apartado.

4.3.1. Envío de una Nota

En esta sección detallaremos paso a paso lo necesario para enviar una nota empleando la aplicación *Around*.

Al abrir la aplicación, si el GPS no está activado se muestra en la pantalla un mensaje que permite al usuario acceder a los ajustes de su dispositivo para activar el servicio GPS. La aplicación pude obtener las coordenadas del dispositivo mediante dos métodos: empleando el GPS o mediante la red con la que el dispositivo se encuentre conectado como se muestra en la Figura 58.

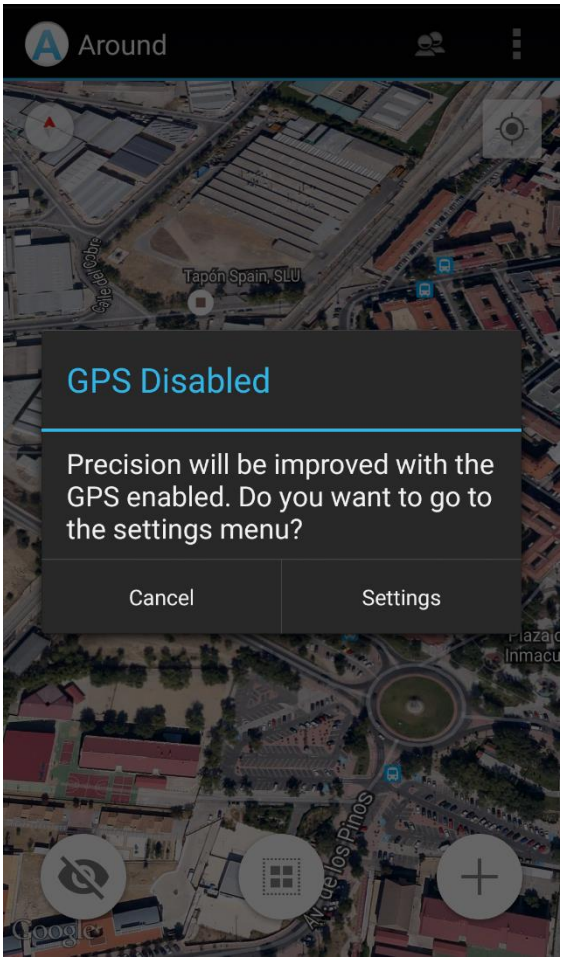

FIGURA 58: INICIO DE LA APLICACIÓN CON EL GPS DESHABILITADO

Después de cancelar o activar el servicio GPS en el dispositivo, la aplicación mostrará la localización del usuario, representándolo como un punto azul en el plano y los marcadores cercanos a ese punto, observándose el detalle en la Figura 59.

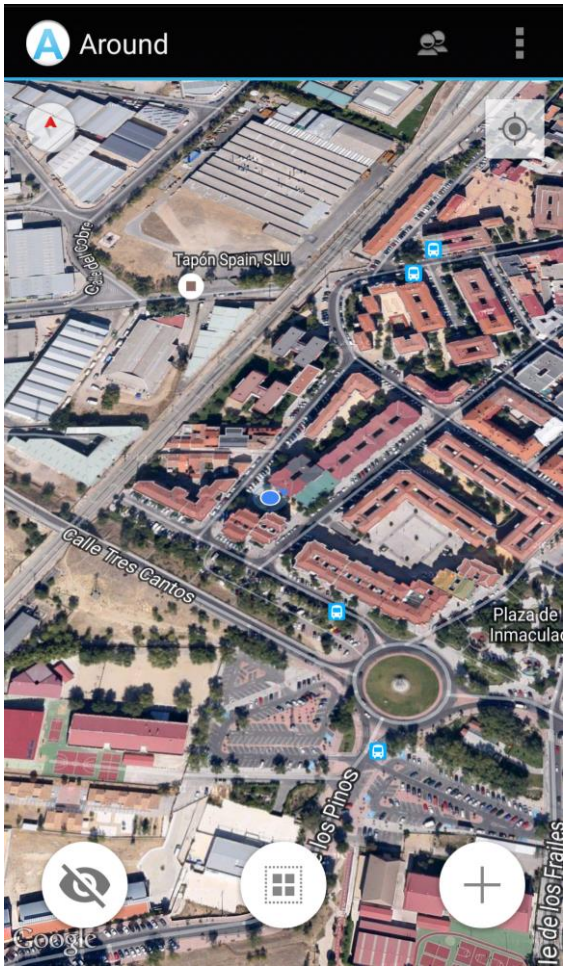

FIGURA 59: PANTALLA INICIAL DE LA APLICACIÓN

Para crear un marcador en la localización actual es preciso pulsar el botón flotante que aparece en la Figura 60.

FIGURA 60: BOTÓN PARA AÑADIR UNA NOTA EN LA POSICIÓN ACTUAL

Tras pulsar el botón, aparece una nueva *activity* que nos lleva a un formulario que debemos rellenar para insertar una nueva Nota.

Los campos textuales a rellenar son el título de la Nota (que es obligatorio) y la descripción.

Además, aparece una *imageView* donde aparecerá la imagen que carguemos para ser asociada a la Nota. Para añadir esta imagen se emplean los dos botones de la parte inferior. El botón de la derecha nos abrirá la galería para seleccionar la imagen a cargar. El botón de la izquierda abrirá la cámara fotográfica para obtener una nueva imagen. Esta *activity* se encuentra representada en la Figura 61.

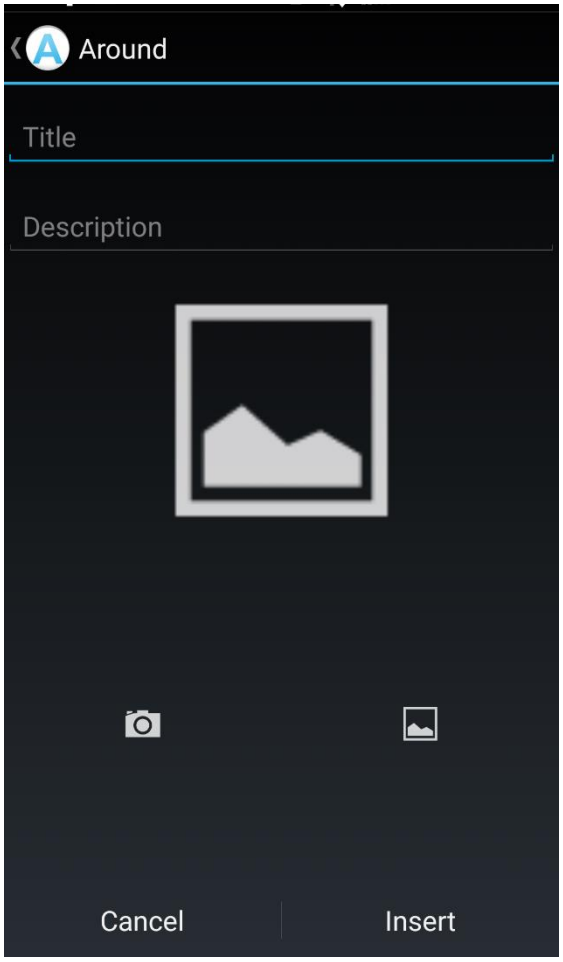

FIGURA 61: FORMULARIO DE INSERCIÓN DE UNA NOTA

Tras rellenar el título y la descripción, insertamos la nueva nota con los detalles que aparecen en la Figura 62.

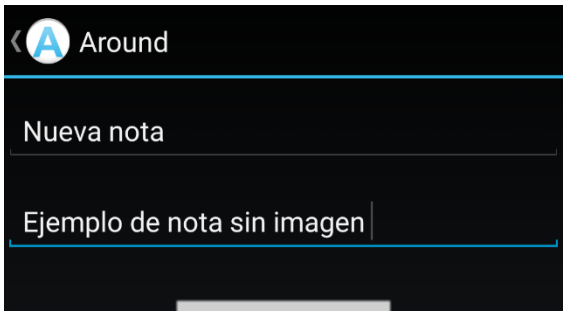

FIGURA 62: ELEMENTOS DE LA NOTA SIN IMAGEN

La nota aparece en el mapa con un nuevo marcador representado en la Figura 63.

FIGURA 63: EJEMPLO DE NOTA SIN IMAGEN

También es posible seleccionar cualquier punto del mapa para añadir una nueva Nota. En este caso, vamos a incluir una Nota con una imagen obtenida de la galería del usuario en una localización cercana a la localización del usuario como se muestra en la Figura 64.

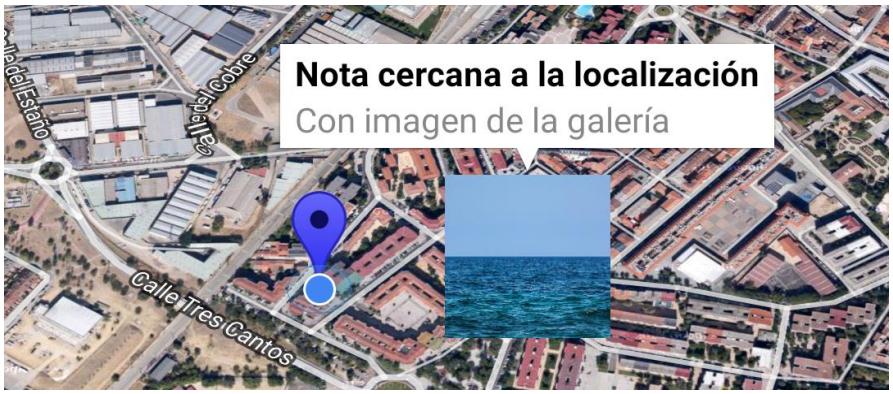

FIGURA 64: EJEMPLO DE NOTA EN POSICIÓN CERCANA A LA LOCALIZACIÓN

En este apartado vamos a añadir unas capturas tomadas haciendo un *sniffing* de la red, para ver una traza DTN enviada por la aplicación empleando el protocolo *bundle*. En la Figura 65 aparece la captura de un *bundle*, donde se muestra el envío entre el *bundle* desde un dispositivo de forma *broadcast*, empleando la aplicación *Around*. En la parte inferior, se observa el *payload* del *bundle* que contiene una Nota. Se puede apreciar en la captura de pantalla, que se recogen los parámetros que se envían en una nota: *timestamp*, título, nombre de usuario, coordenadas, imagen y descripción. Para este caso concreto, los campos rellenos en la nota son el timestamp, el título (prueba), el nombre de usuario (dtn://android-f46b5454.dtn), las coordenadas (50.771,19.549).

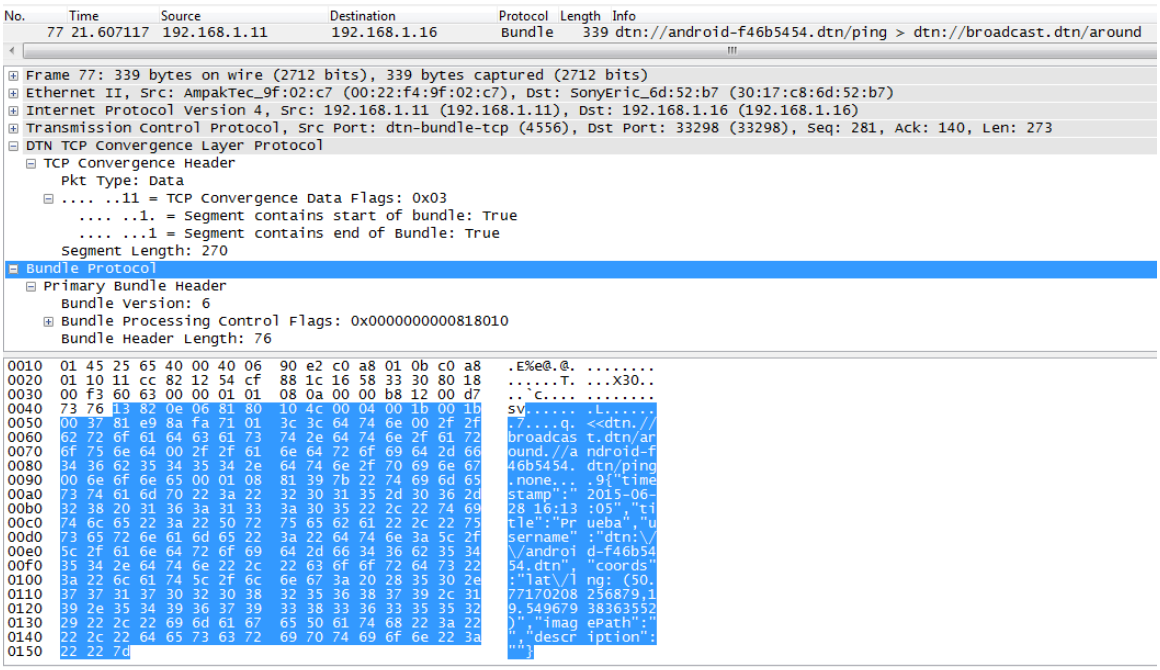

FIGURA 65: CAPTURA DEL BUNDLE ENVIADO

En la Figura 66 se detallan algunos campos del *bundle*, siendo relevante destacar que no es un *bundle* fragmentado. La clase de prioridad es *bulk* (la más baja, siguiendo una política de *best effort*).

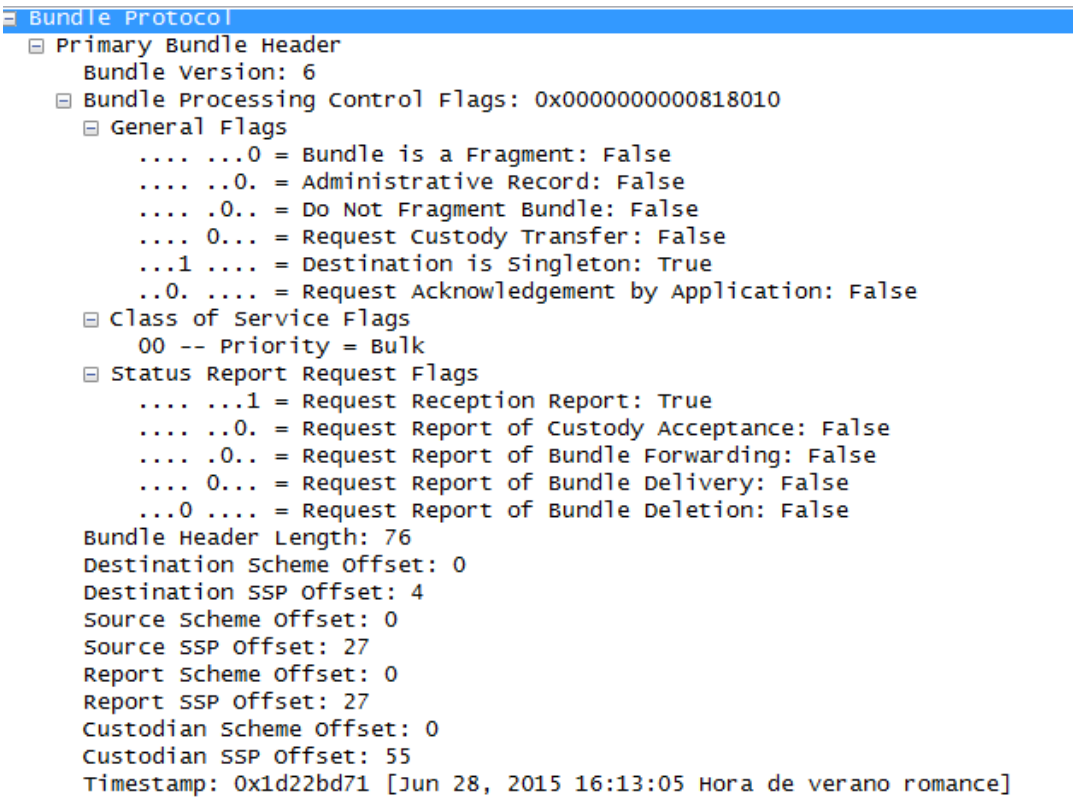

FIGURA 66: DETALLE DE LA TRAZA DTN AROUND (I)

En la Figura 67 aparecen el resto de campos del *bundle* como el origen y el destino, así como el estado de los *flags* de control. Se indica que es el último bloque del *bundle*.

```
    Timestamp Sequence Number: 1
    Lifetime: 60
    Dictionary Length: 60
  ⊟ Dictionary
      Destination Scheme: dtn
      Destination: //broadcast.dtn/around
      Source Scheme: dtn
      Source: //android-f46b5454.dtn/ping
      Report Scheme: dtn
      Report: //android-f46b5454.dtn/ping
      Custodian Scheme: dtn
      Custodian: none
⊟ Payload Header
    Header Type: 1
  ⊟ Block Processing Control Flags: 0x08
      .... ...0 = Replicate Block in Every Fragment: False
      .... ..0. = Transmit Status if Block Can't be Processeed: False
      .... .0.. = Delete Bundle if Block Can't be Processeed: False
      .... 1... = Last Block: True
      ...0 .... = Discard Block If Can't Process: False
      ..0. .... = Block Was Forwarded Without Processing: False
      .0.. .... = Block Contains an EID-reference Field: False
    Payload Length: 185
```

FIGURA 67: DETALLE DE LA TRAZA DTN AROUND (II)

4.3.2. Eliminación de una Nota

Al hacer *click* en el *infowindow* de la Figura 68, se despliega un *alert dialog* que nos pide confirmación de la eliminación.

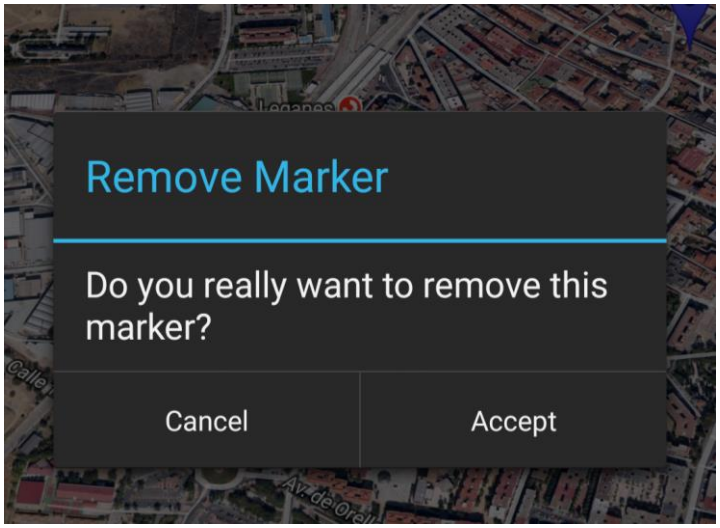

FIGURA 68: ELIMINACIÓN DE UNA NOTA

Tras hacer *click* en aceptar, la nota desaparece del mapa y se elimina de la base de datos tal y como aparece en la Figura 69.

FIGURA 69: RESULTADO TRAS LA ELIMINACIÓN DE LA NOTA

4.3.3. Recepción de una Nota

En este caso realizaremos el envío y la recepción de una Nota entre dos dispositivos. El dispositivo que enviará la nota será una *tablet*, que enviará una Nota sin ninguna imagen. En la siguiente imagen puede observarse la Nota insertada en el primer dispositivo. Puede observarse en la Figura 70 que el marcador representado es de color azul.

FIGURA 70: CAPUTA DE PANTALLA DE UNA NOTA ENVIADA

La Figura 71 corresponde a un una imagen tomada de un s*martphone*, que es el dispositivo que recibe la nota enviada. Se puede observar en la imagen que el color del

marcador recibido es verde, mientras que el resto de Notas, propias del dispositivo, se encuentran representadas de color azul.

FIGURA 71: CAPTURA DE PANTALLA DE UNA NOTA RECIBIDA

También se puede enviar una Nota con imagen como aparece en la Figura 72.

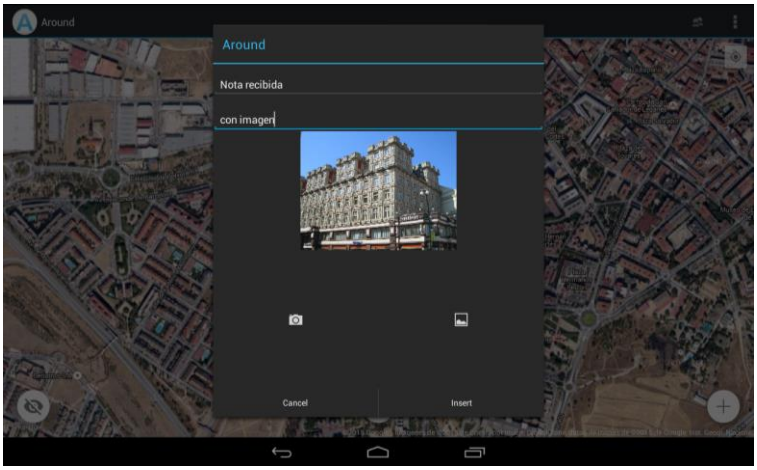

FIGURA 72: CAPTURA DE PANTALLA DE INSERCIÓN DE NOTA CON IMAGEN

En el dispositivo original aparece la Nota enviada con la imagen agregada correspondiente, como se ve en la Figura 73.

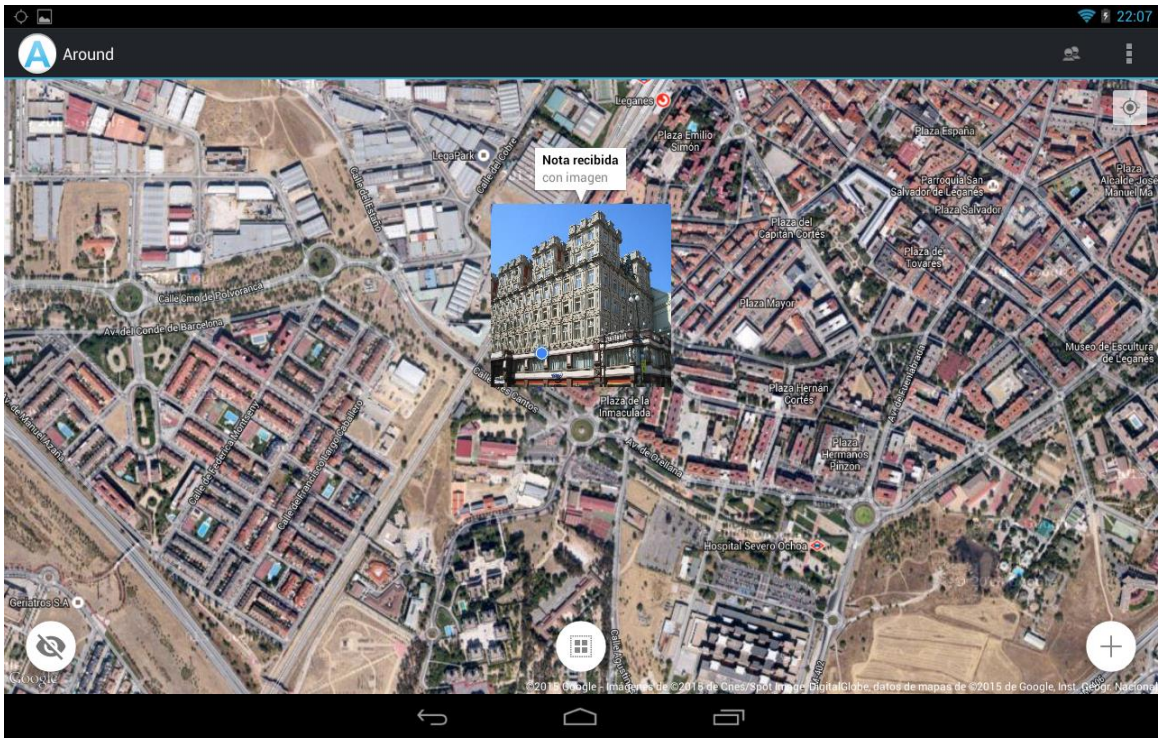

FIGURA 73: CAPTURA DE PANTALLA DE NOTA ENVIADA CON IMAGEN

Tras unos instantes, se puede observar en la Figura 74 la imagen recibida junto a la Nota recibida previamente con el marcador verde.

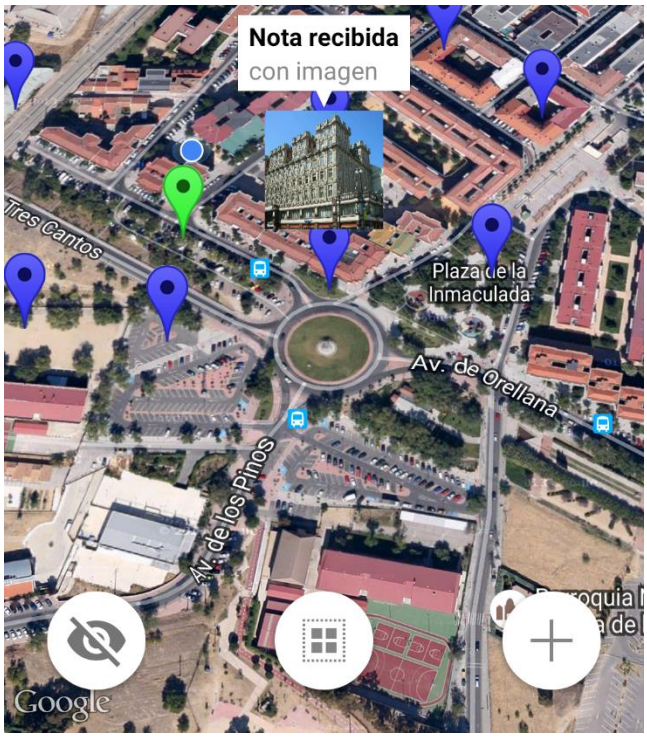

FIGURA 74: CAPTURA DE PANTALLA DE UNA NOTA RECIBIDA CON IMAGEN

4.3.4. Radio de Recepción

La aplicación permite seleccionar el radio (en metros) en el que se permiten recibir notas. Estos ajustes están disponibles en el menú de la aplicación, representado en la Figura 75.

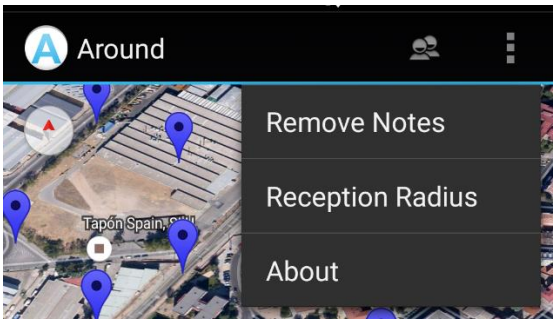

FIGURA 75: MENÚ DE LA APLICACIÓN. RADIO DE RECEPCIÓN

Al seleccionar la opción de radio de recepción en el menú de ajustes, se abre un nuevo *Alert Dialog* que aparece en la Figura 76 para ajustar el radio en metros en el que se desean recibir notas, desde la ubicación actual del dispositivo. Si se selecciona un "0", implica que se desean recibir notas desde cualquier ubicación.

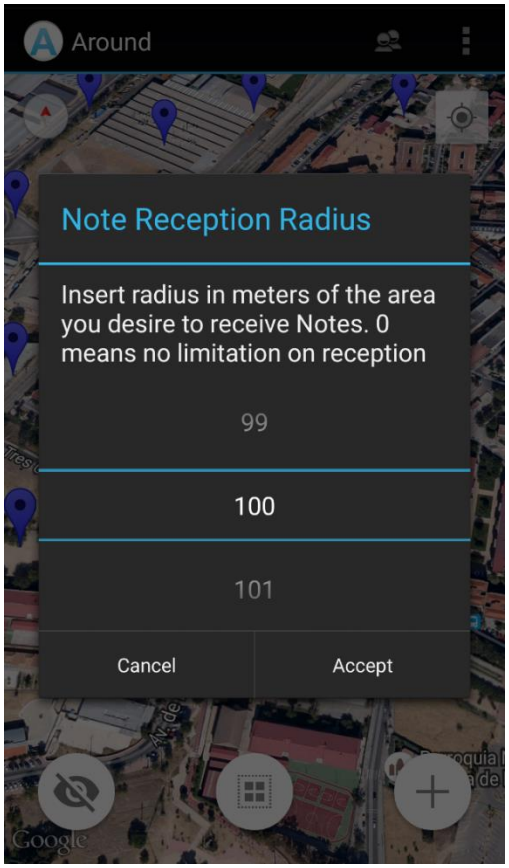

FIGURA 76: AJUSTE DE RADIO DE RECEPCIÓN

Ahora cuando se envía una Nota desde una distancia mayor de 100 metros desde la ubicación del dispositivo, esta nota será descartada por el dispositivo y no se guardará.

Capítulo 5: Conclusiones y Trabajos Futuros

5.1. Historia del Proyecto

Este proyecto nació en 2014, como siguientes pasos, tras la realización de un estudio tecnológico comparando las dos versiones de Android disponibles entonces en el mercado, Android 2 y Android 4.

Tras finalizar ese estudio, la Directora del Proyecto propuso la realización de una aplicación Android que empleara un protocolo de comunicaciones para redes tolerantes al retardo, DTN. Al finalizar el estudio de este protocolo y de las varias implementaciones disponibles, se decidió utilizar la implementación del grupo IBR-DTN para llevar a fin este proyecto. En mayo de 2015 se finalizó el desarrollo de la aplicación, y en Julio de 2015, tras la documentación, se realiza la presentación y defensa del proyecto.

5.2. Conclusiones y Trabajos Futuros

En este proyecto se han aprendido las diferentes funcionalidades y partes de un protocolo de comunicación, como es DTN. Además, se ha visto cómo se lleva a cabo la implementación de un nuevo protocolo y cómo es posible realizar aplicaciones que lo utilicen para llevar a cabo sus comunicaciones.

La aplicación Android desarrollada es un piloto que puede evolucionarse enormemente en el futuro. Se encuentra una gran utilidad en aplicaciones de este tipo para entornos reducidos, donde los usuarios no dispongan de conexión a internet, pero que sus dispositivos puedan interconectarse entre ellos.

Para llevar a cabo este proyecto, se han encontrado varias dificultades. La principal es la falta de material y documentación para realizar nuevas aplicaciones Android empleando IBR-DTN. Aunque disponen de una amplia wiki con contenidos y otros ejemplos, no existe un API como tal que especifique para qué se pueden emplear los diferentes métodos del demonio de IBR-DTN, que habría ayudado a sacar un mayor partido y haber conseguido una implementación de mayor funcionalidad. Todo el aprendizaje de la programación, configuración e instalación de las aplicaciones DTN ha sido otra dificultad por el mismo motivo.

Las líneas básicas de evolución de la aplicación pueden dividirse en dos, la parte DTN y la parte Android.

Por una parte, como línea futura de la aplicación, sería interesante dar soporte a diferentes elementos que quedan fuera de los objetivos de este proyecto, como son temas de calidad de servicio y seguridad. También deberían incluirse en las líneas futuras de la parte DTN del proyecto, el tratamiento de todos los tipos de mensajes que pueden ser recibidos desde el demonio IBR-DTN.

Por otro lado, en la implementación Android, sería interesante adaptar la aplicación a la nueva versión del sistema operativo, Android 5. También es posible añadir funcionalidad a la aplicación de forma sencilla, por la forma en la que ha sido programada. Es aconsejable que las notas enviadas por la aplicación puedan contener elementos que fueron sacados de los objetivos de este proyecto, como son las notas de audio. Además, los diferentes comportamientos gráficos de la aplicación podrían ser adaptados a diferentes idiomas.

Como conclusión final, el trabajo realizado durante estos meses con el protocolo DTN y la aplicación Android desarrollada, me llevan a ver en DTN un protocolo de futuro, cuya implementación está infrautilizada por el estado inicial en que se encuentra. Si se dedican recursos de investigación y no se abandona el desarrollo del protocolo, parece un buen método para resolver los grandes problemas de las comunicaciones con retardos que existen en la actualidad.

Anexo I: Presupuesto

Introducción

En este apartado se comentarán las diferentes fases de las que se compone este proyecto. Se incluirá un diagrama de *Gantt* y un presupuesto final sobre este proyecto.

Fases del Proyecto

El proyecto se dividió en cuatro fases principales:

La *Fase I* consiste en la recapitulación de información acerca del protocolo DTN, del desarrollo en Android y de la implementación de la aplicación IBR-DTN empleada para realizar las comunicaciones utilizando el protocolo *bundle*.

La *Fase II se basa en el* análisis de la información y desarrollo de la idea principal para la implementación de una aplicación piloto en Android para demostrar la utilidad del protocolo DTN.

En la *Fase III* se especifica e implementa la aplicación Android, se habla del desarrollo e instalación del entorno y se realizan las pruebas de la aplicación.

Por último, en la *Fase IV* se realiza la documentación del proyecto, la defensa del trabajo realizado y de los resultados obtenidos.

En el próximo apartado se especifica en un diagrama de *Gantt* las diferentes fases y la duración que tuvieron a lo largo de la realización de este proyecto.

Diagrama de Gantt

En este apartado se detallan las tareas de las cuatro fases del proyecto en un diagrama de *Gantt*, representado en la Figura 77 y la Figura 78.

La planificación del proyecto se inicia el día 3 de Febrero de 2014 y finaliza con el día estimado de defensa y presentación del proyecto ante el tribunal, el día 15 de Julio de 2015.

La Fase I consta de 30 días, y está finalizada por completo. Esta fase se llevó a cabo en los tiempos indicados, arrancando el 3 de Febrero de 2014 y finalizando el 17 de Marzo de 2014.

La segunda fase consta de 180 días de estudio de los diferentes elementos recuperados con la información necesaria para el proyecto. Esta fase comienza el 24 de marzo de 2014 y acaba el 1 de diciembre de 2014. La fase se finalizó al 100% en los tiempos indicados.

Para la tercera fase fueron necesarios 105 días de diseño, desarrollo, implementación y pruebas de la aplicación Android *Around*. Esta fase dio comienzo el 1 de diciembre de 2014 y finalizó el 27 de abril de 2014 al 100%.

La fase de documentación del proyecto, que consta de la estructuración y realización de la presentación, conlleva 57 días y comienza el 27 de Abril y finaliza el 15 de julio de 2015. La memoria se encuentra realizada al 100% pero la presentación se encuentra en proceso, con previsión de estar terminada el 15 de Julio.

Por último, como fecha estimada de presentación y defensa final, se encuentra el 15 de Julio de 2015.

	Nombre	Duración	Inicio	Terminado
1	**Diseño del Proyecto**	**30 days**	**3/02/14 9:00**	**17/03/14 9:00**
2	Recapitulación de Información de DTN	15 days	3/02/14 9:00	24/02/14 9:00
3	Recapitulación de Información de Android	15 days	3/02/14 9:00	24/02/14 9:00
4	Obtención de Información de implementaciones DTN para Android	30 days	3/02/14 9:00	17/03/14 9:00
5	**Análisis de la Información**	**180 days**	**24/03/14 9:00**	**1/12/14 9:00**
6	Estudio de la programación de aplicaciones en Android	50 days	24/03/14 9:00	2/06/14 9:00
7	Estudio del protocolo DTN	50 days	2/06/14 9:00	11/08/14 9:00
8	Estudio y pruebas de la implementación IBR-DTN en Android	80 days	11/08/14 9:00	1/12/14 9:00
9	**Desarrollo de la Aplicación Around**	**105 days**	**1/12/14 9:00**	**27/04/15 9:00**
10	Especificación de los requisitos	5 days	1/12/14 9:00	8/12/14 9:00
11	Diseño de la aplicación	15 days	8/12/14 9:00	29/12/14 9:00
12	Implementación de la aplicación	70 days	29/12/14 9:00	6/04/15 9:00
13	Pruebas de la aplicación	15 days	6/04/15 9:00	27/04/15 9:00
14	**Documentación del Proyecto**	**57 days**	**27/04/15 9:00**	**15/07/15 9:00**
15	Estructuración y realización de la memoria	50 days	27/04/15 9:00	6/07/15 9:00
16	Realización de la presentación	7 days	6/07/15 9:00	15/07/15 9:00
17	Presentación y Defensa Final	1 day	15/07/15 8:00	15/07/15 17:00

FIGURA 77: FASES DEL PROYECTO

Se especifica en la Figura 78 el diagrama de *Gantt*.

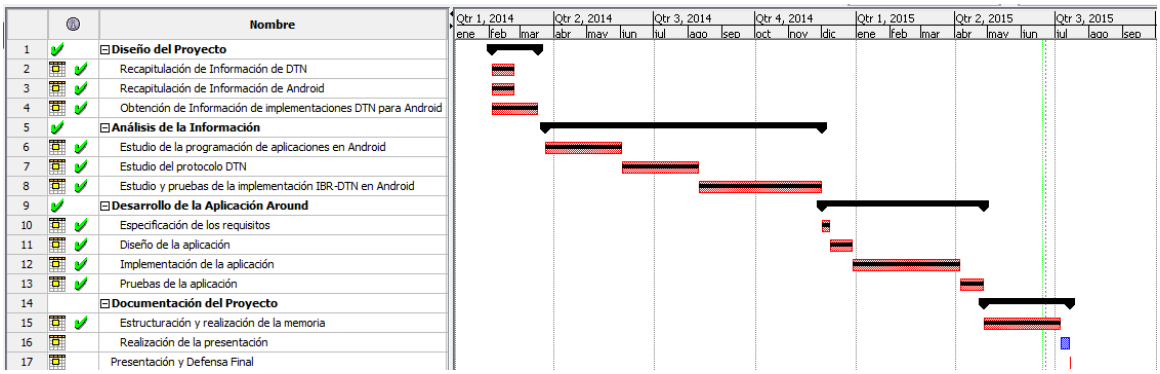

FIGURA 78: DIAGRAMA DE GANTT

Desglose de Costes

El coste total del proyecto asciende a 44.490 € que están divididos en los costes del personal, los costes de los equipos y otros costes indirectos.

En la siguiente tabla se detallan el número de horas totales empleadas por el personal que ha llevado a cabo este proyecto. Se estiman unas 5 horas de media diarias para la realización del proyecto.

TABLA 1: DETALLE DEL NÚMERO DE HORAS SEGÚN LAS FASES

Fase	Número de Horas
Fase I	150
Fase II	900
Fase III	525
Fase IV	285
Total	1860

El personal a cargo del proyecto se detalla a continuación, estimando el número de horas de cada uno empleadas, así como el coste de la hora para cada tipo de puesto. El coste de la hora se ha establecido a 20 € para el ingeniero junior y a 50€ para el ingeniero senior. El ingeniero junior se corresponde con el ingeniero proyectista, y el ingeniero senior se corresponde con la dirección del proyecto.

TABLA 2: DESGLOSE DE GASTOS DE PERSONAL

Apellidos y Nombre	Categoría	Dedicación en Horas	Coste Hora	Coste imputable
de la Casa Riballo, Daniel	Ingeniero Junior	1660	20,00 €	33.200,00 €
Campo Vázquez, Celeste	Ingeniero Sénior	200	50,00 €	10.000,00 €
			Total:	€ 43.200,00

En la siguiente tabla se desglosa el gasto en los equipos. Serán necesarios un equipo hardware para el desarrollo de la aplicación, así como un par de dispositivos móviles para la implementación y las pruebas. Los gastos de estos equipos son estimados.

TABLA 3: DESGLOSE DE GASTOS DE EQUIPOS

Descripción	Coste	% de uso dedicado	Coste imputable
PC para desarrollo	800,00 €	80	640,00 €
2 Dispositivos móviles	900,00 €	50	450,00 €
		Total:	1.090,00 €

Se añade por último una serie de costes varios como los desplazamientos a la universidad, dietas, gasolina, material de oficina, por un valor total estimado de 200€.

TABLA 4: RESUMEN DE LOS COSTES

Concepto	Coste Imputable
Personal	43.200,00 €
Equipos	1.090,00 €
Otros	200,00 €
Total	44.490,00 €

El presupuesto total de este proyecto asciende a la cantidad de 44.490 EUROS.

Leganés a 15 de Julio de 2015

El ingeniero proyectista

Fdo. Daniel de la Casa Riballo

Anexo II: Manual De Usuario

En este apartado se describirán brevemente las funcionalidades de las diferentes *activities* de la aplicación para permitir a los usuarios la navegación por la aplicación.

En la *activity* principal se encuentra el mapa que incluye las diferentes Notas del usuario. Aparecen las notas con imágenes, las notas con marcadores (azules del usuario, verdes recibidas y rojas para Notas que tuvieron imagen pero ésta ya no existe) y los botones inferiores de acción. El botón superior derecho del mapa centra al usuario en su localización actual como se detalla en la Figura 79.

FIGURA 79: IMAGEN DE LA ACTIVITY PRINCIPAL

El menú principal está compuesto por 4 elementos. Tres de ellos están escondidos pero pueden desplegarse en un submenú representado en la Figura 80.

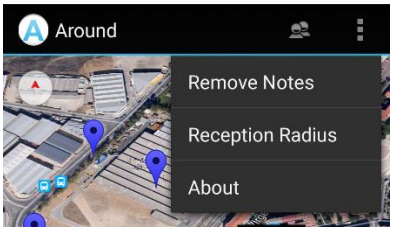

FIGURA 80: MENÚ DE LA APLICACIÓN

El primer elemento del menú, que se representa con un icono, nos indica al pulsarlo los vecinos DTN disponibles, como aparece en la Figura 81.

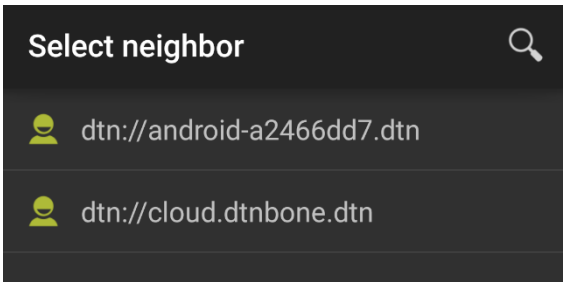

FIGURA 81: VECINOS DTN DISPONIBLES

El segundo elemento, *Remove Notes*, nos permite eliminar todas las notas del mapa. Despliega un *popup* que nos avisa de esta acción, mostrado en la Figura 82.

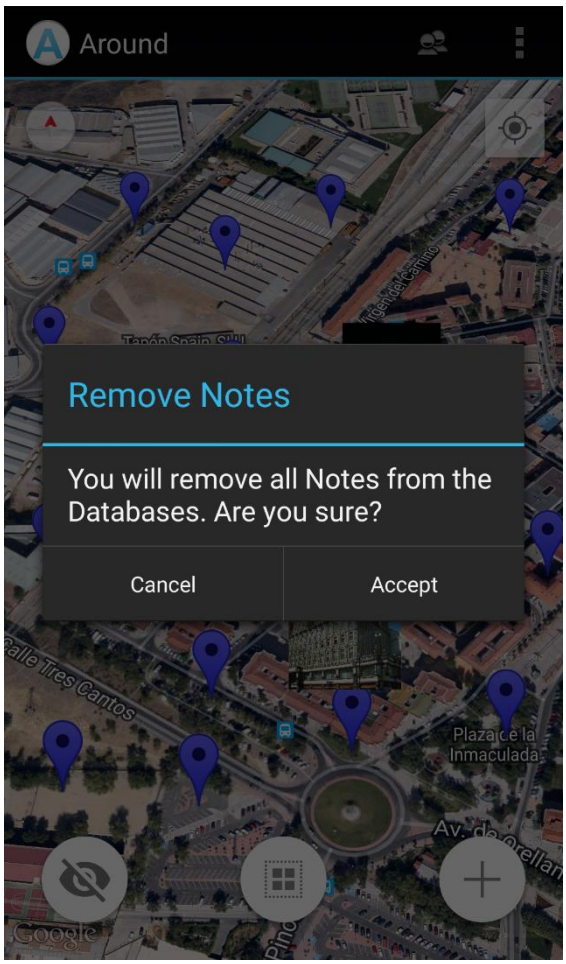

FIGURA 82: AVISO ANTES DE ELIMINAR TODAS LAS NOTAS

Con el tercer elemento, se permite al usuario modificar el radio de alcance en el cual desea recibir notas. Este radio está descrito en metros, y supondrá la distancia máxima respecto a la localización del usuario a la que éste desea recibir Notas a través de DTN del resto de los usuarios de la aplicación. Si se especifica a 0 este valor, indica que el usuario no desea restringir el acceso de Notas DTN desde ninguna localización en

todo el mundo, como se muestra en la Figura 83. Si se desea restringir este radio, el valor máximo soportado es de 10000 metros (10 km).

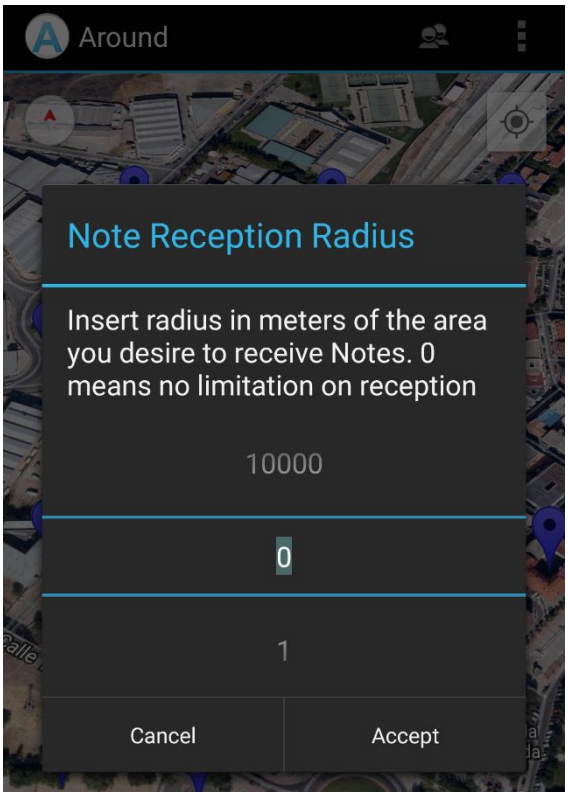

FIGURA 83: ESTABLECIMIENTO DEL RADIO DE RECEPCIÓN DE NOTAS

Por último, al pulsar en About, aparece información básica sobre la aplicación, indicada en la Figura 84.

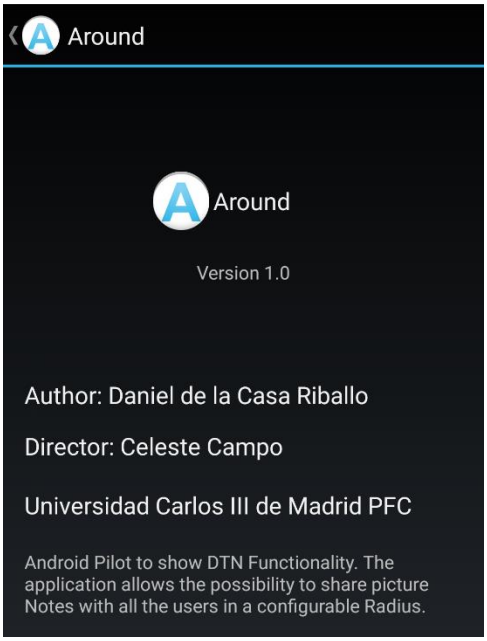

FIGURA 84: PÁGINA DE INFORMACIÓN DE LA APLICACIÓN

En la parte superior del mapa, aparecen tres botones flotantes, detallados en la Figura 85.

FIGURA 85: BOTONES FLOTANTES DE LA APLICACIÓN

El de la derecha, permite añadir una nueva Nota en la posición actual del usuario. Además de al pulsar este botón, al pulsar cualquier otra posición en el mapa, se lanzará una nueva *activity* que permitirá añadir una nueva nota, que se muestra en la Figura 86.

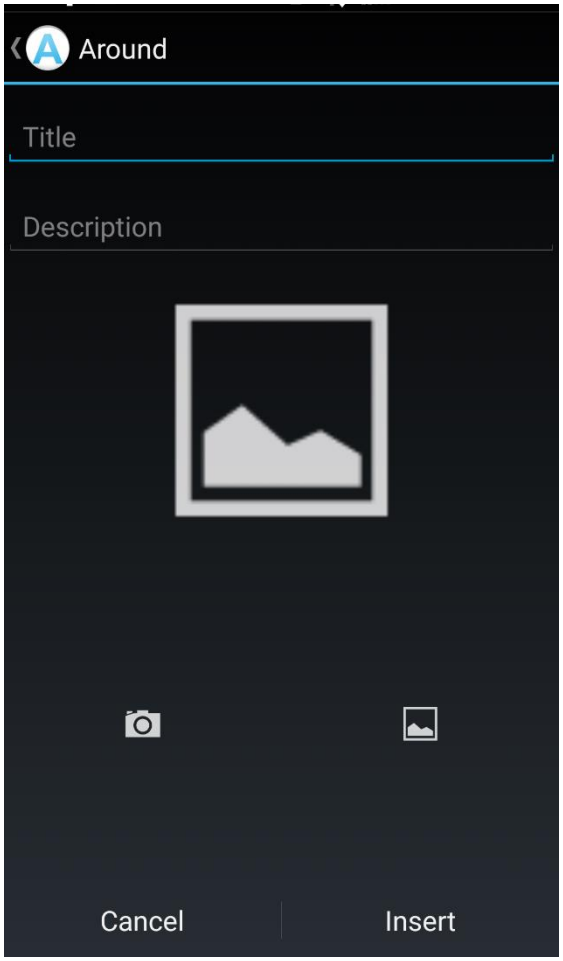

FIGURA 86: PLANTILLA DE ENVÍO DE NOTAS

En la imagen aparecen dos campos para insertar texto con el título y la descripción de la nota. El único campo obligatorio para insertar una nota es el título. Encima de los botones de Insertar y Cancelar, aparecen dos botones adicionales. El de la derecha nos permite seleccionar una imagen de la galería y el de la izquierda hacer una nueva imagen para enviarla a otro dispositivo. En ningún caso es obligatorio añadir una imagen a una Nota, pero si esta acción se realiza, la imagen a enviar será mostrada en esta pantalla.

Los otros dos botones restantes del mapa, nos permiten ocultar los marcadores para poder seleccionar una posición que se encuentre cubierta por otras Notas. Esto es posible al pulsar el botón de la izquierda, como se muestra en la Figura 87.

FIGURA 87: OCULTACIÓN DE MARCADORES

Se puede apreciar en la Figura 87, que el botón central también se ha ocultado en este caso. Ese botón se emplea para que se muestren en la pantalla todas las Notas del usuario. En el caso de la Figura 88, se muestran Notas en diferentes países y continentes.

FIGURA 88: IMAGEN CON TODAS LAS NOTAS DE UN USUARIO

Por último, pulsando en cualquiera de los marcadores, se abre una ventana indicándonos el título de la Nota y la descripción (si tiene) como aparece en la Figura 89. Al pulsar de nuevo en esa ventana se nos permite eliminar esa nota únicamente.

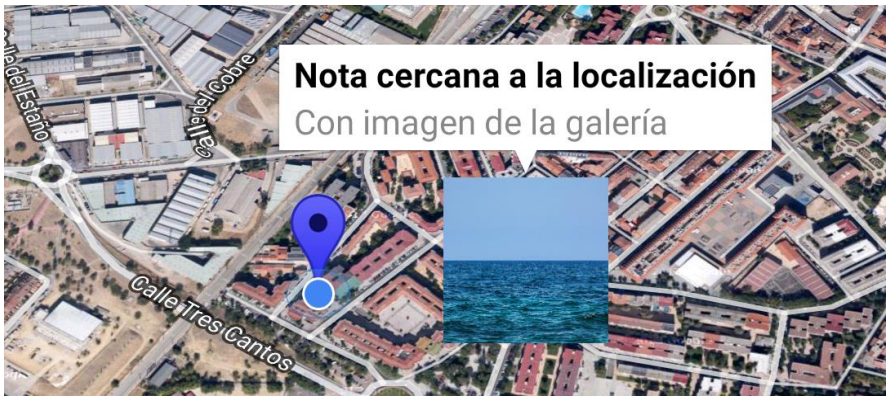

FIGURA 89: VENTANA CON INFORMACIÓN DE UNA NOTA

Al hacer *click* en la ventana de la Figura 89, se despliega un aviso que nos pide confirmación de la eliminación.

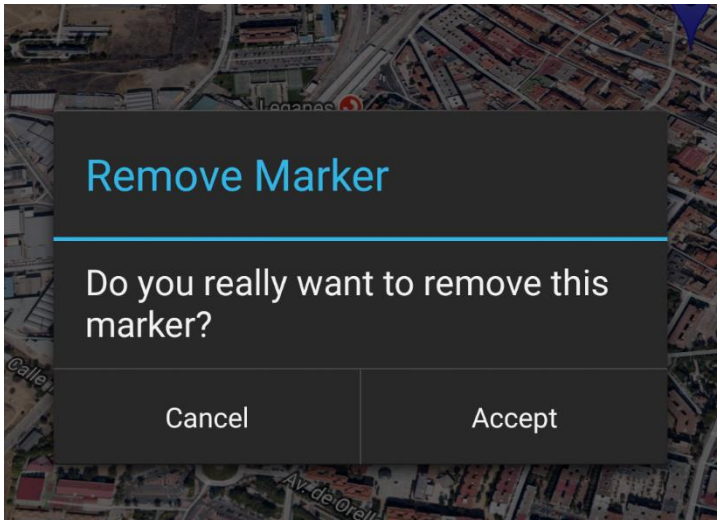

FIGURA 90: ELIMINACIÓN DE UNA NOTA

Al pulsar aceptar en la Figura 90, la Nota será eliminada del mapa y de la base de datos del usuario.

Glosario

ACK	*Acknowledge*
ACL	*Access Control List*
ADU	*Application Data Unit*
API	*Application Programming Interface*
ARP	*Address Resolution Protocol*
ARPANET	*Advanced Research Projects Agency Network*
ART	*Android RunTime*
DTN	*Delay Tolerant Network*
EID	*Endpoint Identifier*
FTP	*File Transfer Protocol*
GPS	*Global Positioning System*
GUI	*Graphical User Interface*
IBR	*Institut für Betriebssysteme und Rechnerverbund*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
LED	*Light Emitting Diode*
LIFO	*Last In First Out*
MAC	*Media Access Control*
MRG	*Minimum Reception Group*
OS	*Operating System*
RFC	*Request for Comments*
RTT	*Round Trip Time*
SQL	*Structured Query Language*
SSL	Secure Sockets Layer
TCP	*Transport Control Protocol*

UDP	*User Datagram Protocol*
URI	*Uniform Resource Identifier*
Wi-Fi	*Wireless Fidelity*

Referencias

[1] RFC 4838. *Delay-Tolerant Networking Architecture. RFC 4838.* Disponible [Internet]: <https://tools.ietf.org/html/rfc4838> [02 de julio de 2015]

[2] IBRDTN. *IBR-DTN Project Technische Universität Braunschweig.* Disponible [Internet]: <https://www.ibr.cs.tu-bs.de/projects/ibr-dtn/> [02 de julio de 2015]

[3] RFC 5050. *Bundle Protocol Specification. RFC 5050.* Disponible [Internet]: <http://tools.ietf.org/html/rfc5050> [02 de julio de 2015]

[4] Android 2 Application Development Book. *Meier, Reto. Professional Android 2 Application Development. Wrox. 2010*

[5] RFC 3986. *Uniform Resource Identifier (URI): Generic Syntax. RFC 3986.* Disponible [Internet]: <https://tools.ietf.org/html/rfc3986> [02 de julio de 2015]

[6] Android Development Web. *Web de Desarrollo Android.* Disponible [Internet]: <http://www.android.com/> [02 de julio de 2015]

[7] IBR-DTN Group Apps Examples. Example a*pplitations available on Google Play of the IBR-DTN Group to show the IBR-DTN daemon capabilities.* Disonible [Internet]: < https://play.google.com/store/apps/developer?id=IBR,+TU+Braunschweig> [02 de julio de 2015]

[8] Android Developers Web. *Web de Desarrolladores Android.* Disponible [Internet]: <http://developer.android.com> [02 de julio de 2015]

[9] IBR-DTN Master App. *Google Play Application for the IBR-DTN Master.* Disponible [Internet]: <https://play.google.com/store/apps/details?id=de.tubs.ibr.dtn> [02 de julio de 2015]

[10] Android 4 Application Development Book. *Meier, Reto. Professional Android 4 Application Development. Wrox. 2012*

[11] IBR-DTN Android Wiki. *Wiki Page of the Android Development area of the IBR-DTN Project.* Disponible. [Internet] <https://trac.ibr.cs.tu-bs.de/project-cm-2012-ibrdtn/wiki/android-dev> [02 de julio de 2015]

[12] DDA. *Google Play Developer Distribution Agreement (DDA).* Disponible [Internet]: <https://play.google.com/about/developer-distribution-agreement.html> [02 de julio de 2015]

[13] DPP. *Google Play Developer Program Policies (DPP).* Disponible [Internet]: <https://play.google.com/about/developer-content-policy.html> [02 de julio de 2015]

[14] IBRDTN Wiki. *Wiki Page of the IBR-DTN Project.* Disponible [Internet]: <https://trac.ibr.cs.tu-bs.de/project-cm-2012-ibrdtn> [02 de julio de 2015]

[15] IBR-DTN Android GitHub. *GitHub repository for the of the IBR-DTN Android Project*. Disponible [Internet] <https://github.com/ibrdtn/ibrdtn/tree/master/android> [02 de julio de 2015]

[16] ART & Dalvik. *ART and Dalvik New Android Runtime Features and Problems*. Disponible [Internet]: <https://source.android.com/devices/tech/dalvik/> [02 de julio de 2015]

[17] Cutoa de Mercado Smartphones Mundial. *Gartner Says Emerging Markets Drove Worldwide Smartphone Sales to 19 Percent Growth in First Quarter of 2015*. Disponible [Internet]: < http://www.gartner.com/newsroom/id/3061917> [02 de julio de 2015]

[18] Video de Demostración de la Aplicación *Around*. Disponible [Internet]: <https://www.youtube.com/watch?v=1kGHoC8k4vg&feature=youtu.be> [02 de julio de 2015]

[19] DTNSEC. *Bundle Security Protocol Specification draft-irtf-dtnrg-bundle-security-19*. Disponible [Internet]: <https://tools.ietf.org/html/draft-irtf-dtnrg-bundle-security-19> [02 de julio de 2015]

[20] RFC 959. *FILE TRANSFER PROTOCOL (FTP) RFC*. Disponible [Internet]: <https://www.ietf.org/rfc/rfc959.txt> [02 de julio de 2015]

[21] RFC 793. Transmission Control Protocol (TCP) RFC. Disponible [Internet]: <https://tools.ietf.org/html/rfc793> [02 de julio de 2015]

[22] Implementaciones DTN. *Lista de las implementaciones DTN disponibles*. Disponible [Internet]: <https://sites.google.com/site/dtnresgroup/home/code> [02 de julio de 2015]

[23] RFC 6257. *Bundle Security Protocol Specification*. Disponible [Internet]: <http://tools.ietf.org/html/rfc6257> [02 de julio de 2015]