Biblioteca para comunicación directa entre dispositivos basada en tecnologías P2P

Library for direct communication between devices based on P2P technologies

Trabajo de Fin de Grado

Grado en Ingeniería Informática

VNiVERSiDAD D SALAMANCA

Septiembre de 2025

Autor

Emilio Cobos Álvarez (emiliocobos@usal.es)

Tutores

Guillermo González Talaván (gyermo@usal.es)
Pedro Martín Vallejo Llamas (pedrito@usal.es)

VNiVERSiDAD D SALAMANCA

Facvltad D Ciencias
VNiVERSiDAD D SALAMANCA

Certificado de los tutores

D. Guillermo González Talaván, profesor del Departamento de Informática y Automática de la Universidad de Salamanca,

D. Pedro Martín Vallejo Llamas, profesor del Departamento de Informática y Automática de la Universidad de Salamanca,

HACEN CONSTAR:

Que el trabajo titulado "Biblioteca para comunicación directa entre dispositivos basada en tecnologías P2P", que se presenta, ha sido realizado por Emilio Cobos Álvarez, con DNI ***12324N y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Ingeniería Informática en esta Universidad.

En Salamanca, a 3 de Septiembre de 2025

Firmado por VALLEJO LLAMAS PEDRO MARTIN - ***2775** el día 01/09/2025 con un certificado emitido por AC FNMT Usuarios

Firmado por GONZALEZ TALAVAN GUILLERMO - DNI ***7746** el día 03/09/2025 con un certificado emitido por AC Sector Público

Resumen

El acceso a internet no es tan universal como suele parecer. Sin embargo, dispositivos convencionales al alcance de la mayoría de la población soportan comunicarse entre ellos de manera directa, via tecnologías estándar como Bluetooth, WiFi-Direct, u otras.

Estas tecnologías tienen casos de usos muy variados, como comunicación en situaciones de emergencia o lugares remotos, intercambio de datos de manera más privada que una conexión a internet convencional...

A pesar de ello, su grado de adopción no es particularmente grande, en parte por la dificultad de uso de estas tecnologías en comparación con internet.

Se desarrollará una biblioteca que abstraiga sobre diferentes tecnologías de comunicación directa, y además proporcione capacidades de agrupación e identificación de más alto nivel.

Abstract

Internet access is not so ubiquitous as it seems. However, lots of common devices accessible to large parts of the population support peer-to-peer communication, via standard tech like Bluetooth, WiFi-Direct, or others.

These technologies have various use cases like communications in remote places or emergency situations, more private data exchange compared to usual internet connections...

The adoption levels of these technologies is not great tho, partially due to the difficulty of use, compared to regular internet use.

We'll develop a library that abstracts over multiple peer-to-peer technologies, and provides higher-level grouping and identification capabilities.

Glosario

AES *Advanced Encryption Standard*, un algoritmo de cifrado simétrico ampliamente utilizado.

ASan *Address Sanitizer*, detector de errores de memoria para lenguajes compilados.

AWDL *Apple Wireless Device Link*, un protocolo propietario de Apple, similar en funcionalidad a WiFi Direct, usado para funcionalidad como *AirDrop*.

CI/CD La Integración Continua/Entrega Continua (CI/CD, por sus siglas en inglés, *Continuous Integration/Continuous Delivery*) es una práctica de desarrollo de software que automatiza el proceso (tradicionalmente manual) de llevar código desde su creación hasta su entrega en producción. Abarca las últimas fases del ciclo de vida del desarrollo de software, incluyendo la compilación, pruebas, implementación y despliegue de aplicaciones.

D-Bus Protocolo para la comunicación entre procesos usado en Linux.

DMA *Digital Markets Act*, regulación a nivel europeo para hacer los mercados digitales más competitivos.

ECDH *Elliptic Curve Diffie-Hellman*, una variante del protocolo Diffie-Hellman para establecer un secreto compartido via un canal inseguro que usa claves basadas en curvas elípticas.

GCM *Galois/Counter Mode*, un modo de operación que, dado algoritmo de cifrado simétrico, permite cifrar datos y también verificarlos.

GPL *GNU General Public License*, licencia copyleft del proyecto GNU que proporciona al usuario la libertad de usar, modificar, y redistribuir el software, con la condición de que trabajos derivados de él sean licenciados bajo los mismos términos.

GTK Originalmente *GIMP ToolKit*, es un toolkit libre para la creación de interfaces de usuario..

hole punching Técnica para establecer una conexión directa entre dos dispositivos a través de internet, donde estos dispositivos pueden estar detrás de firewalls o NAT. Generalmente se usan protocolos como STUN, TURN, o QUIC address

discovery [47]. TailScale tiene una genial explicación de cómo implementan NAT traversal para implementar su plataforma [19].

JNI La Interfaz Nativa de Java (*Java Native Interface* en inglés) es una interfaz que permite interactuar a la JVM con código nativo como el escrito en lenguajes como C, C++, o Rust.

JVM La Máquina Virtual de Java (*Java Virtual Machine* en inglés) es el entorno de ejecución que ejecuta el bytecode de Java para una plataforma específica.

kvm *Kernel-based Virtual Machine*, solución de virtualización de Linux integrada en el núcleo..

LoRa Long Range en inglés, una tecnología propietaria de comunicación por radio de baja frecuencia.

MANET Mobile Ad-Hock Networks, un tipo de red mallada inalámbrica en la que todos los nodos se conectan directamente de manera directa, dinámica, y no jerárquica.

Markdown Lenguaje de marcado ligero con formato de texto plano diseñado para ser fácil de leer y escribir.

memory-safe *Memory safety* se refiere a la capacidad de un sistema o lenguaje de programación de prevenir problemas y vulnerabilidades de seguridad relacionadas con los accesos a memoria.

MITM *Man In The Middle*, o ataque de intermediario, un tipo de ataque en el que un atacante se interpone entre dos partes que están comunicándose, interceptando y posiblemente modificando la información que se intercambia sin que ninguna de las partes se dé cuenta.

MLS *Messaging Layer Security*, un protocolo estándar [12] de la IETF para el intercambio de claves en grupos de mensajería.

MPSC *Multiple Producer Single Consumer*, modelo de paso de mensajes donde uno o varios mensajeros envían (producen) mensajes, y un sólo consumidor los recibe..

NDP *Neighbor Discovery Protocol*, un protocolo de IPv6 para el descubrimiento de vecinos en la misma red y puertas de enlace.

QUIC Protocolo de transporte de red orientado a conexión, pero sobre UDP, lo cual consigue beneficios de rendimiento comparado con TCP. Gracias a que funciona sobre UDP, es más resistente a la pérdida de paquetes y por lo tanto es útil a la hora de hacer bypass de NATs y firewalls [19] [29].

thread-safe *Thread safety* se refiere a la capacidad de un sistema de comportarse correctamente en presencia de accesos desde múltiples hilos concurrentemente.

Trait Colección de métodos, tipos asociados, etc independientes de la implementación en Rust. Similar a las interfaces en otros lenguajes de programación.

Transpilación Proceso mediante el cual se convierte código fuente de un lenguaje de programación a otro de un nivel de abstracción similar. Esto es en contraposición a la compilación, que convierte código de un lenguaje de programación de un nivel más alto a otro de un nivel más bajo.

wpa_supplicant Implementación de software libre de un suplicante WPA, utilizado para gestionar conexiones a redes WiFi en Linux, Android, y otros sistemas operativos.

WPS *WiFi Protected Setup*, método de conexión a una red inalámbrica sin contraseña (via botón o pin).

Agradecimientos

A Pedro, por la insistencia implacable.

A Guillermo y a Pablo, por sacar ese rato todas las semanas.

A Sam, por cada rato extra que me has dado para hacer este trabajo.

A Sammy, por querer teclear conmigo todos los fines de semana.

A Carlos, porque me picas, y si no no hubiera acabado esto.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

La comunicación peer-to-peer (punto a punto, sin necesidad de infraestructura centralizada), no son nada nuevo. Las MANET son comunes en entornos de emergencia, tecnología militar, etc [52].

Esta tecnología es accesible a la mayoría de la población, via protocolos como Bluetooth que usamos constantemente. Sin embargo este tipo de redes no son comunes, a pesar de que las tarjetas de red inalámbrica de teléfonos y ordenadores modernos soportan los protocolos necesarios para comunicación directa por WiFi, y de que las propiedades de privacidad de estas tecnologías son muy superiores a internet.

La principal hipótesis es que esto es porque usar este tipo de tecnologías en hardware de consumo es difícil, por la complejidad de estos protocolos, que se describirá en las siguientes secciones.

Por ello, se plantea la creación de una biblioteca que abstraiga los detalles de más bajo nivel (protocolo utilizado, establecimiento del enlace, direccionamiento básico), para hacer más viable su adopción.

Capítulo 2

Objeto

2.1. Objetivos del Sistema

El objetivo principal del trabajo es desarrollar una biblioteca que facilite la comunicación peer-to-peer entre dispositivos, y además proporcione varias capacidades de alto nivel como autenticación y envío de mensajes.

Los objetivos definidos son:

- Permitir a varios dispositivos enviar mensajes entre ellos sin necesidad de conexión a internet.
- Proveer una abstracción de bajo nivel sobre la tecnología física de comunicación, con al menos una implementación como prueba de concepto. Opcionalmente, la biblioteca también abstraerá diferencias entre plataformas y sistemas operativos.
- Proveer una abstracción de más alto nivel que permitirá:
  - Formación de grupos lógicos dentro de un grupo físico. Opcionalmente, se investigará la posibilidad de que un grupo lógico abarque más de un grupo físico.
  - Identificación (via sistema de clave pública / privada o similar), independiente de la capa física.
  - Opcionalmente, enrutado de mensajes via: Broadcast / Broadcast a un grupo lógico / Mensaje directo entre dos nodos lógicos (identidades).
- Desarrollar una aplicación sencilla que demuestre las capacidades de la biblioteca.

2.2. Objetivos personales

Mi `$dayjob` es desarrollar el motor de renderizado web de Firefox. A pesar de ello, mi conocimiento en las capas de redes inferiores a la capa de transporte antes de empezar este trabajo era bastante superficial.

Similarmente, mis conocimientos de desarrollo en Android eran igualmente superficiales: He tenido que trabajar y depurar en Android, con Java, Kotlin, y la

NDK, pero nunca he tenido que integrar algo nuevo.

Este trabajo se presentó como una buena oportunidad de ampliar el rango de mis conocimientos en ambos campos, mientras pongo a buen uso mi conocimiento pre-existente sobre programación de sistemas, a la vez de explorar y hacer más fácil el uso de tecnologías con mejores características de privacidad que las que la mayoría de la gente usa diariamente.

Capítulo 3

Antecedentes

3.1. Acceso a Internet, centralización y privacidad

Aunque el acceso a internet ha crecido exponencialmente desde finales del siglo veinte, su disponibilidad sigue siendo desigual. Según datos de la Unión Internacional de Telecomunicaciones, a finales de 2024, aproximadamente un 30 % de la población mundial aún no tenía acceso regular a internet [28]. Esta brecha digital afecta especialmente a comunidades rurales, zonas en desarrollo y regiones afectadas por conflictos o desastres naturales.

En sus inicios, Internet fue concebida como una red descentralizada de nodos capaces de comunicarse directamente, diseñada para ser resiliente ante fallos parciales. Sin embargo, con el paso de los años, la arquitectura de internet ha evolucionado hacia un modelo cada vez más centralizado, donde gran parte del tráfico global se canaliza a través de servidores propiedad de un número reducido de grandes corporaciones tecnológicas [37].

Esta centralización tiene varias consecuencias preocupantes, tanto para la resiliencia de las comunicaciones en situaciones de emergencia, conflictos o censura, como para la privacidad, libertad de expresión, y autonomía de los usuarios.

Frente a esta realidad, surgen soluciones que buscan recuperar la descentralización original de internet, permitiendo a los dispositivos comunicarse entre sí directamente, sin necesidad de depender de intermediarios ni infraestructura externa. Estas alternativas permiten fomentar modelos de comunicación más autónomos, resistentes y respetuosos con la privacidad.

3.2. Infraestructura técnica: Redes ad-hoc y comunicación punto a punto

Las tecnologías de comunicación directa entre dispositivos — como Bluetooth, Wi-Fi Direct, y modos ad-hoc de Wi-Fi — permiten establecer redes sin infraes-

tructura, también llamadas redes ad-hoc. Estas redes son especialmente útiles en contextos donde no existe un punto de acceso centralizado, o donde se desea evitar su uso por razones de privacidad o autonomía.

En el ámbito de la investigación, las redes malladas (mesh networks) han sido ampliamente estudiadas como una solución escalable y auto-organizada para conectar múltiples dispositivos sin necesidad de infraestructura fija [3]. Estas redes permiten que cada nodo actúe como cliente y repetidor, lo que amplía el alcance de la red sin necesidad de un servidor central.

La adopción de este tipo tecnologías ha sido limitada tanto por parte de usuarios como de desarrolladores. Por ejemplo, aunque tecnologías como Wi-Fi Direct están ampliamente disponibles a nivel de hardware, su implementación y uso en software comercial sigue siendo marginal.

Desde una perspectiva sociotécnica, esta baja adopción también responde a dinámicas de mercado. Las plataformas centralizadas ofrecen comodidad y servicios integrados que han establecido una expectativa de funcionamiento transparente, lo que dificulta que modelos descentralizados compitan en términos de experiencia de usuario. Además, las grandes plataformas no promueven activamente estas formas de comunicación, en parte porque reducen su capacidad de control y monetización.

Por tanto, la brecha entre el potencial técnico y la adopción real de estas tecnologías subraya la necesidad de herramientas que simplifiquen su uso, abstraigan su complejidad, y proporcionen interfaces más accesibles para los desarrolladores. Esta es precisamente la línea de trabajo en la que se enmarca el presente proyecto.

Capítulo 4

Descripción de la situación actual

A continuación se exponen varias tecnologías en uso o desarrollo relacionadas con el proyecto, divididas en aplicaciones orientadas a dispositivos comunes (basadas en Wifi Direct o Bluetooth), software que necesita hardware menos común como radios (y por lo tanto dirigido a usuarios más especializados), y bibliotecas / SDKs dirigidas a desarrolladores.

Finalmente, se hace una revisión a la implementación de Wifi Direct usada por Android y Linux, ya que durante este trabajo he usado dicha tecnología para el prototipo.

4.1. Aplicaciones móviles descentralizadas

Algunas aplicaciones móviles han explorado activamente el uso de tecnologías de red directa para proporcionar comunicación descentralizada, resiliente y, en muchos casos, privada. La mayoría de los casos de uso están

FireChat, desarrollada por Open Garden, fue una de las primeras aplicaciones populares en usar Wi-Fi Direct y Bluetooth para crear redes mesh entre teléfonos móviles, especialmente útil durante protestas o en lugares sin conexión. Permitía enviar mensajes entre usuarios cercanos sin conexión a internet. Su uso se hizo notable durante protestas como las de Hong Kong en 2014 [13]. Aunque fue descontinuada en 2018, demostró la viabilidad de estas tecnologías para comunicación temporal.

Otras aplicaciones similares están en uso, como *Briar* [14], que funciona mediante Wi-Fi Direct, Bluetooth o Tor, o *Bridgefy* [15] (a pesar de múltiples problemas de seguridad [4] [5]).

Recientemente, una aplicación similar llamada BitChat [1], creada for Jack Dorsey ha ganado tracción gracias a la popularidad de su creador. Su whitepaper [2] describe como usan el protocolo Noise [40] para garantizar la confidencialidad.

4.2. Comunicaciones P2P basadas en radio

Meshtastic [34] es un proyecto open source que utiliza dispositivos con radios LoRa para transmitir mensajes de texto entre usuarios sin internet. Su bajo consumo de energía y gran alcance lo hacen ideal para senderismo, comunidades rurales, o situaciones de emergencia. Cada nodo funciona como repetidor dentro de una red mesh.

Aparte de este proyecto, existe mucha industria militar basada en este tipo de tecnología para crear MANETs [33].

4.3. Librerías y APIs de alto nivel

Con el objetivo de simplificar la integración de comunicación P2P en aplicaciones, diversas plataformas han desarrollado APIs de alto nivel que abstraen los detalles técnicos subyacentes.

4.3.1. Google Nearby

Google Nearby Connections API permite crear redes de comunicación directa entre dispositivos Android usando una combinación de Bluetooth, BLE, Wi-Fi y Wi-Fi Direct. Proporciona detección de proximidad, establecimiento de conexión y transferencia de datos bajo un modelo pub-sub [23].

4.3.2. Multipeer Connectivity Framework

Multipeer Connectivity Framework, de Apple, ofrece capacidades similares en iOS/macOS, permitiendo descubrir dispositivos cercanos y establecer sesiones seguras de comunicación, sin necesidad de servidores externos [10].

4.3.3. Menciones ilustres

Dentro del ecosistema de comunicación peer-to-peer, existen algunas bibliotecas con funcionalidad interesante, pero que no son directamente comparables a lo que queremos hacer, porque no se encargan del descubrimiento de dispositivos, es decir, tienes que saber por adelantado el dispositivo al que te quieres conectar de otra forma.

LibP2P

libp2p, parte del ecosistema de IPFS y utilizado en protocolos relacionados con criptomonedas, ofrece una pila modular para comunicación P2P [30].

Está más centrado en redes distribuidas a gran escala, sus conceptos de transporte y autenticación también son relevantes para entornos sin infraestructura.

iroh

iroh [35] es una biblioteca similar para establecer comunicación directa y un canal via QUIC entre dispositivos a través de internet. Dadas dos direcciones IP, trata de conectarlas directamente, haciendo hole punching y fallback a un relay si es necesario.

Capítulo 5

Normas y referencias

5.1. Métodos

En este apartado se explica la metodología de desarrollo software seguida, desde un punto de vista técnico.

5.1.1. Metodología ágil: Scrum

Dadas las limitaciones temporales (trabajo a tiempo completo del autor y familia), se ha optado por usar una metodología ágil ligera (Scrum), ya que permite una gran adaptabilidad y se centra en el progreso incremental [44].

En Scrum hay tres roles definidos: el *Product Owner*, responsable de definir y priorizar los objetivos del proyecto; el equipo de desarrollo, que se encarga de desarrollar el producto; y el *Scrum Master*, que es responsable de facilitar el proceso, eliminar fricciones entre los otros roles y asegurar que se siguen las prácticas de Scrum.

La metodología se basa en intervalos cortos de trabajo llamados *sprints*, que son períodos de tiempo fijos, generalmente de dos a cuatro semanas. El objetivo de cada sprint es producir un incremento de producto potencialmente entregable.

Cada *sprint* comienza con una reunión de planificación donde el equipo selecciona un conjunto de tareas del *Product Backlog* (lista de tareas pendientes) para completar durante el sprint, y las mueve al *Sprint Backlog* (lista de tareas pendientes del sprint).

Durante el *sprint*, el equipo trabaja en las tareas seleccionadas, y se realizan reuniones diarias de seguimiento, conocidas como *Daily Scrum*, donde se revisa el progreso, se identifican obstáculos y se ajusta el plan si es necesario.

Al final de cada *sprint* se realizan dos eventos. Primero, se lleva a cabo una reunión de revisión para demostrar el trabajo completado y recibir retroalimentación.

Después de la revisión, se realiza una reunión de retrospectiva donde el equipo reflexiona sobre el sprint y discute qué funcionó y qué no, con el fin de mejorar continuamente el proceso.

Aplicación de Scrum al TFG

En el caso de este TFG, los *Product Owner* son los co-tutores del TFG, mientras que el autor del TFG asume los roles de *Scrum Master* y equipo de desarrollo.

Los sprints han tenido una duración de una semana, al final de los cuales tenía lugar una reunión de revisión de sprint con el co-tutor en la que se revisaban las tareas completadas, se recibía retroalimentación y se actualizaban los objetivos del proyecto.

No se han llevado a cabo reuniones diarias de seguimiento, ya que el desarrollador es el único integrante del equipo, y, por tanto, tiene una visión holística del progreso y los obstáculos encontrados.

Para la estimación del esfuerzo de las tareas se han utilizado puntos de historia, una unidad de medida relativa que mide el esfuerzo necesario para completar una tarea comparándola con otras tareas del proyecto.

5.2. Herramientas

A continuación se describen las herramientas utilizadas durante el desarrollo, y documentación del proyecto, agrupadas en cuatro categorías: herramientas de implementación, metodológicas, de documentación y prototipado, y otras.

5.2.1. Herramientas de implementación

Lenguaje de programación: Rust

Se ha elegido Rust[1] para la implementación principal de la biblioteca. Rust es un lenguaje de propósito general que tiene varias propiedades muy interesantes para un proyecto de este tipo:

- **Multi-plataforma**: Rust soporta una amplia variedad de arquitecturas y plataformas [18].
- **Rendimiento**: Rust es un lenguaje compilado sin ningún requerimiento de *runtime* ni recolector de basura. Es un lenguaje que es usable para la programación de sistemas con baja latencia.
- **Seguridad**: A diferencia de otros lenguajes que cumplen los requisitos anteriores, como C / C++, Rust es memory-safe y thread-safe por defecto, lo que evita una gran cantidad de problemas de seguridad que siguen plagando el ecosistema de software actual y lo hace apropiado para aplicaciones críticas

[1] Rust: `https://rust-lang.org`

y a gran escala [42] [43], pero también ayuda durante el desarrollo (evitando problemas difíciles de reproducir y solventar).

El autor de este TFG además está bastante familiarizado con Rust y la interoperabilidad con otros lenguajes (siendo este el mantenedor de varias bibliotecas muy populares para este propósito como bindgen[2] y cbindgen[3], por lo que tener que interoperar con otros lenguajes para las diferentes plataformas no parecía un gran desafío.

Se han usado otras herramientas estándar pertenecientes al ecosistema de Rust como cargo[4] para la gestión de dependencias.

Lenguaje de programación: Kotlin

Se ha elegido Kotlin[5] como lenguaje para desarrollar la demo en Android. Kotlin es un lenguaje moderno y estáticamente tipado basado en la JVM.

A pesar de haber otras alternativas para hacer aplicaciones móviles como Dart[6] (via Flutter[7]), Kotlin es el lenguaje mayoritario y recomendado por Google a la hora de desarrollar en Android [25], y soluciona bastantes problemas comunes y de ergonomía de Java [21].

Lenguaje de programación: Java

Java[8] es un lenguaje de alto nivel orientado a objetos, de propósito general. Se ha usado este lenguaje para interoperar entre Rust y Kotlin, via la JNI.

La JNI está mejor documentada para Java que para Kotlin. Kotlin y Java interoperan de forma casi transparente, lo cual lo hizo una decisión más conveniente a la hora de exponer la biblioteca a Android.

Lenguaje de programación: C

Durante el desarrollo, se han tenido que investigar múltiples problemas y situaciones inesperadas relacionadas con wpa_supplicant, y se han enviado y aceptado varias mejoras a este software escrito en C:

- P2P: Provide better failure reason for group formation errors[9]: Mejora el reporte de errores de wpa_supplicant.

[2]bindgen: `https://github.com/rust-lang/rust-bindgen`
[3]cbindgen: `https://github.com/mozilla/cbindgen`
[4]cargo: `https://doc.rust-lang.org/cargo/`
[5]Kotlin: `https://kotlinlang.org/`
[6]Dart: `https://dart.dev/`
[7]Flutter: `https://flutter.dev/`
[8]Java: `https://www.java.com`
[9]P2P: Provide better failure reason for group formation errors: `https://lists.infradead.org/pipermail/hostap/2025-January/043247.html`

- dbus: Expose P2PDevice's own device address[10]: Beneficioso para evitar colisiones durante la asociación de direcciones.
- dbus: Expose P2P auto_join behavior[11]: Permite unirse automáticamente a un grupo existente.
- dbus: Expose group's GO device address[12]: Expone la dirección del *Group Owner* de WiFi Direct, para poder realizar el intercambio de claves en menos pasos.

Toolkit de interfaz de usuario: Jetpack Compose

Se ha usado Jetpack Compose[13] para la interfaz de usuario en la demo de Android. Es el toolkit recomendado por Google para el desarrollo de aplicaciones Android.

Toolkit de interfaz de usuario: GTK

Se ha usado GTK[14] junto con libadwaita[15] para la interfaz de usuario en la demo de Linux.

5.2.2. Herramientas metodológicas

Depurador: rr

rr[16] es un depurador para Linux que permite grabar la ejecución de un proceso (y todos sus sub-procesos) y luego reproducirla de manera determinista con poca latencia comparada a la ejecución sin un depurador [38].

A diferencia de otros depuradores tradicionales como gdb[17], que solo permiten examinar el estado actual del programa mientras se ejecuta, rr permite depurar la misma ejecución de un programa tantas veces como sea necesario, y también realizar una depuración "hacia atrás". Ambas características son invaluables para depurar trabajos como este, y software con el que el autor no estaba familiarizado como wpa_supplicant.

El autor de este TFG también ha contribuido[18] a este proyecto, aunque no como parte de este trabajo.

[10] dbus: Expose P2PDevice's own device address: `https://lists.infradead.org/pipermail/hostap/2025-May/043428.html`

[11] dbus: Expose P2P auto_join behavior: `https://lists.infradead.org/pipermail/hostap/2025-May/043429.html`

[12] dbus: Expose group's GO device address: `https://lists.infradead.org/pipermail/hostap/2025-August/043695.html`

[13] Jetpack Compose: `https://developer.android.com/compose`

[14] GTK: `https://www.gtk.org/`

[15] libadwaita: `https://gnome.pages.gitlab.gnome.org/libadwaita/`

[16] rr: `https://rr-project.org/`

[17] gdb: `https://sourceware.org/gdb/`

[18] ha contribuido: `https://github.com/rr-debugger/rr/commits?author=emilio`

5.2.3. Herramientas para la elaboración del TFG

Documentación: LaTeX

LaTeX[19] es «un sistema de preparación de documentos de alta calidad. [...] Es el estándar de facto para la comunicación y publicación de documentos científicos y técnicos» [41].

Documentación: Pandoc

Pandoc[20] es una herramienta de software libre escrita en Haskell[21] que permite convertir entre lenguajes de marcado. En particular, se ha usado para usar Markdown para la mayoría del contenido de la memoria y transpilarlo a LaTeX, para su inclusión en el documento final.

Documentación: Mermaid

Mermaid[22] es una herramienta de software libre para generar diagramas a partir de texto plano. Se ha utilizado para generar los varios diagramas incluidos en esta memoria y los anexos.

Documentación: Claude

Claude[23] es un modelo de inteligencia artificial de Anthropic. Se ha utilizado para generar versiones iniciales de algunos de los diagramas en los anexos, ya que escribir mermaid a mano es propenso a errores, y el autor no estaba demasiado familiarizado con Mermaid previa a la elaboración de esta memoria.

Como persona escéptica de estos modelos en general he de decir que, a pesar de cometer una gran cantidad de errores (muchos de los diagramas generados no compilaban) generó diagramas correctos a grandes rangos, lo que fue útil como punto de partida.

También se ha utilizado para producir un resumen inicial del log de git, con resultados similares (resultados aceptables como punto de partida, pero una gran cantidad de revisión necesaria).

Documentación: rustdoc

rustdoc[24] es la herramienta oficial de Rust para generar documentación desde el código fuente.

[19] LaTeX: `https://www.latex-project.org/`
[20] Pandoc: `https://pandoc.org/`
[21] Haskell: `https://www.haskell.org/`
[22] Mermaid: `https://mermaid.js.org/`
[23] Claude: `https://claude.ai/`
[24] rustdoc: `https://doc.rust-lang.org/rustdoc/what-is-rustdoc.html`

5.2.4. Otras herramientas

Control de versiones: Git

Git[25] es un sistema de control de versiones distribuido de código abierto.

Es ampliamente utilizado en la industria del software para gestionar el código fuente y colaborar en proyectos de desarrollo.

Alojamiento de código: GitHub

GitHub[26] es una plataforma de alojamiento de código que utiliza Git como sistema de control de versiones.

Proporciona funcionalidades adicionales sobre Git, como la gestión de colaboración en proyectos de software, incluyendo la posibilidad de crear solicitudes de incorporación de cambios (en inglés, *pull requests*), gestionar incidencias (en inglés, *issues*), wikis, y la integración con herramientas de CI/CD.

[25] Git: `https://git-scm.com`
[26] GitHub: `https://github.com`

Capítulo 6

Requisitos iniciales

En este capítulo se recogen, de manera resumida, los requisitos del sistema, especificados siguiendo la Metodología para la Elicitación de Requisitos de Sistemas Software de Durán Toro y Bernárdez Jiménez [9].

Este capítulo constituye un resumen de las partes más relevantes de dicho anexo.

6.1. Requisitos funcionales

Los requisitos funcionales definen «qué debe hacer el sistema con la información almacenada para alcanzar los objetivos de su negocio».

Se han definido los siguientes requisitos funcionales para el sistema:

- El sistema deberá permitir a varios dispositivos enviar mensajes entre ellos sin necesidad de conexión a internet.
- El sistema deberá abstraer la tecnología física de comunicación.
- El sistema deberá opcionalmente proveer implementaciones para distintas plataformas.
- El sistema debe permitir la creación, unión y salida de grupos lógicos dentro de un grupo físico.
- El sistema deberá permitir la identificación via un sistema de clave pública / privada independiente de la capa física.
- Opcionalmente, el sistema debe permitir la interconexión de múltiples grupos físicos en un solo grupo lógico.
- El sistema debe garantizar que los mensajes puedan ser autenticados y, opcionalmente, cifrados de extremo a extremo.
- Debe desarrollarse una aplicación que use la biblioteca y valide sus capacidades.

6.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que definen cualidades sobre el sistema que no están directamente relacionadas con la funcionalidad del mismo, sino con aspectos como el rendimiento, la usabilidad, la seguridad, etc [22].

Así, se han definido los siguientes requisitos no funcionales para el sistema:

1. **Portabilidad**: Accesibilidad desde diferentes plataformas y sistemas operativos.
2. **Extensibilidad**: Facilidad para añadir nuevas plataformas y transportes físicos.
3. **Seguridad**: Identificación y cifrado de mensajes independiente de la capa física.
4. **Accesibilidad**: La biblioteca deberá funcionar con dispositivos accesibles / no requerir hardware especial.

Capítulo 7

Hipótesis, restricciones y alcance

En este capítulo se detallan la hipótesis inicial que ha dado lugar al proyecto, las restricciones que tiene la solución, y el impacto que se espera de la misma.

7.1. Hipótesis

Como se ha comentado en el capítulo 3, la principal hipótesis es que las tecnologías de conexión punto a punto no están tan extendidas como cabría esperar, a pesar de sus características muy beneficiosas para el usuario final (resiliencia, privacidad).

La principal hipótesis del trabajo es que una interfaz más sencilla de utilizar, que abstraiga los detalles profundos del direccionamiento, podría aumentar la adopción de estas tecnologías.

7.2. Restricciones

En esta sección se detallan las limitaciones que han motivado las decisiones tomadas durante el desarrollo del proyecto. Se han elaborado a partir de los requisitos no funcionales, especificados en el apartado 6.2, y más en detalle en el anexo 1.

7.2.1. Restricciones técnicas: entorno de explotación

La biblioteca ha de ser multiplataforma, lo cual ha supuesto una serie de restricciones técnicas, como el uso de lenguajes que soporten todas las plataformas necesarias.

Similarmente, el requisito de que funcione en plataformas móviles fuerza a usar determinadas APIs de esos sistemas operativos.

7.2.2. Restricciones técnicas: Transporte inicial y limitaciones del sistema operativo

A la hora de realizar la elección de qué capa de transporte usar inicialmente para esta biblioteca, se eligió WiFi Direct por una variedad de razones:

- Disponibilidad en Android [26] y Linux [32].
- Mayor rango de alcance comparado con Bluetooth y Bluetooth LE [8].
- Soporte para varios grupos físicos en la misma tarjeta Wi-Fi [6], lo cual permite en teoría extender el alcance de la red infinitamente, dados los suficientes nodos intermedios.

Sin embargo, Android no permite a un mismo dispositivo estar conectado a dos grupos de WiFi-Direct a la vez (a pesar de usar wpa_supplicant, que lo soporta, y tener una interfaz interna, WifiP2pGroupList[1]), aunque es un área activa de investigación [49].

Una línea de trabajo futura muy interesante que parece factible podría ser expandir la red usando una variedad de transportes físicos (implementar Bluetooth, y usar Bluetooth como conexión entre dos grupos).

7.2.3. Restricciones técnicas: Acceso a identificadores

Otra restricción interesante que cambió el diseño de la interfaz es que Android restringe el acceso a la dirección MAC del dispositivo[2] con un permiso de sistema (que aplicaciones normales no pueden solicitar).

Similarmente, wpa_supplicant no soportaba exponer esta dirección directamente, aunque se ha enviado un parche[3] y tests para hacerlo.

Esta ID sería útil, porque es la que otros dispositivos y la capa de transporte ven, pero como compromiso, la biblioteca soporta asociarse por nombre inicialmente (aunque eso por supuesto tiene más posibilidades de colisiones).

[1] WifiP2pGroupList: `https://cs.android.com/android/platform/superproject/main/+/main:packages/modules/Wifi/framework/java/android/net/wifi/p2p/WifiP2pGroupList.java;drc=9767925c3dbc08eeb6990a7e1109b916910b846c`

[2] restringe el acceso a la dirección MAC del dispositivo: `https://cs.android.com/android/platform/superproject/main/+/main:packages/modules/Wifi/service/java/com/android/server/wifi/p2p/WifiP2pServiceImpl.java;l=7502;drc=61197364367c9e404c7da6900658f1b16c42d0da`

[3] parche: `https://lists.infradead.org/pipermail/hostap/2025-May/043428.html`

7.2.4. Restricciones técnicas: Permisos en Linux

El ecosistema de Linux es muy variado, y no se ha hecho un estudio exhaustivo sobre qué distribuciones limitan el acceso por D-Bus a `wpa_supplicant`, pero al menos Arch Linux limita el acceso a `root` por defecto:

```
<!DOCTYPE busconfig PUBLIC ...>
<busconfig>
  <policy user="root">
    <allow own="fi.w1.wpa_supplicant1"/>
    <allow send_destination="fi.w1.wpa_supplicant1"/>
    <allow send_interface="fi.w1.wpa_supplicant1"/>
    <allow receive_sender="fi.w1.wpa_supplicant1"
     ↪  receive_type="signal"/>
  </policy>
  <policy context="default">
    <deny own="fi.w1.wpa_supplicant1"/>
    <deny send_destination="fi.w1.wpa_supplicant1"/>
    <deny receive_sender="fi.w1.wpa_supplicant1"
     ↪  receive_type="signal"/>
  </policy>
</busconfig>
```

Por lo tanto para acceder a esas APIs desde user-space se ha tenido que añadir algo como:

```
<policy group="wheel">
  <allow own="fi.w1.wpa_supplicant1"/>
  <allow send_destination="fi.w1.wpa_supplicant1"/>
  <allow send_interface="fi.w1.wpa_supplicant1"/>
  <allow receive_sender="fi.w1.wpa_supplicant1"
   ↪  receive_type="signal"/>
</policy>
```

Para permitir el acceso a todos los usuarios del grupo `wheel`. Otras alternativas serían usar un `dbus-daemon` diferente, como se ha hecho para testear localmente con múltiples instancias.

7.2.5. Restricciones técnicas: DHCP

WiFi Direct no asigna ninguna dirección IP en sí una vez se crea el grupo físico. Eso quiere decir que es el dispositivo el que tiene que saber de alguna manera la dirección del resto de los nodos.

IPv6 permite descubrir vecinos via NDP [48], lo cual parecía prometedor, pero los paquetes necesarios solo pueden ser creados por aplicaciones privilegiadas via

`CAP_NET_RAW`[4].

Tras investigar cómo Android conseguía comunicar la dirección IP del *Group Owner*, se se descubrió la opción de usar usar direcciones de link local de IPv6 [36] (sección 5.3).

Sin embargo, algunos servidores DHCP rompen esto por defecto, como dhcpcd[5]. El autor sugirió una mejora[6] para facilitar la configuración correcta a este proyecto que se aceptó y resolvió rápidamente.

7.2.6. Restricciones de usabilidad: Permisos en Android

Usar WiFi Direct en Android requiere amplios permisos[7], y tener los servicios de ubicación activados.

Esto fuerza a la biblioteca a depender más del contexto de la aplicación en Android, y a la inicialización a ser asíncrona, ya que estos permisos requieren interacción del usuario.

La hipótesis para requerir esto es que técnicamente puedes usar el escaneo de redes WiFi para geolocalización, usando bases de datos como BeaconDB[8].

7.2.7. Restricciones de usabilidad: Interacción del usuario

Idealmente, para casos de uso como crear redes ad-hoc, la biblioteca o aplicación que la use podría anunciarse y conectarse a dispositivos sin interacción.

Sin embargo Android no soporta ese caso de uso, y requiere una interacción la primera vez que intentas conectarte a un dispositivo. Por lo tanto la información que la biblioteca puede exponer inicialmente sobre el dispositivo es mucho más limitada.

7.2.8. Restricciones de usabilidad: Interacción entre `wpa_supplicant` y `NetworkManager`

Actualmente, usar la biblioteca via `wpa_supplicant` requiere desactivar `NetworkManager`[9]. Esto es porque `NetworkManager` ve una interfaz que no conoce y la desactiva. Se ha reportado[10] al proyecto, y se colaborará para llegar a una solución.

[4] `CAP_NET_RAW`: `https://man7.org/linux/man-pages/man7/capabilities.7.html`
[5] dhcpcd: `https://github.com/NetworkConfiguration/dhcpcd`
[6] mejora: `https://github.com/NetworkConfiguration/dhcpcd/issues/473`
[7] requiere amplios permisos: `https://developer.android.com/develop/connectivity/wifi/wifi-direct#permissions`
[8] BeaconDB: `https://beacondb.net/`
[9] `NetworkManager`: `https://networkmanager.dev`
[10] reportado: `https://gitlab.freedesktop.org/NetworkManager/NetworkManager/-/issues/1804`

7.3. Alcance funcional del proyecto e impacto esperado

Dado a la duración del proyecto y las limitaciones temporales, el alcance del proyecto es limitado.

El proyecto cumple la gran mayoría de los objetivos propuestos, incluida una demo funcional en Android, pero hay mucha funcionalidad y mejoras extra que no han podido hacerse por falta de tiempo y por las varias restricciones descritas anteriormente, que han consumido más tiempo de desarrollo del esperado.

De ellas, los grupos lógicos es la mayor omisión, pero se pueden implementar sobre la infraestructura existente sin demasiado esfuerzo, o incluso por encima de la biblioteca.

En términos de la adopción esperada, lo ideal sería que esta biblioteca se convirtiera en un proyecto viable para la adopción de este tipo de tecnologías a gran escala.

En la práctica, la biblioteca es bastante útil en su estado actual para aplicaciones P2P sencillas. Sin embargo, hacerla útil para redes a gran escala requeriría bastante trabajo extra, descrito a continuación.

Linux

- Coordinación con `NetworkManager` para evitar interacciones no deseadas.
- Coordinación con distribuciones para proveer acceso a este mecanismo a la biblioteca. Esto probablemente requiera una aplicación o servicio "privilegiado" para gestionar las conexiones.
- Coordinación con distribuciones para que los servidores DHCP usen direcciones de link local por defecto para conexiones P2P o formas alternativas de descubrir direcciones como NDP, ver apartado 7.2.5.

Android:

- Implementación de múltiples grupos físicos via Bluetooth, o
- Extensión y mejoras de las APIs de Android para soportar múltiples grupos físicos. Esto es probable que sea difícil (aunque técnicamente se pueden enviar parches a Android[11], la tendencia ha sido a hacer el desarrollo más opaco[12].

[11] enviar parches a Android: `https://source.android.com/docs/setup/contribute/submit-patches`

[12] hacer el desarrollo más opaco: `https://www.androidauthority.com/google-android-development-aosp-3538503/`

Otras plataformas

- Añadir soporte para Windows[13] debería ser posible y razonablemente fácil.
- Añadir soporte para macOS e iOS, que no soportan WiFi Direct y por lo tanto requerirían Bluetooth para comunicarse con otras plataformas.

[13] Windows: `https://learn.microsoft.com/en-us/windows/win32/nativewifi/using-the-wi-fi-direct-api`

Capítulo 8

Estudio de alternativas y viabilidad

En este capítulo se detallan las alternativas consideradas al tomar varias decisiones, tanto técnicas como no. Finalmente se realiza un pequeño comentario sobre la viabilidad del proyecto.

8.1. Alternativas consideradas

8.1.1. Lenguaje de programación

La elección de Rust ha sido explicada en el apartado 5.2.1.

Se consideró C y C++ como lenguajes con similar rendimiento a Rust en tiempo de ejecución y buena integración con otras plataformas, pero varias características de Rust decantaron la balanza:

- Gestión de paquetes: Rust tiene un ecosistema de paquetes enorme[1], y un gestor de paquetes integrado. Adquirir dependencias complejas con C y C++, especialmente cuando estás compilando para diferentes plataformas, es mucho más complejo.
- Sistema de tipos: El sistema de tipos de Rust es muy flexible, y previene muchos errores comunes en C y C++. El soporte para programación asíncrona[2] en el lenguaje es muy útil y conveniente cuando se programa código de red, que por naturaleza es asíncrono.
- Características de seguridad: Saber que tu código no tiene data races no tiene precio y ayuda con la depuración inmensamente.
- Compilación cruzada fácil: Rust provee *toolchains* pre-compiladas para una gran variedad de plataformas, y compilar para otra plataforma es tan sencillo como hacer `cargo build --target=x86_64-linux-android`.

[1] ecosistema de paquetes enorme: `https://crates.io`

[2] soporte para programación asíncrona: `https://rust-lang.github.io/async-book/`

Se consideraron otras alternativa a estos tres lenguajes también:

- Zig[3]: A pesar de que tiene funcionalidad muy interesante, sobre todo con respecto a la metaprogramación[4], no tiene las características de seguridad de Rust, y el autor no conoce el lenguaje tan profundamente, lo que hubiera supuesto un desafío extra.
- Java[5]: Depender de la JVM dificulta usarlo en plataformas como iOS, ya que Apple no permite la generación de código dinámico en esa plataforma [27]. OpenJDK tenía un proyecto para soportar iOS[6], pero se quedó en un experimento en JDK 9 y parece estar muerto.
- Kotlin[7]: Tiene la misma restricción que Java, pero Kotlin Native[8] es una alternativa más reciente. En cualquier caso, ese tipo de problemas lo hacen más complejo que usar un lenguaje completamente compilado como C, C++, o Rust.

8.1.2. Capa de transporte

Para la capa de transporte, sólo Bluetooth y WiFi Direct son alternativas estándar en hardware de consumo. Bluetooth está disponible también en plataformas Apple, pero tiene un rango menor.

Alternativas no estándar incluyen AWDL de Apple y Sparklink[9] de Huawei. Dado que el desarrollo no iba a poder realizarse en dichas plataformas de todas maneras (porque el autor no tiene acceso a ellas), se eligió WiFi Direct como la plataforma más prometedora.

WiFi Aware, también conocido como *Neighbor Awareness Networking* [7] es la siguiente iteración de la WiFi Alliance sobre WiFi Direct, y funciona en Android[10], pero no en ninguna otra plataforma.

Dado el panorama, se eligió WiFi direct porque sus características (como rango) eran más favorables que Bluetooth, y permitía ser implementado en múltiples plataformas.

Parece que la Unión Europea, via DMA, está presionando a Apple para abrir AWDL e implementar WiFi Aware [16] [17] (sec 2.1).

> Apple shall provide effective interoperability with the high-bandwidth peer-to-peer ("P2P") Wi-Fi connection feature.

[3]Zig: `https://ziglang.org/`

[4]sobre todo con respecto a la metaprogramación: `https://zig.guide/language-basics/comptime/`

[5]Java: `https://java.com/`

[6]proyecto para soportar iOS: `https://openjdk.org/projects/mobile/ios.html`

[7]Kotlin: `https://kotlinlang.org/`

[8]Kotlin Native: `https://kotlinlang.org/docs/native-overview.html`

[9]Sparklink: `https://www.sparklink.org.cn/`

[10]funciona en Android: `https://developer.android.com/develop/connectivity/wifi/wifi-aware`

[...]

Apple shall: * Implement Wi-Fi Aware in its iOS devices and iOS in accordance with the Wi-Fi Aware specification.

- Allow third-party iOS app developers to establish a Wi-Fi Aware connection between an iOS device and any third-party connected physical device that supports Wi-Fi Aware.

Y parece que Apple está implementando WiFi Aware[11] (actualmente en iOS Beta). Así que tal vez en un futuro no tan lejano los consumidores tengan el futuro que se merecen con WiFi Aware...

8.1.3. Criptografía

Para la criptografía se ha usado ring[12], una biblioteca de Brian Smith que usa código de BoringSSL[13] (que a su vez es un fork the OpenSSL[14]).

Se barajó usar las dos bibliotecas mencionadas anteriormente directamente, libsodium[15], RustCrypto[16], y cosas más avanzadas como mls-rs[17], que implementa el protocolo MLS [12].

Se eligió ring por su énfasis en ser difícil de usar mal. Dicho eso, la biblioteca criptográfica no es difícil de cambiar, y adaptar MLS u otra biblioteca no debería ser un desafío.

No se consideró implementar las primitivas criptográficas por la falta de tiempo y lo común que es hacerlo mal[18].

8.1.4. Toolkit de interfaz de usuario para Android

Se usó Jetpack Compose para la interfaz de usuario en Android (ver apartado 5.2.1).

Android soporta varias alternativas para interfaces de usuario, desde los Fragments[19] a otros toolkit multiplataforma como podrían ser Flutter[20] o React Native[21].

El autor solo estaba familiarizado superficialmente con estos toolkits, y la curva de aprendizaje iba a ser significativa en cualquier caso, así que se optó por una

[11] WiFi Aware: `https://developer.apple.com/documentation/WiFiAware`
[12] ring: `https://crates.io/crates/ring`
[13] BoringSSL: `https://boringssl.googlesource.com/boringssl`
[14] OpenSSL: `https://www.openssl.org/`
[15] libsodium: `https://doc.libsodium.org`
[16] RustCrypto: `https://github.com/rustcrypto`
[17] mls-rs: `https://crates.io/crates/mls-rs`
[18] común que es hacerlo mal: `https://security.stackexchange.com/questions/18197/why-shouldnt-we-roll-our-own`
[19] Fragments: `https://developer.android.com/guide/fragments`
[20] Flutter: `https://flutter.dev/`
[21] React Native: `https://reactnative.dev/`

solución en Kotlin para evitar problemas de integración con otros lenguajes (ya que la librería tenía que usar al menos Java para la interacción con el sistema).

Entre Fragments y Jetpack Compose, que son las dos alternativas que soportaban Kotlin o Java, se eligió el último por ser más moderno y activamente recomendado en la documentación oficial:

> Jetpack Compose is Android's recommended modern toolkit for building native UI with less code, powerful tools, and intuitive Kotlin APIs. [24]

8.1.5. Toolkit de interfaz de usuario para Linux

Se usó GTK para la interfaz de usuario en Linux (ver apartado 5.2.1).

La demo de Linux intencionadamente es extremadamente sencilla, siendo su mayor utilidad probar el protocolo sin tener que compilar Android y usar varios dispositivos. Por lo tanto cualquier toolkit podría proporcionar la funcionalidad necesaria, y se priorizó facilidad y familiaridad del autor sobre otros factores como rendimiento y funcionalidad.

En cuestión de toolkits populares que soporten Linux, los más populares son GTK y Qt[22]. Sin embargo, la comunidad de GNOME proporciona bindings para Rust[23] listos para usar, y el autor es un contribuidor ocasional a GTK[24], lo cual lo hacían una elección mucho más fácil.

8.1.6. Control de versiones y alojamiento

Para el control de versiones del proyecto se ha utilizado Git (ver apartado 5.2.4) y para el alojamiento del código GitHub (ver apartado 5.2.4) y un mirror[25] propio para compartir progreso con los tutores (ya que el repositorio era inicialmente privado).

El autor conoce también Mercurial[26], y de hecho considera a Mercurial superior en muchos aspectos a Git. Sin embargo, Git es mucho más utilizado en la industria [39], el autor también lo conoce en profundidad, y existen herramientas como Jujutsu[27] inspiradas en la experiencia de usuario de Mercurial. Eso, junto a la existencia de servicios para alojar el código como GitHub, hicieron a Git la elección simple.

Tampoco se ha planteado no utilizar un sistema de control de versiones, ya que es una herramienta esencial para el desarrollo de software, y permite llevar un seguimiento de los cambios realizados en el código, así como colaborar con otros desarrolladores de forma más eficiente en un futuro.

[22] Qt: `https://www.qt.io/platform/desktop-app-development`

[23] bindings para Rust: `https://gtk-rs.org/gtk4-rs/stable/latest/book/introduction.html`

[24] contribuidor ocasional a GTK: `https://gitlab.gnome.org/GNOME/gtk/-/merge_requests/?sort=closed_at_desc&state=all&author_username=emilio`

[25] mirror: `https://crisal.io/git/?p=ngn.git;a=summary`

[26] Mercurial: `https://www.mercurial-scm.org/`

[27] Jujutsu: `https://jj-vcs.github.io/jj/latest/`

8.1.7. Documentación

Para la documentación del proyecto se ha utilizado una mezcla de LaTeX(apartado 5.2.3) y Pandoc (apartado 5.2.3).

Se consideró usar una alternativa más moderna al sistema de macros de LaTeXllamada Typst[28], pero el conocimiento previo de LaTeXdel autor, la Transpilación desde Markdown, la más amplia documentación de LaTeX, y la falta de tiempo hicieron que no mereciera la pena.

Se podría haber utilizado cualquier procesador de textos como LibreOffice Writer[29] o Google Docs[30], pero usar LaTeXy Markdown se consideró una ventaja por la mejor calidad, y la capacidad de usar el mismo sistema de control de versiones para el texto.

8.2. Coste del proyecto

No se ha recibido ninguna compensación por el desarrollo del proyecto, pero haciendo una estimación conservadora de las horas empleadas en el trabajo, incluyendo el desarrollo de la memoria y documentación, de aproximadamente 1 hora de trabajo por cada *commit*, y el salario que el autor recibe como parte de su trabajo habitual como *Senior Staff Software Engineer*:

$$\text{Coste} = 140h \times 88€/h = 12320€$$

Dado a que el coste de herramientas es 0€, ya que todas son herramientas de software libre, ese sería también el coste total.

Dicho eso, sería posible que el tiempo de desarrollo y coste hubieran sido menores sin las limitaciones temporales que han aplacado a este trabajo.

El mantenimiento del proyecto en si no tiene costes asociados, dado el uso de plataformas gratuitas para el alojamiento de código. Eventualmente sería ideal tener integración continua con dispositivos reales, que no es fácil de conseguir gratis, pero en cualquier caso ese coste sería opcional.

8.2.1. Monetización

No se prevé una monetización de este proyecto ya que es software libre distribuido gratuitamente. Si consiguiera la adopción deseada, sería tal vez posible monetizarlo con contratos de soporte o desarrollo, o tal vez relicenciando el código para compañías con productos propietarios que no quieran adherirse a los términos de la GPL.

[28] Typst: `https://typst.app/`

[29] LibreOffice Writer: `https://www.libreoffice.org/discover/writer`

[30] Google Docs: `https://docs.google.com`

Sin embargo, el autor no es demasiado optimista al respecto, ya que mantiene software muy utilizado desinteresadamente (si bien es cierto que nunca ha tratado de obtener rédito económico de ello).

Capítulo 9

Descripción de la solución propuesta

En esta sección se presenta el software resultante del proyecto, se describirá su funcionalidad, y se proveerán ejemplos de uso. Se proporcionará acceso a una demo para Android.

9.1. Acceso

Siendo una biblioteca de software, el producto principal está en el repositorio, accesible tanto en GitHub[1], como en el mirror personal[2].

La demo de Android se puede compilar con Android Studio, o descargar desde GitHub Releases[3].

9.2. Estructura general del proyecto

A continuación se expone una visión simplificada de la estructura del proyecto:

```
1  ├── Cargo.toml
2  ├── doc
3  ├── examples
4  │   ├── android
5  │   └── dbus
6  ├── LICENSE
7  ├── README.md
8  ├── src
9  │   ├── lib.rs
10 │   ├── platform
11 │   │   ├── android
```

[1]GitHub: `https://github.com/emilio/ngn`
[2]mirror personal: `https://crisal.io/git/?p=ngn.git;a=summary`
[3]GitHub Releases: `https://github.com/emilio/ngn/releases`

```
│   │   │   ├── mod.rs
│   │   │   └── src/main/java/io/crisal/ngn
│   │   │       ├── NgnListener.kt
│   │   │       └── NgnSessionProxy.java
│   │   ├── dbus
│   │   │   ├── mod.rs
│   │   │   ├── store.rs
│   │   │   └── wpa_supplicant
│   │   └── mod.rs
│   ├── protocol
│   │   ├── encryption.rs
│   │   ├── identity.rs
│   │   ├── key_exchange.rs
│   │   ├── mod.rs
│   │   └── signing.rs
│   └── utils.rs
└── test
    ├── dbus-system-bus-mock.conf
    ├── setup-android.sh
    ├── setup.sh
    └── simple.conf
```

Todo el proyecto es parte del mismo paquete de `cargo`, definido en `Cargo.toml`. Ahí es donde los datos básicos y dependencias están declaradas:

```toml
[package]
name = "ngn"
version = "0.1.0"
edition = "..."
license = "..."
# ...

[lib]
name = "ngn"
crate-type = ["cdylib", "lib"]

[dependencies]
tokio = { version = "1", features = ["full"] }
# ...
[target.'cfg(target_os = "android")'.dependencies]
jni = "0.21"
# ...
```

También donde se declaran la estructura y dependencias de la demo de Linux, que vive en `examples/dbus`:

```toml
[dev-dependencies]
```

```
gtk = { version = "0.9.6", package = "gtk4", features = ["v4_18"] }
adw = { version = "0.7.2", package = "libadwaita", features = ["v1_4"] }

[[example]]
name = "dbus"
crate-type = ["bin"]
```

El código de Android también se divide en dos. La biblioteca, en `src/platform/android`, con su parte de Java / Kotlin en `src/platform/android/src/main/java/io/crisal/ngn`, y la aplicación de demostración en `examples/android`.

Por conveniencia, se ha usado `tokio`[4] como *runtime*[5] asíncrona. El uso de tokio en la biblioteca no es particularmente especial y se podrían soportar varias *runtimes* sin problema.

9.3. Interfaces y estructuras principales

9.3.1. `P2PSession` y `P2PSessionListener`

La interfaz principal de la biblioteca está en `src/lib.rs`, donde se define el Trait `P2PSession`, cuya implementación varía por plataforma, y es la que expone métodos para iniciar el descubrimiento de dispositivos (`discover_peers`), conectarse (`connect_to_peer`) y enviar mensajes (`message_peer`):

```rust
#[async_trait::async_trait]
pub trait P2PSession: Sized + Debug + Send + Sync + 'static {
    async fn new(
        args: Self::InitArgs<'_>,
        listener: Arc<dyn P2PSessionListener<Self>>,
    ) -> GenericResult<Arc<Self>>;
    async fn stop(&self) -> GenericResult<()>;
    async fn wait(&self) -> GenericResult<()>;
    async fn discover_peers(&self) -> GenericResult<()>;
    fn peer_identity(&self, id: PeerId) -> Option<protocol::PeerIdentity>;
    fn all_peers(&self) -> Vec<(PeerId, protocol::PeerIdentity)>;
    fn own_identity(&self) -> &protocol::identity::OwnIdentity;
    async fn connect_to_peer(&self, id: PeerId) -> GenericResult<()>;
    async fn message_peer(&self, id: PeerId, message: &[u8]) -> GenericResult<()>;
```

[4] tokio: https://tokio.rs/
[5] runtime: https://www.ncameron.org/blog/what-is-an-async-runtime/

```
}
```

La inicialización de la sesión requiere un `P2PSessionListener`, que es la forma de reaccionar a cambios de manera asíncrona. La implementación por defecto simplemente loguea los eventos.

```
pub trait P2PSessionListener<S: P2PSession>: Debug + Send + Sync {
    fn peer_discovered(&self, _: &S, peer_id: PeerId);
    fn peer_lost(&self, _: &S, peer_id: PeerId);
    fn peer_discovery_stopped(&self, _: &S);
    fn joined_group(&self, _: &S, group_id: GroupId, is_go: bool);
    fn left_group(&self, _: &S, group_id: GroupId, is_go: bool);
    fn peer_joined_group(&self, _: &S, group_id: GroupId, peer_id: PeerId);
    fn peer_left_group(&self, _: &S, group_id: GroupId, peer_id: PeerId);
    fn peer_messaged(&self, _: &S, peer_id: PeerId, group_id: GroupId, message: &[u8]);
}
```

9.3.2. Identificación de dispositivos y grupos

Los identificadores que se usan para el enrutamiento de mensajes (`PeerId` y `GroupId`) son independientes de la capa de transporte y plataforma. Son simplemente un *handle* de 64 bits:

```
#[repr(transparent)]
#[derive(Copy, Clone, Debug, Eq, PartialEq, PartialOrd, Ord, Hash)]
pub struct PeerId(pub(crate) handy::Handle);

#[repr(transparent)]
#[derive(Copy, Clone, Debug, Eq, PartialEq, PartialOrd, Ord, Hash)]
pub struct GroupId(pub(crate) handy::Handle);
```

El enrutamiento de mensajes a un `PeerId` es independiente del grupo físico (`GroupId`) al que está conectado. El grupo físico se expone ahora mismo en el `Listener`, pero es probable que se elimine porque no es necesario y es probable que en otras capas de transporte no haya múltiples grupos.

La identidad *lógica* de un dispositivo (independiente de la capa de transporte) es simplemente un *nick* (nombre) y una clave criptográfica:

```
#[derive(Debug)]
```

```rust
pub struct OwnIdentity {
    pub nickname: String,
    pub key_pair: KeyPair,
}

#[derive(Encode, Decode, Debug, PartialEq, Eq, Clone)]
pub struct LogicalPeerIdentity {
    pub nickname: String,
    pub key: MaybeInvalidPublicKey,
}
```

9.4. Enrutado de mensajes en WiFi Direct

Cuando se forma un grupo de WiFi Direct, hay efectivamente dos modos de operación. En cada grupo, hay un GO o *Group Owner*, y el resto de miembros son clientes.

Los clientes se pueden comunicar entre sí sin pasar por el GO **una vez sepan su dirección IP**, pero es el GO el que tiene que encargarse de comunicar la existencia de nuevos miembros a los existentes.

Durante la formación del grupo, la única dirección IP que podemos saber por adelantado en todas las plataformas es la dirección del GO. En el mundo ideal, podríamos depender the las direcciones de link local [36], para obtener una dirección dada la dirección MAC de la interfaz, pero Android no la expone, y aunque puedes solicitar el uso de IPv6[6], es una API bastante reciente.

Por lo tanto la solución que se adoptó establece **dos canales** por cada cliente, un canal de *control* no cifrado para la gestión del grupo e intercambio de claves, y uno para la comunicación cifrada y firmada de mensajes entre dispositivos.

El único puerto que tiene que ser conocido de antemano es el del canal de control del *Group Owner*, el cual está definido en `src/protocol/mod.rs`:

```rust
/// The port the GO of the group listens to.
pub const GO_CONTROL_PORT: u16 = 9001;
```

El resto de puertos son dinámicos.

9.5. Criptografía

La solución propuesta implementa cifrado punto a punto usando el cifrado simétrico AES-256-GCM, con un intercambio de claves usando ECDH con el algoritmo X25519 [31] para la generación de claves efímeras.

[6]solicitar el uso de IPv6: `https://developer.android.com/reference/android/net/wifi/p2p/WifiP2pConfig#GROUP_CLIENT_IP_PROVISIONING_MODE_IPV6_LINK_LOCAL`

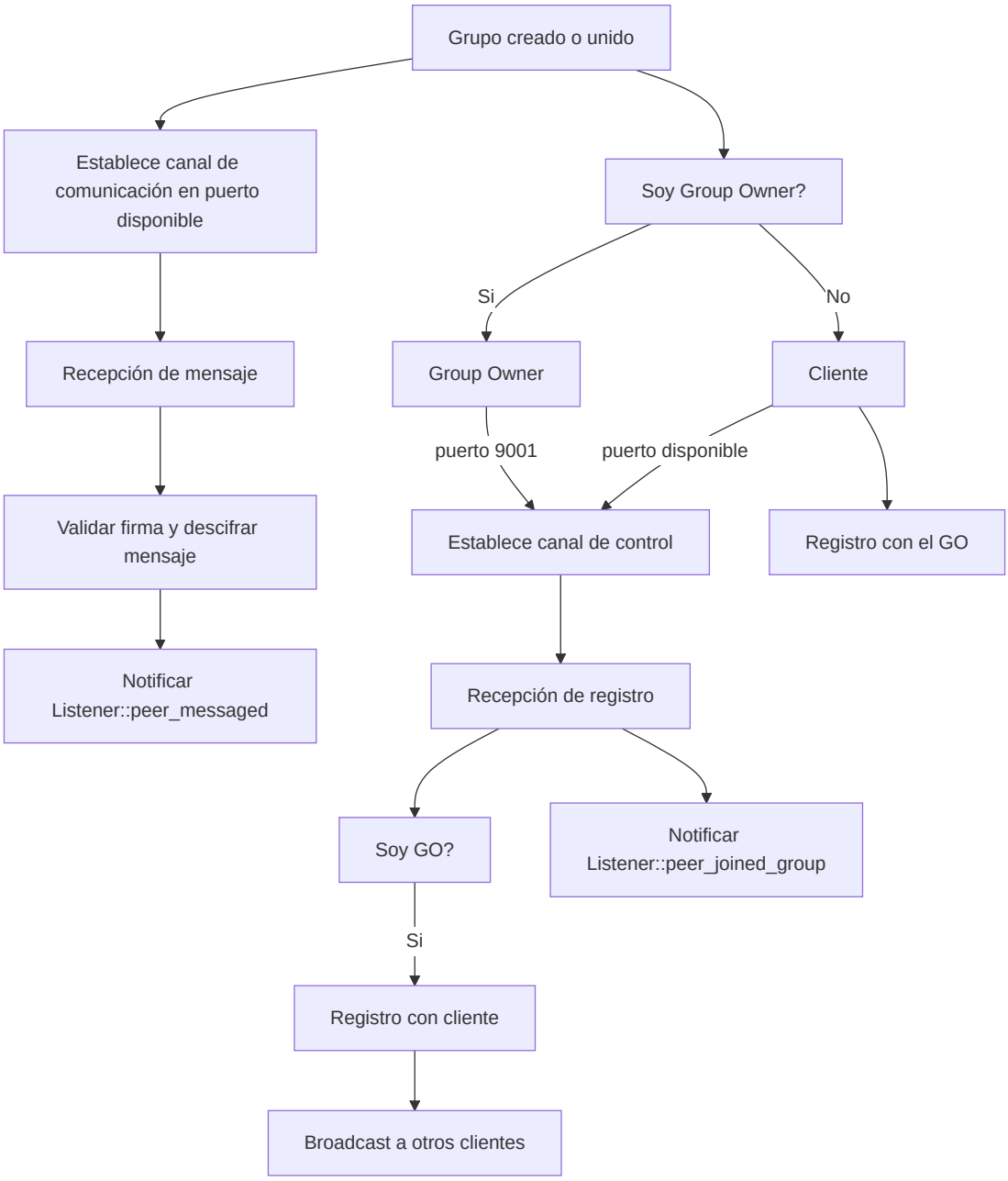

Figura 9.1: Flujo de control al conectarse a un nuevo grupo

Adicionalmente, los mensajes entre los clientes están firmados con su clave de identidad, que es una clave ed25519[7].

Nótese que esta firma es innecesaria para la seguridad de la transmisión, ya que que el mensaje está intacto también es garantizado por el algoritmo de cifrado. Sin embargo, garantiza que quien lo envía es quien dice ser, en caso de que hubiera un MITM durante el intercambio de claves.

A pesar de que en el prototipo las claves de identidad son efímeras por conveniencia (no se ha implementado persistencia ni una base de datos de identidades conocidas), la idea es que estas claves ed25519 pudieran ser persistentes.

El cifrado (una vez se han acordado las claves correspondientes) y la firma de los mensajes son relativamente sencillos, por lo que no se indagará mucho más en profundidad en ellos en esta sección. Viven en `src/protocol/encryption.rs` y `src/protocol/signature.rs`, respectivamente.

La clave de cifrado es efímera para cada sesión de comunicación entre dos clientes, y está generada tras el proceso de intercambio de claves descrito en apartado 9.5.1.

La clave de firma se especifica durante la creación de la sesión y su parte pública se envía como parte del mensaje de asociación a cualquiera de los clientes.

9.5.1. Intercambio de claves

El código que encapsula el intercambio de claves vive en `src/protocol/key_exchange.rs`. El sistema de tipos de Rust garantiza que no podamos usar una clave efímera (`EphemeralPrivateKey`) para más de una operación de intercambio.

```
1  #[derive(Debug)]
2  enum KeyExchangeState {
3      InProgress(PrivateKey),
4      Completed(Arc<super::encryption::Keys>),
5      Errored,
6  }
7
8  #[derive(Debug)]
9  pub struct KeyExchange {
10     public_key: PublicKey,
11     state: State,
12 }
```

Cuando se descubre un cliente nuevo, se crea un objeto `KeyExchange` en el estado `InProgress`, con una clave privada:

```
1  impl KeyExchange {
2      pub fn new() -> Result<Self, Unspecified> {
```

[7] ed25519: `https://ed25519.cr.yp.to/`

```
        let private =
        ↪ ring::agreement::EphemeralPrivateKey::generate(
            &X25519,
            &ring::rand::SystemRandom::new(),
        )?;
        let public_key = private.compute_public_key()?;
        Ok(Self {
            public_key,
            state: State::InProgress(private),
        })
    }
}
```

La clave pública se exporta en el mensaje de asociación:

```
impl KeyExchange {
    pub fn export_public_key(&self) -> MaybeInvalidPublicKey {
        ↪ MaybeInvalidPublicKey(self.public_key.as_ref().try_into().unwrap())
    }
}
```

Cuando recibimos la clave del otro cliente, se reemplaza el estado `InProgress` con `Errored` (en caso de algún error criptográfico), o `Completed`, con la clave AES256 usada para cifrar y descifrar mensajes:

```
impl KeyExchange {
    pub fn finish(&mut self, peer_key: &MaybeInvalidPublicKey)
    ↪ -> GenericResult<()> {
        if !matches!(self.state,
        ↪ KeyExchangeState::InProgress(..)) {
            return Err(trivial_error!("Exchange already
            ↪ completed"));
        }
        let result = std::mem::replace(&mut self.state,
        ↪ State::Errored);
        self.state = match result {
            KeyExchangeState::InProgress(private) => {
                let peer_key = UnparsedPublicKey::new(
                    &X25519,
                    &peer_key.0[..]
                );
                State::Completed(Arc::new(
                    Keys::from_shared_secret(private, peer_key)?
                ))
            }
            _ => unreachable!(),
```

```
18          };
19          Ok(())
20      }
21  }
```

9.6. Formato y envío de mensajes

El formato de mensaje es común para tanto el canal de control como el de comunicación.

Consiste en una cabecera de 64 bits con un número mágico de 16 bits (`0xdead`), un número de version de 16 bits (del cual se podrían usar 8 para flags varias, ya que la versión actual sería siempre `1`), y la longitud del mensaje binario en 32 bits.

Los mensajes del canal de control no van cifrados ni firmados, mientras que los mensajes entre clientes sí. Las rutinas de envío de mensajes se encargan de cifrar/descifrar y firmar/validar el mensaje de forma transparente.

Las rutinas de envío y recepción de mensajes son totalmente ajenas al formato que usen los clientes. De hecho, la demo de Linux intercambia cadenas de caracteres planas, y la demo de Android intercambia JSON, por ejemplo.

Para el canal de control se ha utilizado bincode[8] para codificar los mensajes de control, ya que junto a las capacidades de meta-programación de Rust permite derivar la codificación de los mensajes:

```
1  #[derive(Encode, Decode, Debug)]
2  pub enum ControlMessage {
3      // ...
4  }
5
6  // ...
7
8  #[derive(Encode, Decode, Debug, Clone)]
9  pub struct DecodableMacAddr {
10     is_v8: bool,
11     bytes: [u8; 8],
12 }
```

Esto facilita los cambios en la fase de prototipado, y evita errores innecesarios que inevitablemente pasan de otra forma.

[8] bincode: `https://crates.io/crates/bincode`

9.7. Interoperabilidad con Java / Android

Para interactuar con las APIs del sistema de Android, debemos usar la JNI. Esta interfaz permite a programas en C llamar a Java, y vice versa.

Rust, convenientemente, también soporta la interfaz de llamadas de C[9]. Se ha usado una librería pre-existente para usar tipos algo más convenientes llamada jni[10].

Un ejemplo sencillo podría ser cómo inicializamos la sesión nativa. `ngn_session_init` devuelve un puntero nativo (en un `jlong`), y `ngn_session_drop` lo destruye desde java cuando ya no es necesario:

```
impl Session {
    #[export_name =
    ↪ "Java_io_crisal_ngn_NgnSessionProxy_ngn_1session_1init"]
    extern "C" fn init<'l>(
        mut env: JNIEnv<'l>,
        _class: JClass<'l>,
        owner: JObject<'l>,
        device_name: JString<'l>,
        nickname: JString<'l>,
    ) -> jlong {
        let session = Self::new_sync(init,
        ↪ Arc::new(crate::LoggerListener));
        Arc::into_raw(session) as jlong
    }

    /// Breaks the cyclic owner <-> native listener.
    #[export_name =
    ↪ "Java_io_crisal_ngn_NgnSessionProxy_ngn_1session_1drop"]
    extern "C" fn drop<'l>(_env: JNIEnv<'l>, _class: JClass<'l>,
    ↪ raw: jlong) {
        trace!("Session::drop({raw:?})");
        let _ = unsafe { Arc::from_raw(raw as *const Self) };
    }
}
```

Desde Java, se utiliza como una variable más:

```
public class NgnSessionProxy ... {
    @Override
    public void onDeviceInfoAvailable(@Nullable WifiP2pDevice
    ↪ wifiP2pDevice) {
        // ...
```

[9] interfaz de llamadas de C: `https://doc.rust-lang.org/book/ch20-01-unsafe-rust.html#using-extern-functions-to-call-external-code`

[10] jni: `https://docs.rs/jni`

```
5           m_native = ngn_session_init(this,
            ↪ wifiP2pDevice.deviceName, m_nickName);
6       }
7
8       public void messagePeer(String aMacAddress, byte[] aMessage,
        ↪ Function<Boolean, Void> onResult) {
9           // ...
10          ngn_session_message_peer(m_native, aMacAddress,
            ↪ aMessage, onResult);
11      }
12
13      @Override
14      protected void finalize() {
15          if (m_native == 0) {
16              return; // Already finalized, or not initialized.
17          }
18          ngn_session_drop(m_native);
19      }
20  }
```

9.7.1. Notificación de eventos de Java a Rust

Como Rust usa su propio bucle de eventos via tokio, se ha usado paso de mensajes para comunicar notificaciones del sistema. Se utiliza un enum llamado `JavaNotification` que el bucle de eventos de Rust recibe via un canal MPSC en `Session::run_loop`:

```
1   /// Representation of system messages that we need to handle
2   #[derive(Debug)]
3   enum JavaNotification {
4       UpdateDevices(Vec<PhysiscalPeerIdentity>),
5       GroupStarted {
6           iface_name: String,
7           is_go: bool,
8           go_device_address: MacAddr,
9           go_ip_address: IpAddr,
10      },
11      // ...
12  }
13
14  #[derive(Debug)]
15  pub struct Session {
16      proxy: GlobalRef,
17      // ...
18      java_notification: mpsc::UnboundedSender<JavaNotification>,
19      /// Task handle to our run loop. Canceled and awaited on
        ↪ drop.
```

```
    run_loop_task:
    ↪ RwLock<Option<JoinHandle<GenericResult<()>>>>,
}

impl Session {
    async fn run_loop(
        session: Arc<Self>,
        mut rx: mpsc::UnboundedReceiver<JavaNotification>,
    ) -> GenericResult<()> {
        trace!("Session::run_loop");
        while let Some(message) = rx.recv().await {
            match message { ... }
        }
    }
}
```

A continuación se muestra el flujo de una notificación del sistema de Android como un cambio en la lista de dispositivos disponibles.

9.7.2. Notificación de eventos de Rust a Java

Rust almacena una referencia a la clase `NgnSessionProxy` cuando se crea la sesión nativa, y pueded llamar a Java desde cualquier hilo.

Por ejemplo, cuando se recibe un mensaje, Rust invoca el método `NgnSessionProxy.peerChanged` dinámicamente:

```
impl Session {
    fn peer_messaged_internal(
        &self,
        peer_id: PeerId,
        group_id: GroupId,
        buf: &[u8],
    ) -> GenericResult<()> {
        self.listener.peer_messaged(self, peer_id, group_id,
        ↪ buf);
        let mut env = self.vm.attach_current_thread()?;
        let (peer_name, peer_dev_addr, peer_logical_id) = {
            // ...
        };
        let byte_array = env.byte_array_from_slice(buf)?;
        self.call_proxy(
            &mut env,

            ↪ "(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;[B)V",
            "peerMessaged",
            &[
```

```
                (&peer_name).into(),
                (&peer_dev_addr).into(),
                (&peer_logical_id).into(),
                (&byte_array).into(),
            ],
        )?;
        Ok(())
    }

    fn call_proxy<'local>(
        &self,
        env: &mut JNIEnv<'local>,
        sig: &'static str,
        method: &'static str,
        args: &[jni::objects::JValue],
    ) -> GenericResult<jni::objects::JValueOwned<'local>> {
        let result = env.call_method(self.proxy.as_obj(),
        ↪  method, sig, args)?;
        Ok(result)
    }
}
```

Este método de java es el que se encarga de procesar la notificación:

```
public class NgnSessionProxy ...{
    // NOTE: called via the JNI.
    private void peerMessaged(String name, String mac_addr,
    ↪  String logicalId, byte[] message) {
        m_listener.messageReceived(new Peer(name, mac_addr,
        ↪  logicalId), message);
    }
}
```

Es el código de Java el que es responsable de, si es necesario, cambiar al hilo principal (`activity.runOnUiThread`):

```
class Listener(val activity: MainActivity) : NgnListener() {
    override fun messageReceived(from: Peer, content: ByteArray)
    ↪  {
        super.messageReceived(from, content)
        activity.runOnUiThread {
            // ...
        }
    }
}
```

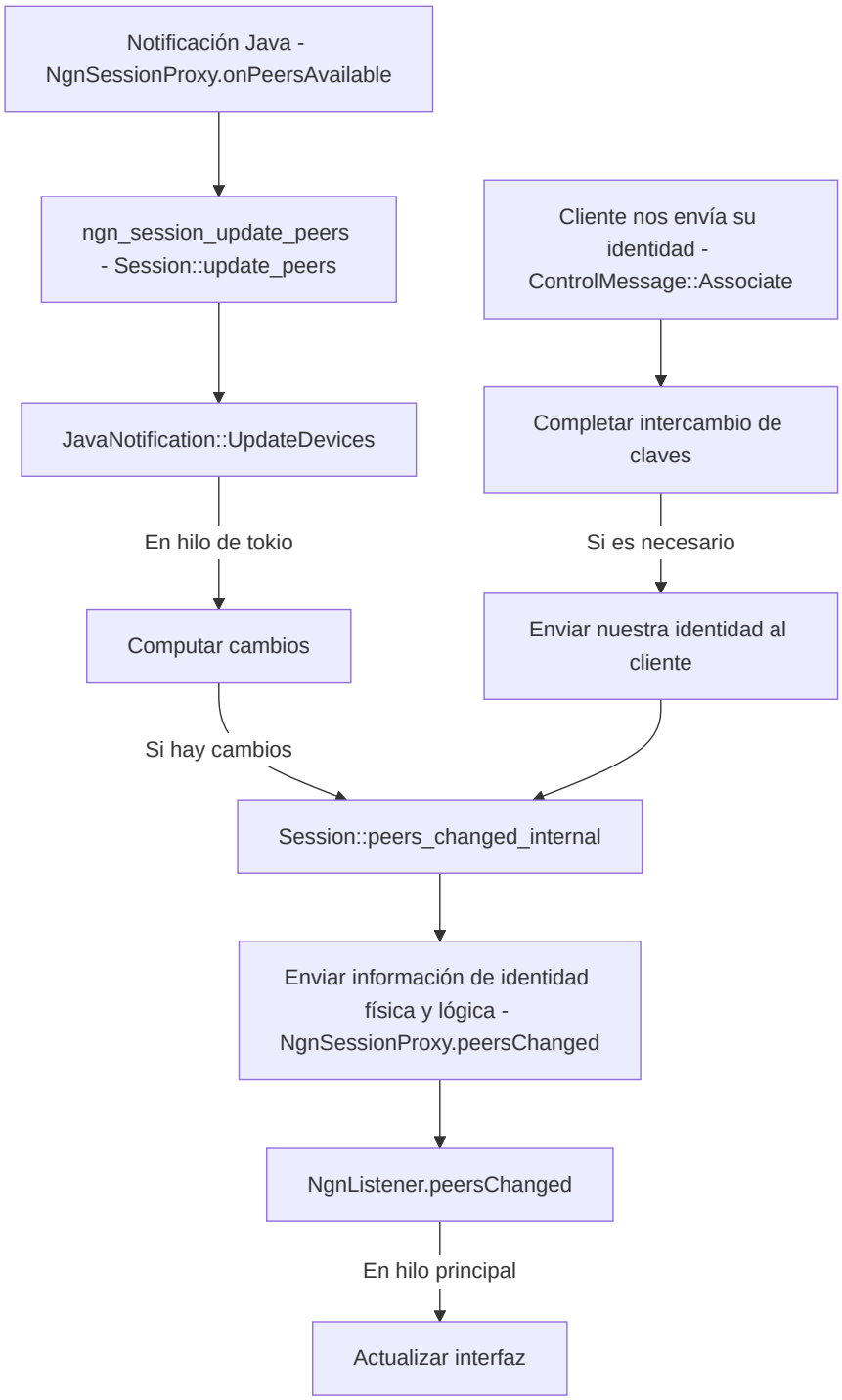

Figura 9.2: Ejemplo de flujo de control entre Java y Rust

9.8. Aplicación demostrativa

Se ha escrito un **juego multijugador por turnos** para demostrar la funcionalidad de la biblioteca, basado en el popular (si bien sencillo) juego 2048 [51].

9.8.1. Tecnología utilizada

El código de la aplicación es relativamente sencillo, y se encuentra en `examples/android`. Se ha utilizado Jetpack Compose como el toolkit para la interfaz (ver apartado 5.2.1 y apartado 8.1.4), y Kotlin como el lenguaje de programación (ver apartado 5.2.1).

La lógica del juego se encuentra implementada en `GameBoard.kt`, y es una conversión casi directa del código fuente original en Javascript[11], y la interfaz y la interacción con la biblioteca se encuentra en `MainActivity.kt`.

9.8.2. Pantalla inicial

La pantalla inicial sólo contiene una entrada de texto que permite seleccionar un nombre de usuario:

Una vez el usuario ha seleccionado el nombre de usuario y concede los permisos necesarios para el uso de WiFi Direct, se genera una identidad efímera con una clave creada aleatoriamente, y se comienza un escaneo de dispositivos adyacentes.

Una mejora obvia es persistir esta identidad en el almacenamiento del dispositivo (ver apartado 9.5). Esto no se ha hecho por falta de tiempo.

9.8.3. Lista de dispositivos disponibles

Una vez ha comenzado el escaneo, se muestra la lista de dispositivos disponibles (figura 9.4).

Al hacer click en alguno de ellos, se envía una solicitud de conexión, que el otro dispositivo necesita aprobar (figura 9.5).

La lista de dispositivos se actualiza con el nombre de usuario de los dispositivos conectados (figura 9.6).

Nótese que los detalles de esta pantalla están marcados por las limitaciones descritas en el apartado 7.2. En particular, idealmente se intentaría conectar directamente con los dispositivos cercanos, para así poder mostrar su identidad *lógica* (y no solo los datos del dispositivo), pero la limitación de Android de un grupo físico hace esto imposible.

[11] código fuente original en Javascript: `https://github.com/gabrielecirulli/2048`

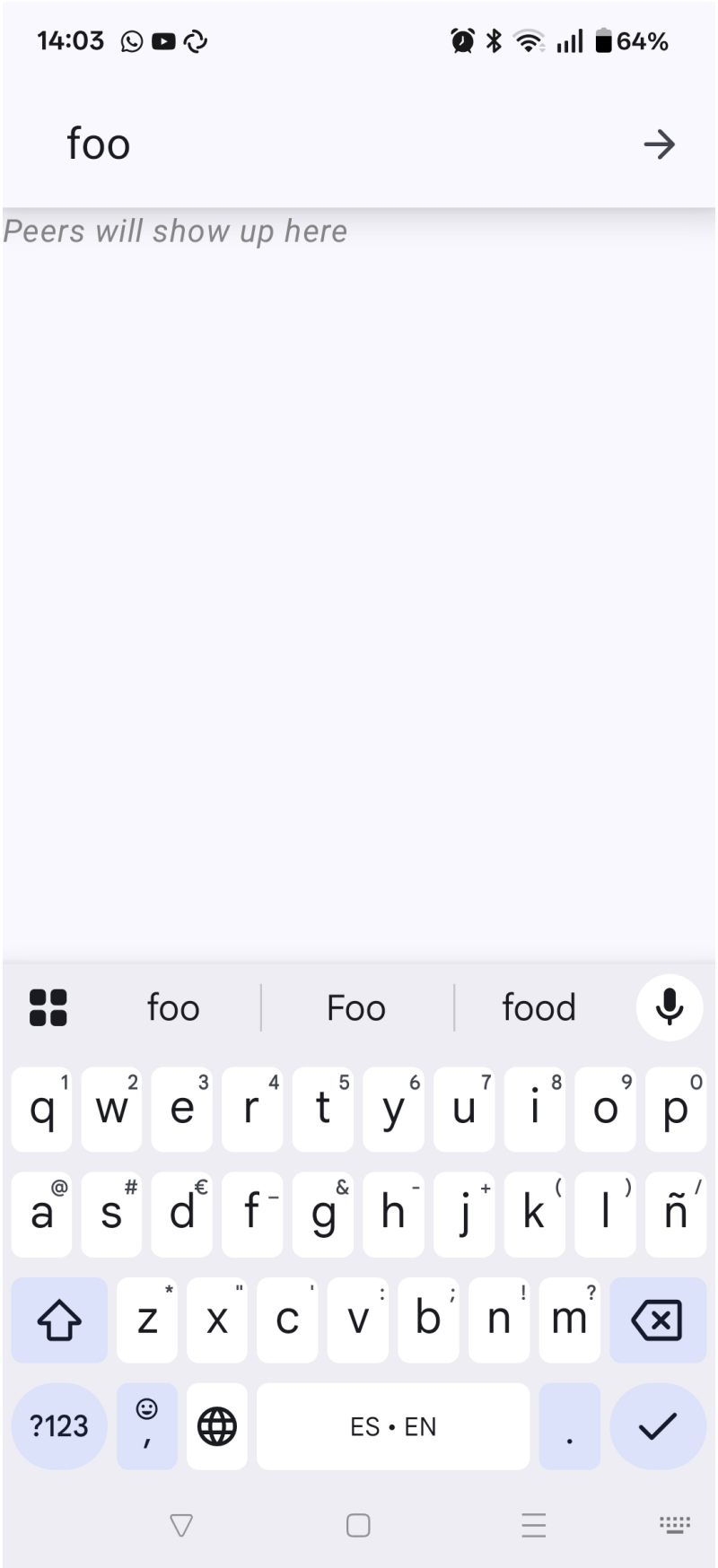

Figura 9.3: Captura de pantalla inicial de la aplicación

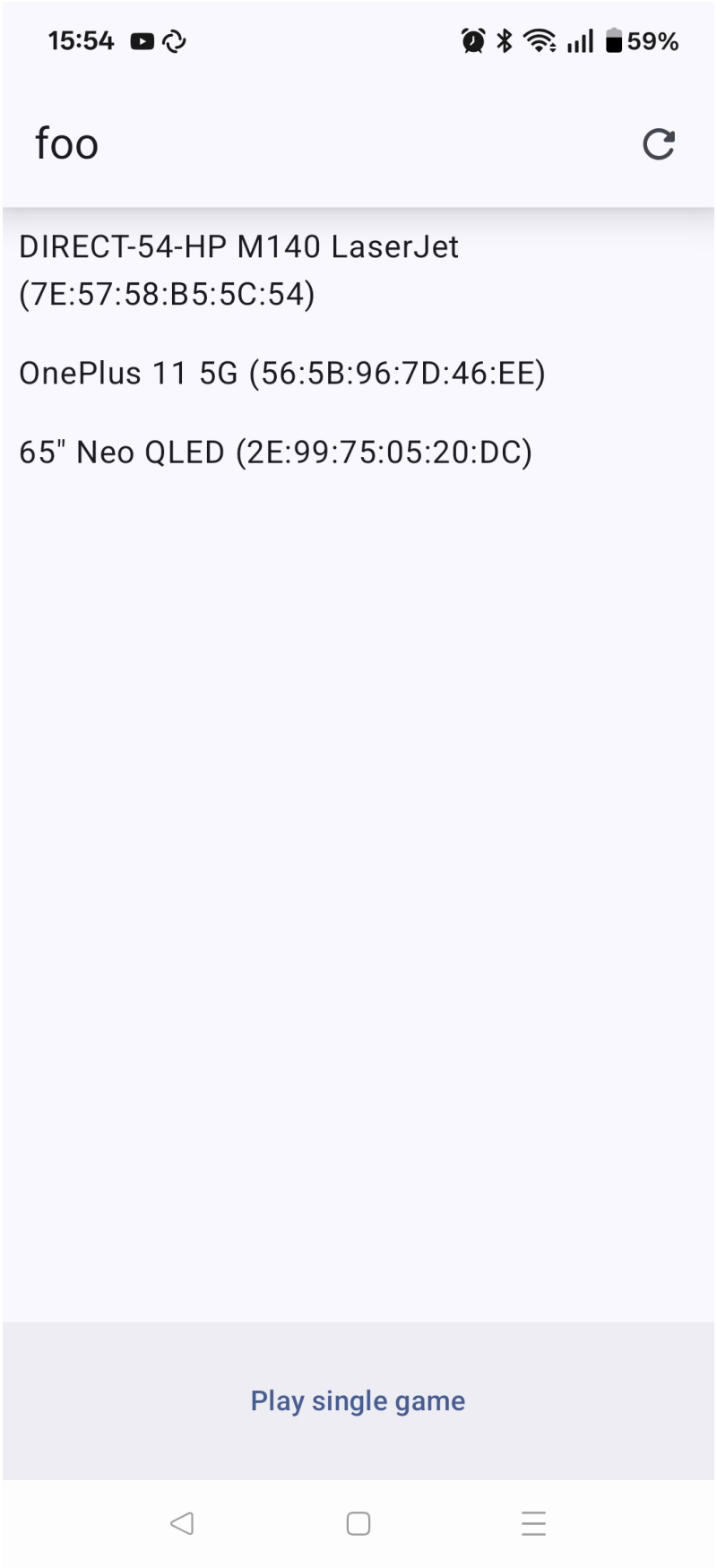

Figura 9.4: Lista de dispositivos disponibles

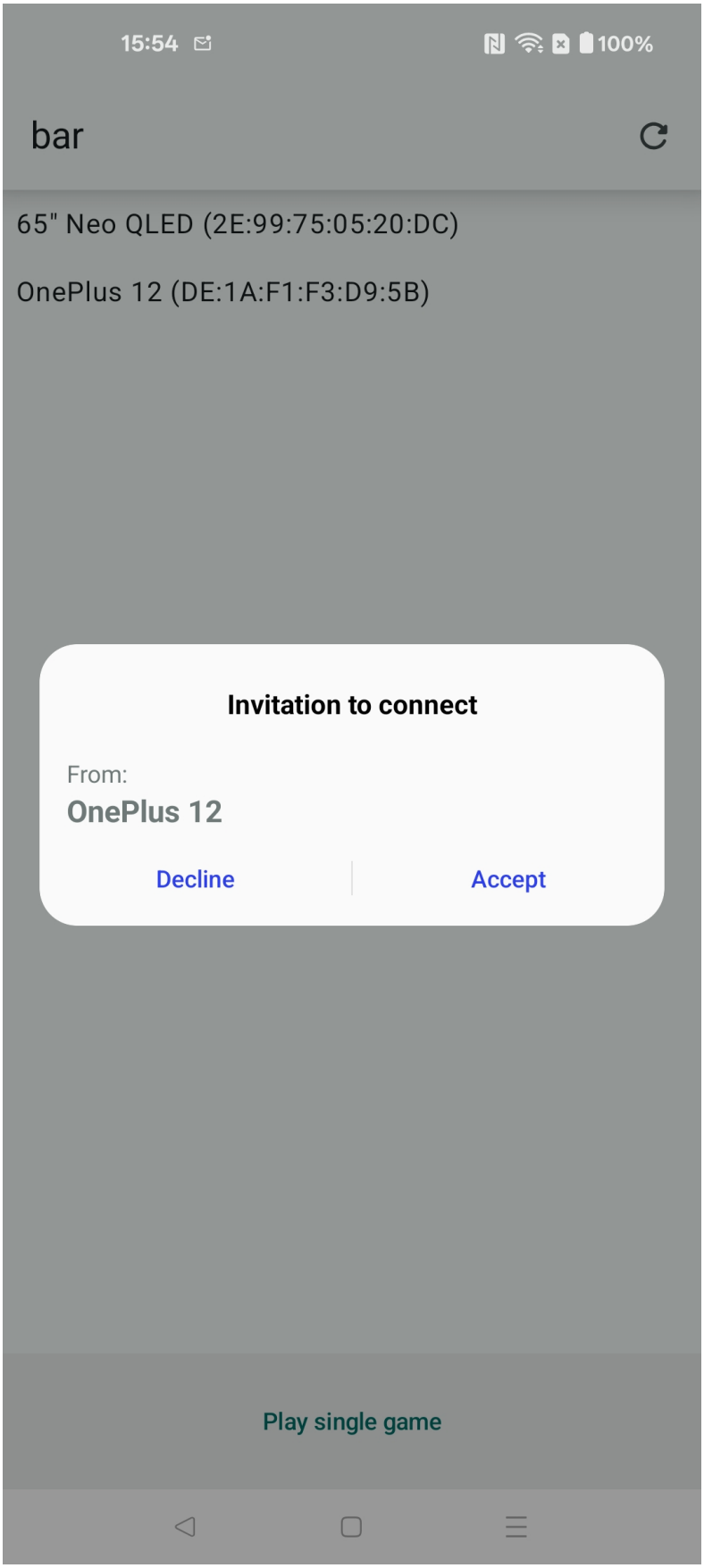

Figura 9.5: Solicitud de conexión

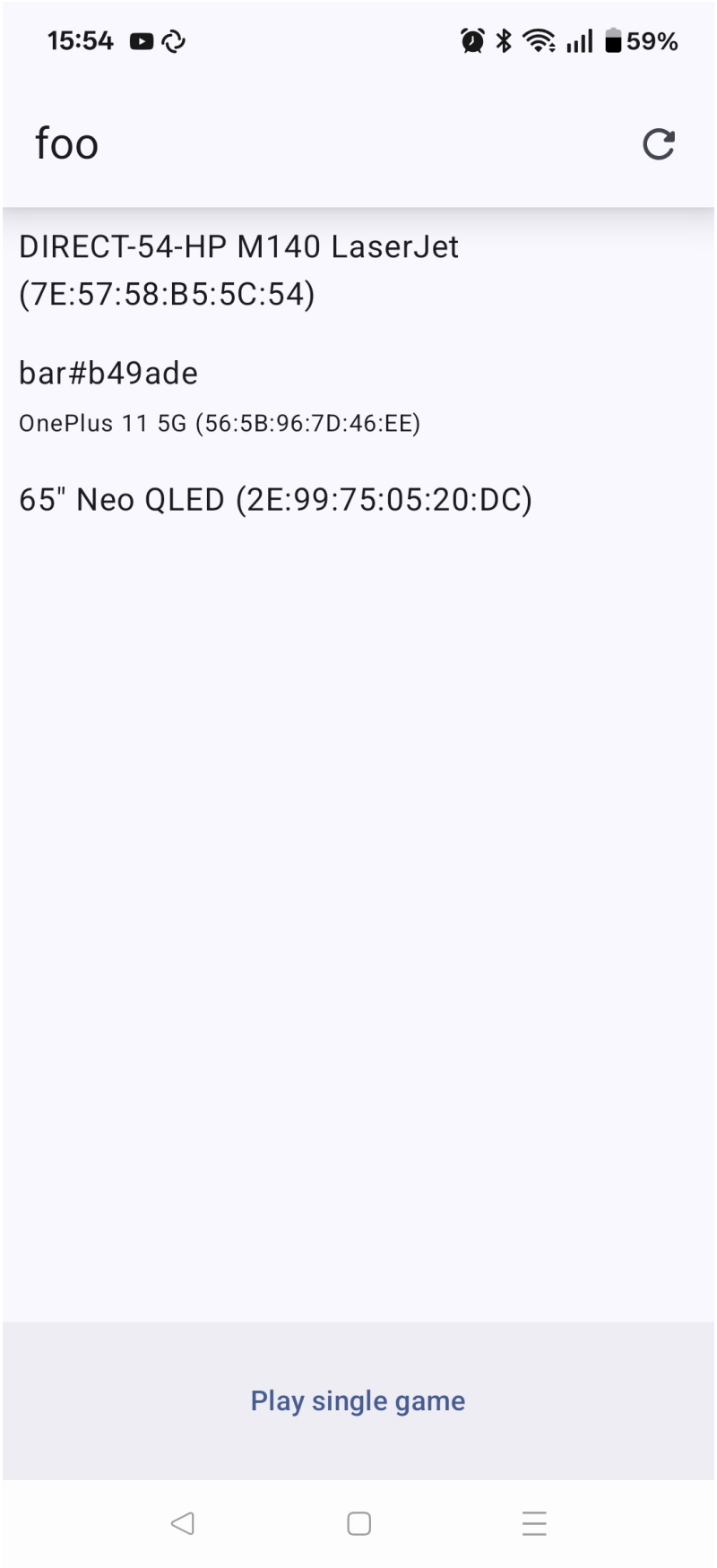

Figura 9.6: Lista de dispositivos con un dispositivo conectado

9.8.4. Pantalla de juego

El juego comienza al hacer click otra vez en la fila de conexión. La pantalla de juego muestra la puntuación total, y el turno actual (figura 9.7). El usuario sólo puede mover en su turno.

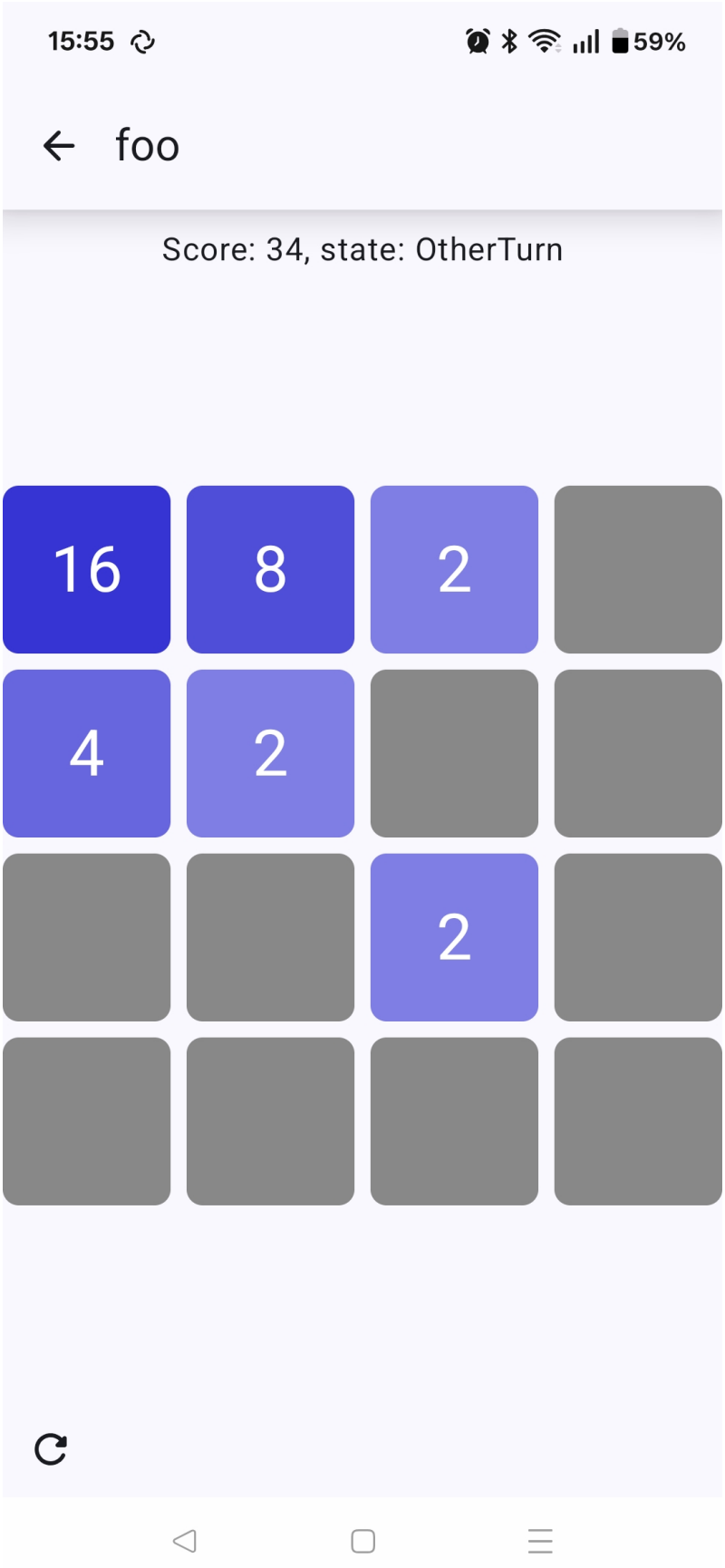

Figura 9.7: Pantalla de juego

Capítulo 10

Análisis de Riesgos

Este capítulo presenta el análisis de riesgos del proyecto. Se analizan los riesgos de aceptación, tecnológicos y de integración.

Se presenta a continuación una descripción detallada de los identificados en este análisis, incluyendo su probabilidad e impacto, así como las medidas correctivas o de mitigación propuestas.

R1	**Errores de sincronización**
Descripción	Es probable que en condiciones de red más inestables se descubran errores
Probabilidad	Media
Prioridad	Alta
Mitigaciones	Mejores tests automáticos. Para errores no reparables, mejor gestión de reconexiones

Cuadro 10.1: R1: Errores de sincronización

R2	**Falta de adopción**
Descripción	No se obtiene la adopción esperada de esta biblioteca.
Probabilidad	Alta
Prioridad	Alta
Mitigaciones	Implementación en otras plataformas y de otras capas de transporte.

Cuadro 10.2: R2: Falta de adopción

R3	Dificultad de mantenimiento
Descripción	Las abstracciones entre plataformas y capas de transporte pueden volverse difíciles de mantener con el tiempo.
Probabilidad	Alta
Prioridad	Media
Mitigaciones	Intentar mantener tanto código como sea posible en la capa común. Tests de integración en diferentes plataformas.

Cuadro 10.3: R3: Dificultad de mantenimiento

R4	Problemas de integración
Descripción	Las plataformas imponen restricciones varias, causando problemas severos de usabilidad.
Probabilidad	Alta
Prioridad	Media
Mitigaciones	En plataformas abiertas o semi-abiertas como Linux y Android respectivamente, trabajar con *upstream* para mejorar la situación en cuanto a permisos, y tal vez permitir múltiples grupos físicos, o descubrimiento pasivo.

Cuadro 10.4: R4: Problemas de integración

Capítulo 11

Organización y gestión del proyecto

11.1. Organización

La arquitectura del proyecto ha sido descrita con bastante detalle en el capítulo 9, por lo que se omitirá de esta sección.

11.1.1. Proceso de desarrollo

En este apartado se describe de manera resumida el proceso de desarrollo del proyecto, explicando las decisiones más importantes que se han tomado, y los problemas que se han encontrado durante el desarrollo.

Exploración inicial

Lo primero que se hizo fue realizar una exploración inicial[1] y algo superficial de las capacidades de comunicación P2P de Android. Esto solo buscaba descartar rápidamente si esa avenida era imposible.

Capa de transporte en Linux

Durante las primeras reuniones con los tutores, se debatió por donde afrontar la implementación. En contra de la recomendación de Guillermo, y con la intención de obtener algo funcional desde el principio, se decidió trabajar en la capa de transporte.

Fue bastante obvio durante el comienzo del desarrollo que trabajar directamente en la funcionalidad de Android iba a ser un dolor (ver apartado 12.2.1), por lo que primero se afrontó una implementación en Linux usando wpa_supplicant y D-Bus.

Se descubrió (lenta y dolorosamente) que el estado de estas tecnologías no era tan maduro como se esperaba inicialmente. La implementación de la capa de transporte

[1] exploración inicial: `https://github.com/emilio/android-wifip2p-test`

en Linux fue extremadamente lenta comparado con lo que hubiera sido de esperar, en parte por todas las complicaciones descritas en el apartado 12.2.1.

Cliente básico

Junto con la capa de transporte se creó un cliente básico en `examples/dbus` con el propósito de testearla, que eventualmente se convertiría en una pequeña interfaz de usuario usando GTK para testear casos más complejos con múltiples miembros.

Creación de abstracciones

A pesar de que el código se había escrito con la intención de ser reutilizable en otras plataformas, no se realizó el trabajo de factorizarlo y usar interfaces claras hasta que la capa de transporte funcionaba de manera robusta, ya que las restricciones la capa de transporte afectaban a la interfaz que se podía exponer.

Capa de transporte y demo en Android

Una vez se abstrajo la capa de transporte detrás de una interfaz reutilizable, se comenzó la implementación del soporte para Android, junto con la demo.

Ambas se desarrollaron en paralelo, y el desarrollo de la capa de transporte en sí fue bastante rápido (sólo necesitó adaptaciones menores para las restricciones de Android).

Implementación de cifrado punto a punto

En paralelo al desarrollo de la capa de transporte se desarrolló el sistema de identidad basada en clave pública y la firma de mensajes, pero aún quedaba afrontar la implementación del cifrado punto a punto.

Hubo complicaciones menores, ya que no había manera de conseguir el estado efímero del GO sin su identidad en Linux, pero se solucionaron y enviaron los cambios upstream.

11.1.2. Pruebas realizadas

La mayoría de pruebas realizadas han sido manuales (tristemente, ver apartado 12.2.1).

Se ha creado una pequeña interfaz en Linux para testear configuraciones de grupos arbitrarias (`test/setup.sh` admite configurar interfaces arbitrarias).

El código de Android ha sido testeado en los dispositivos disponibles (OnePlus 11/12 y Xiaomi Redmi). Por la naturaleza de la aplicación no puede ser testeada en un emulador.

11.2. Gestión del proyecto

En esta sección se realiza una breve descripción de la planificación y los recursos utilizados en el proyecto.

11.2.1. Recursos materiales y personales

Los recursos materiales utilizados en el proyecto cubren el software y el hardware necesarios para el desarrollo del proyecto.

El hardware utilizado para el desarrollo del proyecto han sido el ordenador de sobremesa del autor, que ha permitido ejecutar el entorno de desarrollo y las herramientas necesarias para el proyecto, y una variedad de teléfonos móviles usados para probar la demo en Android, incluyendo los OnePlus 11 y 12 del autor, y un Xiaomi de su pareja temporalmente.

Respecto al software, además de las herramientas de desarrollo (ver apartado 5.2), se ha utilizado Arch Linux[2] como sistema operativo de desarrollo.

Como editores de código se han utilizado Neovim[3] y Android Studio[4].

Los recursos personales utilizados, por su parte, han sido el autor del proyecto, Emilio Cobos Álvarez, y sus tutores, Guillermo González Talaván y Pedro Martín Vallejo Llamas.

Las responsabilidades del autor han abarcado el ciclo de vida completo del proyecto, siendo el encargado de la planificación, diseño, desarrollo, implementación, pruebas y documentación del mismo.

Los tutores del proyecto han sido los encargado de guiar al autor durante el proceso de desarrollo, ayudando a definir los objetivos del proyecto, y revisando el progreso del proyecto en las reuniones semanales y correos de seguimiento.

11.2.2. Planificación temporal

La planificación temporal del proyecto se resume en este apartado, pero se puede ver de manera más detallada en el *Anexo III. Estimación del tamaño y esfuerzo.*

La planificación del proyecto se ha realizado siguiendo la metodología ágil Scrum, de la cual se han descrito tanto su marco teórico como su implementación práctica en el proyecto en el apartado 5.1.1, y con mayor detalle en el anexo tres.

El progreso en la implementación ha sido bastante intermitente por las limitaciones temporales descritas en el apartado 5.1.1. Se puede ver una visualización de los commits de cada semana[5] a continuación:

[2]Arch Linux: `https://archlinux.org/`

[3]Neovim: `https://neovim.io/`

[4]Android Studio: `https://developer.android.com/studio`

[5]commits de cada semana: `https://github.com/emilio/ngn/graphs/contributors`

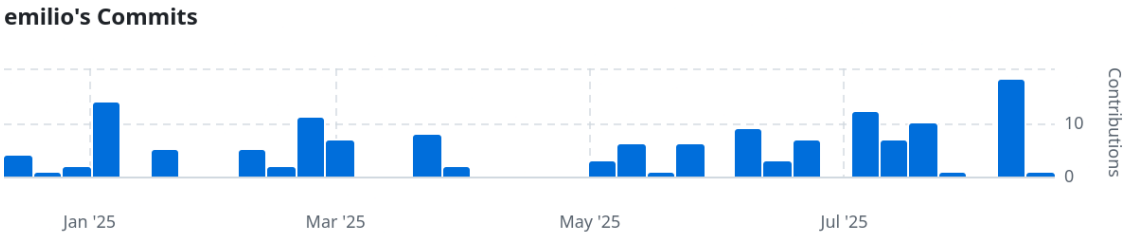

Figura 11.1: Número de commits por semana

Otro dato interesante (aunque también poco sorprendente), es el número de commits por día de la semana, donde se puede ver que ha sido mayormente un trabajo de fines de semana:

Día de la semana	Commits
Domingo	81
Sábado	28
Lunes	13
Miércoles	11
Jueves	7
Viernes	6
Martes	0

Cuadro 11.1: Número de commits por día de la semana

Capítulo 12

Conclusiones y trabajo futuro

12.1. Cumplimiento de los objetivos

12.1.1. Objetivos del sistema

Los objetivos del sistema planteados en el apartado 2.1 se han cumplido, con la excepción de la formación de múltiples grupos lógicos en el mismo grupo físico.

Este último es relativamente trivial de implementar tanto como parte de la biblioteca como por encima, y no se ha realizado por falta de tiempo, que se empleado tanto en implementar el objetivo opcional de abstraer varias plataformas y resolver otras complicaciones que se discuten a continuación.

Se ha cumplido adicionalmente el objetivo opcional de abstraer varias plataformas y sistemas operativos. Esto fue básicamente un imperativo, ya que desarrollar en Linux la mayoría del protocolo permitió ignorar muchas de las complicaciones de testear esta tecnología en Android (como tener acceso a múltiples dispositivos y depurar remotamente).

12.1.2. Requisitos no funcionales

Los requisitos no funcionales descritos en el apartado 6.2 son algo más complicado de evaluar objetivamente, pero considero que se han cumplido:

Portabilidad

La biblioteca funciona en Linux y Android, sistemas operativos con modelos de seguridad y limitaciones muy diferentes.

El código es mayoritariamente multi-plataforma, y el código que es específico a cada plataforma está confinado a `src/platform`.

La extensibilidad a otras plataformas depende de su capacidad de soportar la capa de transporte implementada, o de soportar capas de transporte alternativas. Parece que en este sentido el futuro puede ser más brillante que el presente, ya que

dada la discusión en el apartado 8.1.2 es posible que en un futuro no muy lejano todas las plataformas soporten el estándar WiFi Aware.

Extensibilidad

El código se ha diseñado para que la librería no exponga detalles inherentes a la capa de transporte, usando identificadores opacos (ver apartado 9.3.2). No se espera que extender la biblioteca para utilizar algo como Bluetooth como capa de transporte o incluso modos mixtos sea problemático.

Seguridad

Con la advertencia necesaria de que el autor no es un experto en seguridad y criptografía (y por lo tanto uso extenso de esta biblioteca necesitaría una auditoría externa), se han utilizado primitivas criptográficas sólidas, y una arquitectura que se corresponde a la que usan protocolos de paso de mensajes en producción.

Accesibilidad

La complejidad de este TFG reside en que funcione en dispositivos de consumo, y se ha conseguido sin lugar a dudas, dentro de las limitaciones que las diferentes plataformas imponen.

12.2. Problemas encontrados

Muchos de los problemas encontrados han sido ya discutidos en el apartado 7.2. Las limitaciones y diseño de las diferentes plataformas han influenciado el desarrollo de manera sustantiva, y consumido bastantes recursos de desarrollo.

Aparte de todos esos, hay otros problemas que son dignos de mención.

12.2.1. Dificultad de testeo

Especialmente en las fases iniciales de desarrollo, no estaba muy claro cuál iba a ser la mejor forma de probar el código. Tras un primer prototipo de aplicación P2P para Android, estaba claro que tener que testear en con dos dispositivos disponibles continuamente no iba a ser factible: Android Studio no funcionaba particularmente bien con múltiples dispositivos conectados a la vez, y el autor no tenía un segundo dispositivo disponible de manera continua.

Esto fue lo que hizo que Linux fuera la primera plataforma que la librería soportó (el autor disponía de manera continua dos ordenadores con tarjeta de red inalámbrica).

Aún así, testear con dos ordenadores en paralelo no era particularmente ideal, ya que es necesario tener el entorno de desarrollo en ambos, y asegurarse de que el proyecto está en el mismo estado, lo cual es propenso a errores.

El autor pensó (inocentemente) que mientras el ordenador tuviera múltiples tarjetas de red sería posible controlarlas independientemente, y por lo tanto adquirió una tarjeta de red extra para su escritorio. Esto resulta no ser posible tal y como se detalla en una discusión[1] en la lista de hostap.

Junto con la interacción entre `wpa_supplicant` y `NetworkManager` descrita en el apartado 7.2.8, hizo un quebradero de cabeza conseguir un flujo de desarrollo que no requiriera múltiples dispositivos. Hubo que indagar en cómo `hostap` testea su propio código, y sólo así se llegó a la solución actual (en `test/setup.sh`) de correr múltiples instancias de `wpa_supplicant`, cada una controlando una interfaz creada por el driver `mac80211_hwsim`, y un bus independiente.

Esta solución sigue sin ser ideal, porque tener que desconectar `NetworkManager` y `wpa_supplicant` del sistema implica que si no tienes Ethernet estás (temporalmente) offline. Una alternativa o mejora potencial sería usar virtualización más agresivamente (via kvm o similar), pero eso vuelve a traer algunos de los problemas de asegurarse de que el código que corre en la máquina virtual sea el que estás editando.

12.2.2. Depuración complicada

Depurar sistemas que son dependientes de factores externos y poco deterministas es notablemente difícil. Descubrir muchas de las interacciones problemáticas entre componentes requirió sesiones complejas de depuración que, siendo realistas, sólo fueron posibles gracias a rr (ver apartado 5.2.2).

Los scripts de testeo incluyen opciones para depurar tanto `wpa_supplicant` como la aplicación. También compilan la aplicación con ASan, que es útil para cazar errores de memoria o sincronización (Rust los previene por defecto, pero aún así el autor es estúpido[2]).

Similarmente, fue difícil encontrar el por qué el provisionamiento de direcciones usando direcciones IPv6 de link-local funcionaba en un ordenador y no otro (siendo por supuesto DHCP la causa, ver apartado 7.2.5).

12.3. Aprendizaje personal

Este proyecto ha sido guiado mayormente por la curiosidad personal del autor, más que por la necesidad del reconocimiento académico. En ese sentido el proyecto ha sido muy exitoso, ya que el autor ha aprendido mucho más de lo que esperaba en un alto espectro de áreas: arquitectura y estándares de redes, criptografía, desarrollo en Android, comunicación entre procesos via DBus.

[1] discusión: `https://lists.infradead.org/pipermail/hostap/2015-September/033754.html`

[2] el autor es estúpido: `https://github.com/emilio/ngn/commit/e24706ad567969c2a47663ab8de2e0808086a7db`

12.3.1. Contribuciones a proyectos externos

Una de las partes más gratificantes de este proyecto ha sido salir del ecosistema donde el autor suele manejarse en su día a día (navegadores y estándares web, Linux, GTK), y ver cómo la experiencia adquirida es útil y valiosa en otros campos.

El autor ha sido capaz de hacer contribuciones técnicas a `wpa_supplicant` (descritas en el apartado 5.2.1), dchpcd (descrita en el apartado 7.2.5), y zbus[3], las cuales incluyen una mejora de rendimiento[4], una mejora de rendimiento acepada pero pendiente[5] y una discusión sobre el generador de código[6].

12.4. Línea de tiempo alternativa

Hay una cosa en particular que, con el beneficio de mirar hacia atrás, tal vez hubiera hecho de manera diferente (o al menos me gustaría tener el espejo de Galadriel[7] para ver la versión alternativa).

La decisión que más ha marcado tanto el tiempo como la dirección técnica del proyecto ha sido sin lugar a dudas la de comenzar por una abstracción de la capa de transporte en Linux (ver apartado 11.1.1). Esta fue una decisión unilateral en contra de la sugerencia inicial de Guillermo, razonada con que sin una capa de transporte funcional, todo el proyecto sería *vaporware*[8].

Por una parte, ignorarla en su momento hubiera ahorrado una gran cantidad de tiempo y esfuerzo que se hizo por adelantado.

Por otra parte, muchas de las complicaciones que se encontraron y de la investigación necesaria para solucionarlas, fueron las que hicieron posible que la implementación en Android fuera relativamente rápida, y muchas de las decisiones que se hicieron para acomodar a las APIs de `wpa_supplicant` fueron útiles en Android.

Mi hipótesis de cómo sería esta línea de tiempo alternativa sería que la demo de Android sería mucho más completa (tal vez persistencia, identidades conocidas, etc), y la biblioteca expondría mucha más funcionalidad, pero o bien no tendría una capa de transporte robusta (tal vez abusando que Android nos proporciona la IP del GO directamente), o no sería multi-plataforma (estoy seguro de que no me hubiera dado tiempo a solventar todos los inconvenientes que encontré en Linux).

En cualquier caso, estoy seguro de que hubiera acabado con conocimiento mucho más superficial de los protocolos utilizados e involucrados en mover unos y ceros por el aire.

[3] zbus: `https://github.com/dbus2/zbus`

[4] una mejora de rendimiento: `https://github.com/dbus2/zbus/pull/1188`

[5] una mejora de rendimiento acepada pero pendiente: `https://github.com/dbus2/zbus/pull/1189`

[6] una discusión sobre el generador de código: `https://github.com/dbus2/zbus/issues/1180`

[7] espejo de Galadriel: `https://es.wikipedia.org/wiki/Espejo_de_Galadriel`

[8] *vaporware*: `https://en.wikipedia.org/wiki/Vaporware`

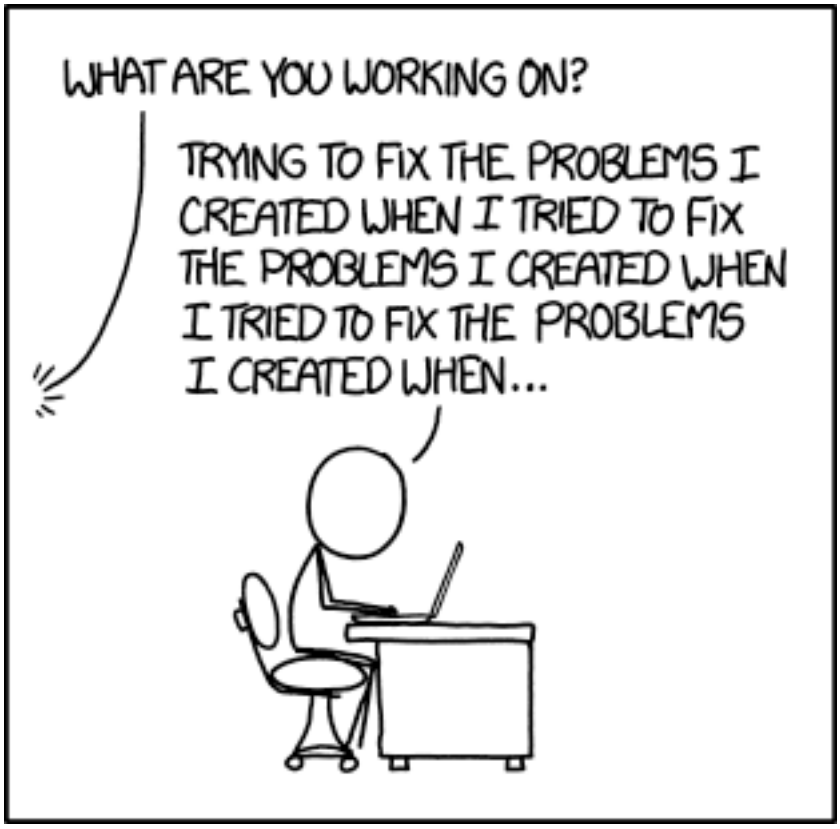

Figura 12.1: Problemas autoinfligidos (Fixing Problems — XKCD 1739)

12.5. Conclusión final y trabajo futuro

Hay muchas cosas que se han quedado en el tintero. Muchas son simplemente trabajo de implementación por hacer, como implementar más capas de transporte (Bluetooth, WiFi Aware) o plataformas (Windows siendo el candidato obvio, pero macOS e iOS también es posible especialmente si implementan WiFi Aware, ver el apartado 8.1.2), o implementar la funcionalidad de grupos lógicos para simplificar el direccionamiento de mensajes.

Otras son más complicadas. Este tipo de tecnologías tienen un potencial inmenso, pero desafortunadamente las plataformas móviles actualmente no permiten los casos de uso más interesantes que permiten crear redes más complejas. Se deja como ejercicio al lector hipotetizar si esto es por falta de recursos o casos de uso, o intereses corporativos.

Una serie de mejoras concretas que al autor le gustaría ver en Android serían:

- Soporte para múltiples grupos de WiFi Direct en Android: WiFi Aware permite el equivalente, que podría ser una alternativa.
- Parece que las asociaciones de WiFi Direct se almacenan persistentemente a nivel del sistema, e indefinidamente, lo cual puede ser no ideal para casos de uso ad-hoc.
- Hacer el servicio de localización no requerido para usar WiFi Direct / WiFi

Aware.
- Permitir conexión entre dispositivos no conocidos sin interacción de usuario. El autor es consciente de los problemas de seguridad y privacidad que esto podría conllevar, dicho eso...

Soporte para WiFi Aware en otras plataformas que no sean Android sería genial, ya que WiFi Aware tiene mejoras significativas sobre WiFi Direct. Por ejemplo, permite enviar algunos datos entre nodos antes del emparejamiento (podría usarse para la identidad lógica por ejemplo), y no requiere un nodo coordinador como el *Group Owner* de WiFi direct.

Parece que esto viene en camino en plataformas de Apple gracias a la DMA (ver apartado 8.1.2). En Linux existe una implementación llamada OpenNAN[9], pero no parece activa o mantenida. Implementar WiFi Aware en `wpa_supplicant` sería un proyecto futuro particularmente interesante.

[9]OpenNAN: `https://github.com/seemoo-lab/opennan`

Bibliografía

[1] Bitchat. https://bitchat.free/, 2025. Accessed: 2025-08-09.

[2] Bitchat protocol whitepaper. https://github.com/permissionlesstech/bitchat/blob/d4262fab6ce3cd775266dbf91883dd7219b137ce/WHITEPAPER.md, 2025. Accessed: 2025-08-09.

[3] Ian F. Akyildiz and Xudong Wang. A survey on wireless mesh networks. *IEEE Communications Magazine*, 43(9):S23–S30, 2005.

[4] Martin R. Albrecht, Jorge Blasco, Rikke Bjerg Jensen, and Lenka Mareková. Mesh messaging in large-scale protests: Breaking bridgefy. Cryptology ePrint Archive, Paper 2021/214, 2021.

[5] Martin R. Albrecht, Raphael Eikenberg, and Kenneth G. Paterson. Breaking bridgefy, again: Adopting libsignal is not enough. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 269–286, Boston, MA, August 2022. USENIX Association.

[6] WiFi Alliance. Wi-fi direct specification v1.9. 2021. Accessed: 2025-08-13.

[7] WiFi Alliance. Wifi aware specification. https://www.wi-fi.org/file/wi-fi-aware-specification, 2022. Accessed: 2025-08-14.

[8] WiFi Alliance. How far does a wi-fi direct connection travel? https://www.wi-fi.org/faq/how-far-does-wi-fi-direct-connection-travel, 2025. Accessed: 2025-08-13.

[9] Beatriz Bernárdez Jiménez Amador Durán Toro. *Metodología para el análisis de Requisitos de sistemas software*. Universidad de Sevilla, 2000.

[10] Apple. Multipeer connectivity framework — apple developer documentation. https://developer.apple.com/documentation/multipeerconnectivity. Accessed: 2025-08-10.

[11] Atlassian. Kanban, a brief introduction. https://www.atlassian.com/agile/kanban. Accessed: 2025-08-28.

[12] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. RFC 9420, July 2023.

[13] Archie Bland. Firechat — the messaging app that's powering the hong kong protests. *The Guardian*, 2014.

[14] Briar Project. Briar project. https://briarproject.org, 2025. Accessed: 2025-08-09.

[15] Bridgefy Project. Bridgefy mesh network basics. https://bridgefy.notion.site/Bridgefy-Mesh-Network-Basics-002ecd4c7bb64986bf00beb6d43d3036, 2025. Accessed: 2025-08-09.

[16] European Commission. Apple – operating systems – ios – article 6(7) – sp – features for connected physical devices. https://digital-markets-act-cases.ec.europa.eu/cases/DMA.100203, 2025. Accessed: 2025-08-14.

[17] European Commission. Overview of proposed measures - apple - operating systems - ios - article 6(7) - sp - features for connected physical devices. https://digital-markets-act.ec.europa.eu/document/download/8f28e456-5bd4-4b33-af95-b9f52aeb8a03_en, 2025. Accessed: 2025-08-14.

[18] The Rust Project Contributors. Platform support - the rustc book. https://doc.rust-lang.org/nightly/rustc/platform-support.html, 2025.

[19] David Anderson, Tailscale. How nat traversal works. https://tailscale.com/blog/how-nat-traversal-works, 2020. Accessed: 2025-08-09.

[20] Derek Davidson. Why do we use story points for estimating? https://www.scrum.org/resources/blog/why-do-we-use-story-points-estimating. Accessed: 2025-02-24.

[21] Kotlin Foundation. Comparison to java — kotlin documentation. https://kotlinlang.org/docs/comparison-to-java.html, 2025. Accessed: 2025-08-10.

[22] Francisco J. García-Peñalvo, Sergio Bravo Martín, and Miguel Ángel Conde González. https://hdl.handle.net/10366/56058, 2000.

[23] Google. Nearby connections overview. https://developers.google.com/nearby/connections/overview. Accessed: 2025-08-09.

[24] Google. Develop ui for android — jetpack compose — android developers. https://developer.android.com/develop/ui, 2025. Accessed: 2025-08-24.

[25] Google. Kotlin and android — android developers. https://developer.android.com/kotlin, 2025. Accessed: 2025-08-10.

[26] Google. Wi-fi direct overview — android developers. https://developer.android.com/develop/connectivity/wifi/wifip2p, 2025. Accessed: 2025-08-13.

[27] Apple Inc. Security of runtime process in ios, ipados, and visionos. https://support.apple.com/guide/security/security-of-runtime-process-sec15bfe098e/web, 2024. Accessed: 2025-08-14.

[28] International Telecommunication Union. Measuring digital development: Facts and figures 2022. https://www.itu.int/hub/publication/D-IND-ICT_MDD-2024-4/, 2024. Accessed: 2025-08-09.

[29] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.

[30] Protocol Labs. libp2p overview. https://docs.libp2p.io/concepts/introduction/overview/. Accessed: 2025-08-10.

[31] Adam Langley, Mike Hamburg, and Sean Turner. Elliptic Curves for Security. RFC 7748, January 2016.

[32] Jouni Malinen. wpa_supplicant / hostapd: wpa_supplicant d-bus api. https://w1.fi/wpa_supplicant/devel/dbus.html, 2025. Accessed: 2025-08-13.

[33] Maral Kaplan, Anna Corrente, Eugenio Montalti. An overview of manet technologies: Advantages and disadvantages in the military. https://finabel.org/an-overview-of-manet-technologies-advantages-and-disadvantages-in-the-military/, 2022. Accessed: 2025-08-09.

[34] Meshtastic Project. Meshtastic open source lora mesh communication. https://meshtastic.org, 2025. Accessed: 2025-08-09.

[35] n0 computer. Iroh overview. https://www.iroh.computer/docs/overview. Accessed: 2025-08-10.

[36] Dr. Thomas Narten, Tatsuya Jinmei, and Dr. Susan Thomson. IPv6 Stateless Address Autoconfiguration. RFC 4862, September 2007.

[37] Mark Nottingham. Centralization, Decentralization, and Internet Standards. RFC 9518, December 2023.

[38] Robert O'Callahan, Chris Jones, Nathan Froyd, Kyle Huey, Albert Noll, and Nimrod Partush. Engineering record and replay for deployability: Extended technical report. *CoRR*, abs/1705.05937, 2017.

[39] Stack Overflow. Stack overflow developer survey 2022. https://survey.stackoverflow.co/2022/#section-version-control-version-control-systems, 2022. Accessed: 2025-08-14.

[40] Trevor Perrin. The noise protocol framework. https://noiseprotocol.org/noise.html, 2018. Accessed: 2025-08-09.

[41] LaTeX Project. Latex – a document preparation system. https://www.latex-project.org, 2025. Accessed: 2025-08-10.

[42] Alex Rebert and Christoph Kern. Secure by design: Google's perspective on memory safety. Technical report, Google Security Engineering, 2024.

[43] NSA Media Relations. Software memory safety. Technical report, 2022. Accessed: 2025-08-10.

[44] Ken Schwaber and Jeff Sutherland. La guía scrum. 2020.

[45] Ken Schwaber and Jeff Sutherland. *La Guía Scrum*. 2020.

[46] the Home of Scrum! Scrum.org. What is scrum? https://www.scrum.org/resources/what-scrum-module. Accessed: 2025-02-24.

[47] Marten Seemann and Christian Huitema. QUIC Address Discovery. Internet-Draft draft-ietf-quic-address-discovery-00, Internet Engineering Task Force, March 2025. Work in Progress.

[48] William A. Simpson, Dr. Thomas Narten, Erik Nordmark, and Hesham Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861, September 2007.

[49] Ryah Nughaimesh Sultan, Wameedh Flayyih, and Hamid Ali. Extending wi-fi direct single-group network to multi-group network based on android smartphone. *Iraqi Journal of Science*, pages 419–438, 01 2023.

[50] Jeff Sutherland. Story points: Why are they better than hours? - scrum inc. https://www.scruminc.com/story-points-why-are-they-better-than, 2013. Accessed: 2025-02-24.

[51] Wikipedia. 2048 (videojuego) — Wikipedia, the free encyclopedia. http://es.wikipedia.org/w/index.php?title=2048%20(videojuego)&oldid=162078427, 2025. [Online; accessed 24-August-2025].

[52] Wikipedia. Wireless ad hoc network, Applications — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Wireless%20ad%20hoc%20network&oldid=1277152102#Applications, 2025. [Online; accessed 02-March-2025].

Anexos

Anexo 1

Especificaciones del sistema

1.1. Introducción

En este anexo se recoge la especificación de requisitos del Trabajo de Fin de Grado.

Para ello, el documento sigue, de manera aproximada, la estructura de documento de requisitos de sistema propuesta por Durán Toro y Bernárdez Jiménez en su *Metodología para la Elicitación de Requisitos de Sistemas Software* [9].

También se utilizan las plantillas para especificación de requisitos presentados por los mismos.

Los requisitos que se recogen en el presente anexo han sido obtenidos a partir de entrevistas con los tutores del proyecto, Guillermo González Talaván y Pedro Martín Vallejo Llamas. A partir de estas, se han elaborado tanto la propuesta, que incluye los objetivos funcionales del sistema, como el resto de requisitos, funcionales y no funcionales, recogidos en este anexo.

1.1.1. Propuesta de Trabajo de Fin de Grado

La propuesta se coordinó con los tutores via gestión directa. Se incluyen a continuación las partes relevantes de la propuesta, que se coordinó con los tutores por gesta:

Descripción

El acceso a internet no es tan universal como suele parecer. Sin embargo, dispositivos convencionales al alcance de la mayoría de la población soportan comunicarse entre ellos de manera directa, via tecnologías estándar como Bluetooth, WiFi-Direct, u otras.

Estas tecnologías tienen casos de usos muy variados, como comunicación en situaciones de emergencia o lugares remotos, intercambio de datos de manera más privada que una conexión a internet convencional...

A pesar de ello, su grado de adopción no es particularmente grande, en parte por la dificultad de uso de estas tecnologías en comparación con internet. Se desarrollará una biblioteca que abstraiga sobre diferentes tecnologías de comunicación directa, y además proporcione capacidades de agrupación, identificación, y opcionalmente enrutamiento, de más alto nivel.

Objetivos funcionales

- La biblioteca permitirá a varios dispositivos enviar mensajes entre ellos sin necesidad de conexión a internet.
- La biblioteca proveerá una abstracción de bajo nivel sobre la tecnología física de comunicación.
- Tendrá al menos una implementación como prueba de concepto.
- Opcionalmente, la biblioteca también abstraerá diferencias entre plataformas / sistemas operativos.
- La biblioteca proveerá una abstracción de más alto nivel que permitirá:
  - Formación de grupos lógicos dentro de un grupo físico. Opcionalmente, se investigará la posibilidad de que un grupo lógico abarque más de un grupo físico.
  - Identificación (via sistema de clave pública / privada o similar), independiente de la capa física.
  - Opcionalmente, enrutado de mensajes via: Broadcast / Broadcast a un grupo lógico / Mensaje directo entre dos nodos lógicos (identidades).
- Se desarrollará una aplicación sencilla que demuestre las capacidades de la biblioteca.

Entornos de desarrollo y explotación

Vim, Android, Android Studio, rr, Linux, Kotlin, Java, C, C++, Rust, Python.

1.2. Participantes en el proyecto

En este proyecto hay tres participantes: El alumno y los dos tutores, pertenecientes ambos a la misma organización, la Universidad de Salamanca.

Participante	Emilio Cobos Álvarez
Rol	Desarrollador
Es desarrollador	Sí
Es cliente	No
Es usuario	Sí
Comentarios	Ninguno

Cuadro 1.1: Participante: Emilio Cobos Álvarez

Participante	Guillermo González Talaván
Rol	Tutor
Es desarrollador	No
Es cliente	Sí
Es usuario	Sí
Comentarios	Ninguno

Cuadro 1.2: Participante: Guillermo González Talaván

Participante	Pedro Martín Vallejo Llamas
Rol	Tutor
Es desarrollador	No
Es cliente	No
Es usuario	Sí
Comentarios	Ninguno

Cuadro 1.3: Participante: Pedro Martín Vallejo Llamas

Organización	Universidad de Salamanca
Dirección	Patio de Escuelas Mayores, 1, 37008, Salamanca, España
Teléfono	+34 923 29 44 00
Fax	—
Comentarios	Ninguno

Cuadro 1.4: Organización: Universidad de Salamanca

1.3. Objetivos del sistema

Se detallan a continuación los objetivos que se pretenden alcanzar con el desarrollo del sistema.

OBJ-1	Abstracción de la capa física
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	Diseñar e implementar un módulo que oculte las complejidades de las distintas tecnologías de comunicación directa (Bluetooth, WiFi-Direct, etc.), permitiendo enviar y recibir mensajes sin

OBJ-1	Abstracción de la capa física
	necesidad de conocer las APIs específicas.
Subobjetivos	—
Importancia	vital
Urgencia	inmediatamente
Estado	validado
Estabilidad	alta
Comentarios	ninguno

Cuadro 1.5: OBJ-1: Abstracción de la capa física

OBJ-2	Interoperabilidad multiplataforma
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	Diseñar abstracciones que permitan que la biblioteca funcione en múltiples sistemas operativos (Android, iOS, Linux, Windows), garantizando compatibilidad con la abstracción de capa física.
Subobjetivos	—
Importancia	vital
Urgencia	inmediatamente
Estado	validado
Estabilidad	alta
Comentarios	ninguno

Cuadro 1.6: OBJ-2: Interoperabilidad multiplataforma

OBJ-3	Gestión de grupos lógicos
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván
Descripción	Implementar la capacidad de formar grupos lógicos dentro de una red física, explorar la posibilidad de interconectar varios grupos físicos.
Subobjetivos	—
Importancia	quedaría bien
Urgencia	puede esperar
Estado	validado
Estabilidad	alta
Comentarios	ninguno

OBJ-3	Gestión de grupos lógicos

Cuadro 1.7: OBJ-3: Gestión de grupos lógicos

OBJ-4	Identificación y seguridad
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	Incorporar un sistema de identificación basado en criptografía de clave pública/privada, garantizando autenticidad y privacidad de mensajes, independiente de la tecnología física de comunicación.
Subobjetivos	—
Importancia	vital
Urgencia	hay presión
Estado	validado
Estabilidad	alta
Comentarios	ninguno

Cuadro 1.8: OBJ-4: Identificación y seguridad

OBJ-5	Prueba de concepto
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	Implementar una aplicación sencilla como validación de la biblioteca y medio para obtener retroalimentación de facilidad de uso y estabilidad.
Subobjetivos	—
Importancia	vital
Urgencia	hay presión
Estado	validado
Estabilidad	alta
Comentarios	ninguno

Cuadro 1.9: OBJ-5: Prueba de concepto

1.4. Catálogo de requisitos del sistema

1.4.1. Requisitos funcionales

Los requisitos funcionales definen «qué debe hacer el sistema con la información almacenada para alcanzar los objetivos de su negocio».

La elicitación de requisitos basada en casos de usos no es particularmente aplicable a una biblioteca de software, por lo que se ha simplificado sustancialmente las plantillas de Durán y Bernárdez.

RF-1	Comunicación directa entre dispositivos
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	El sistema debe permitir a los dispositivos intercambiar mensajes sin requerir conexión a internet.
Relaciones	OBJ-1, OBJ-5

Cuadro 1.10: RF-1: Comunicación directa entre dispositivos

RF-2	Abstracción de la capa física
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	El sistema debe ofrecer una API unificada para enviar y recibir mensajes, independientemente de la tecnología utilizada (Bluetooth, WiFi-Direct, etc.).
Relaciones	OBJ-1

Cuadro 1.11: RF-2: Abstracción de la capa física

RF-3	Interoperabilidad multiplataforma
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	El sistema debe opcionalmente proveer implementaciones para distintas plataformas.
Relaciones	OBJ-2

Cuadro 1.12: RF-3: Interoperabilidad multiplataforma

RF-4	Gestión de grupos lógicos
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	El sistema debe permitir la creación, unión y salida de grupos lógicos dentro de un grupo físico.
Relaciones	OBJ-3

Cuadro 1.13: RF-4: Gestión de grupos lógicos

RF-5	Interconexión de grupos
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	Opcionalmente, la biblioteca debe permitir la interconexión de múltiples grupos físicos en un solo grupo lógico.
Relaciones	OBJ-3

Cuadro 1.14: RF-5: Interconexión de grupos

RF-6	Identificación segura de nodos
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	El sistema debe implementar un sistema de identificación único para cada dispositivo, basado en criptografía de clave pública/privada.
Relaciones	OBJ-4

Cuadro 1.15: RF-6: Identificación segura de nodos

RF-7	Privacidad y autenticación de mensajes
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	El sistema debe garantizar que los mensajes puedan ser autenticados y, opcionalmente, cifrados de extremo a extremo.
Relaciones	OBJ-4

RF-7	Privacidad y autenticación de mensajes

Cuadro 1.16: RF-7: Privacidad y autenticación de mensajes

RF-8	Aplicación de prueba de concepto
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	Debe desarrollarse una aplicación que use la biblioteca y valide sus capacidades.
Relaciones	OBJ-5

Cuadro 1.17: RF-8: Aplicación de prueba de concepto.

1.4.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que definen cualidades sobre el sistema que no están directamente relacionadas con la funcionalidad del mismo, sino con aspectos como el rendimiento, la usabilidad, la seguridad, etc [22].

RNF-1	Portabilidad
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	Accesibilidad desde diferentes plataformas y sistemas operativos.
Relaciones	OBJ-2

Cuadro 1.18: RNF-1: Portabilidad

RNF-2	Extensibilidad
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	Facilidad para añadir nuevas plataformas y transportes físicos.
Relaciones	OBJ-2

Cuadro 1.19: RNF-2: Extensibilidad

RNF-3	Seguridad
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	Identificación y cifrado de mensajes independiente de la capa física.
Relaciones	OBJ-4

Cuadro 1.20: RNF-3: Seguridad

RNF-4	Accesibilidad
Versión	1.0 (20/10/2024)
Autores	Emilio Cobos Álvarez
Fuentes	Guillermo González Talaván, Pedro Martín Vallejo Llamas
Descripción	La biblioteca deberá funcionar con dispositivos accesibles / no requerir hardware especial.
Relaciones	OBJ-1, OBJ-5

Cuadro 1.21: RNF-4: Accesibilidad

1.5. Matriz de rastreabilidad objetivo/requisitos

Requisito	OBJ-1	OBJ-2	OBJ-3	OBJ-4	OBJ-5
RF-1	X				X
RF-2	X				
RF-3		X			
RF-4			X		
RF-5			X		
RF-6				X	
RF-7				X	
RF-8					X
RNF-1		X			
RNF-2		X			
RNF-3				X	
RNF-4	X				X

Cuadro 1.22: Matriz de rastreabilidad objetivo/requisitos

Anexo 2

Análisis y diseño del sistema

La arquitectura de la biblioteca ha sido descrita en prosa en el capítulo 9, por lo que se va a evitar reiterar sobre ello en exceso.

Sin embargo, se pueden detallar determinadas interacciones y relaciones con diagramas más descriptivos, que ocupaban demasiado espacio en la memoria. A continuación se proveen algunos de estos diagramas más descriptivos.

2.1. Diagrama de componentes

A continuación se muestra la relación entre los principales componentes de la biblioteca:

- Aplicación externa
- Interfaz principal (`P2PSession` y `P2PSessionListener`)
- Implementación del protocolo (protocolo binario y seguridad)
- Plataforma (D-Bus y Android)
- Utilidades

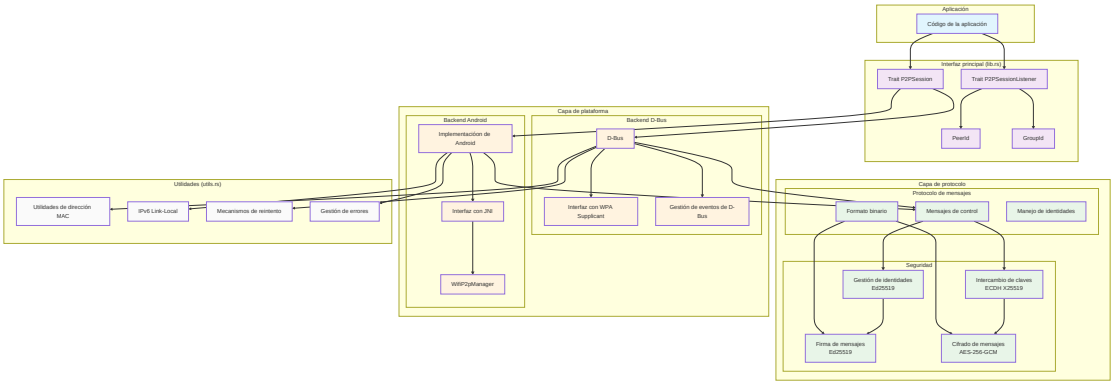

Figura 2.1: Componentes

2.2. Diagramas de secuencia

A continuación se muestran los flujos operacionales principales usando diagramas de secuencia.

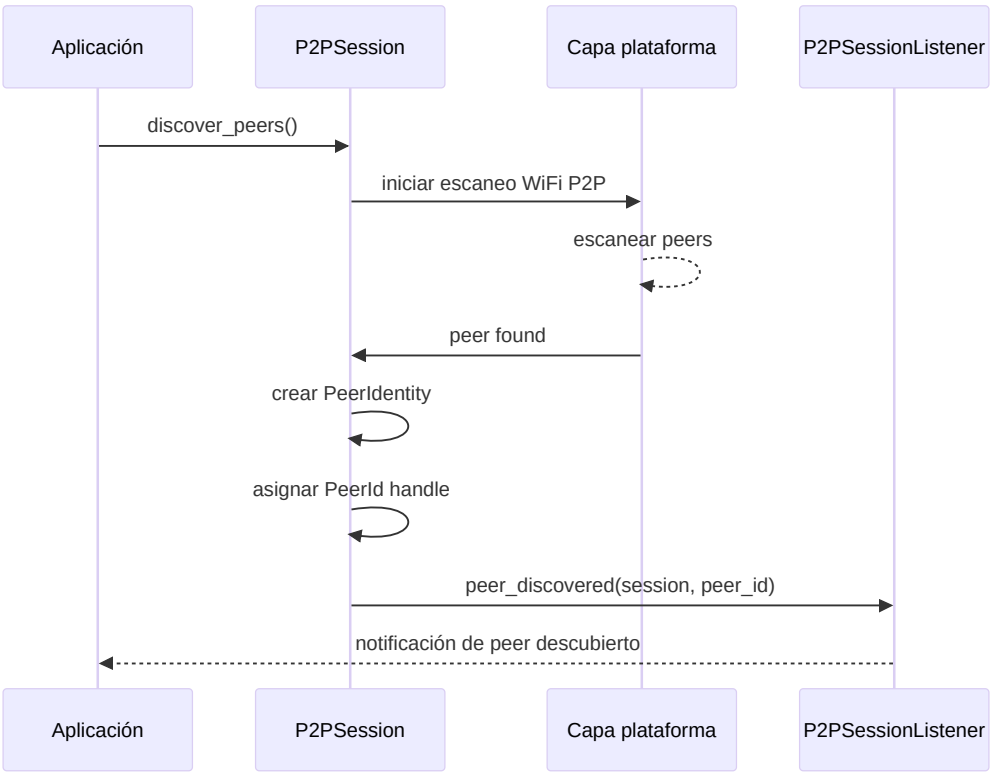

Figura 2.2: Descubrimiento de dispositivos

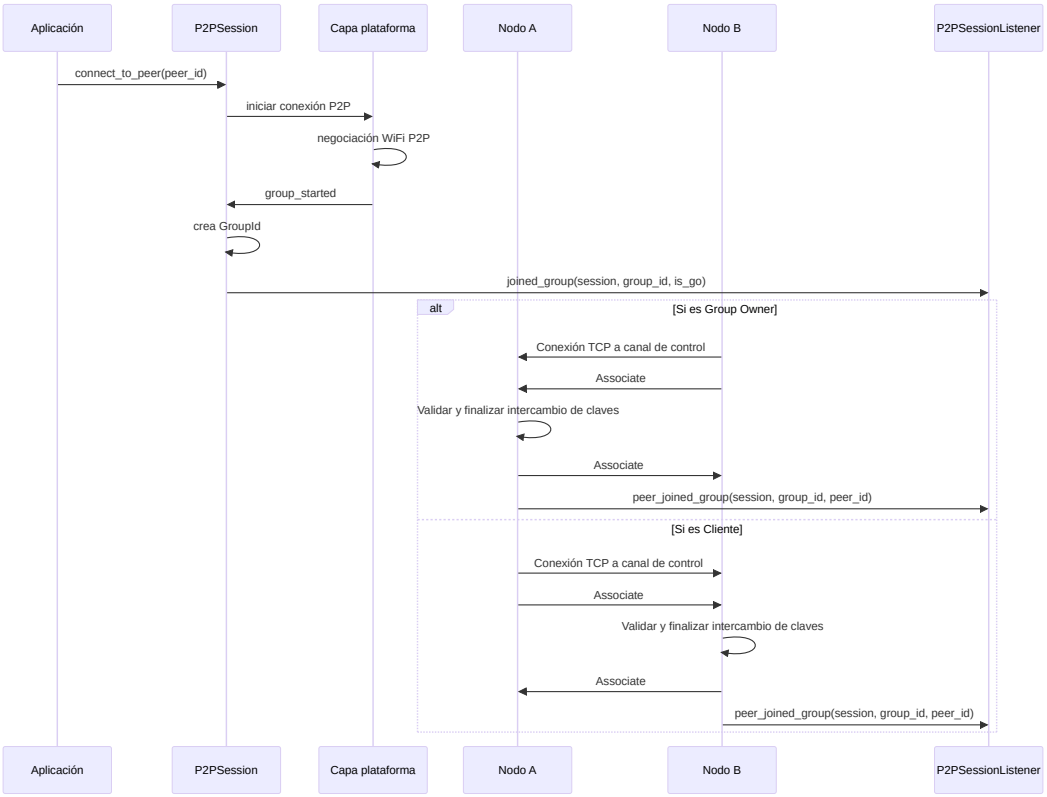

Figura 2.3: Establecimiento de conexión

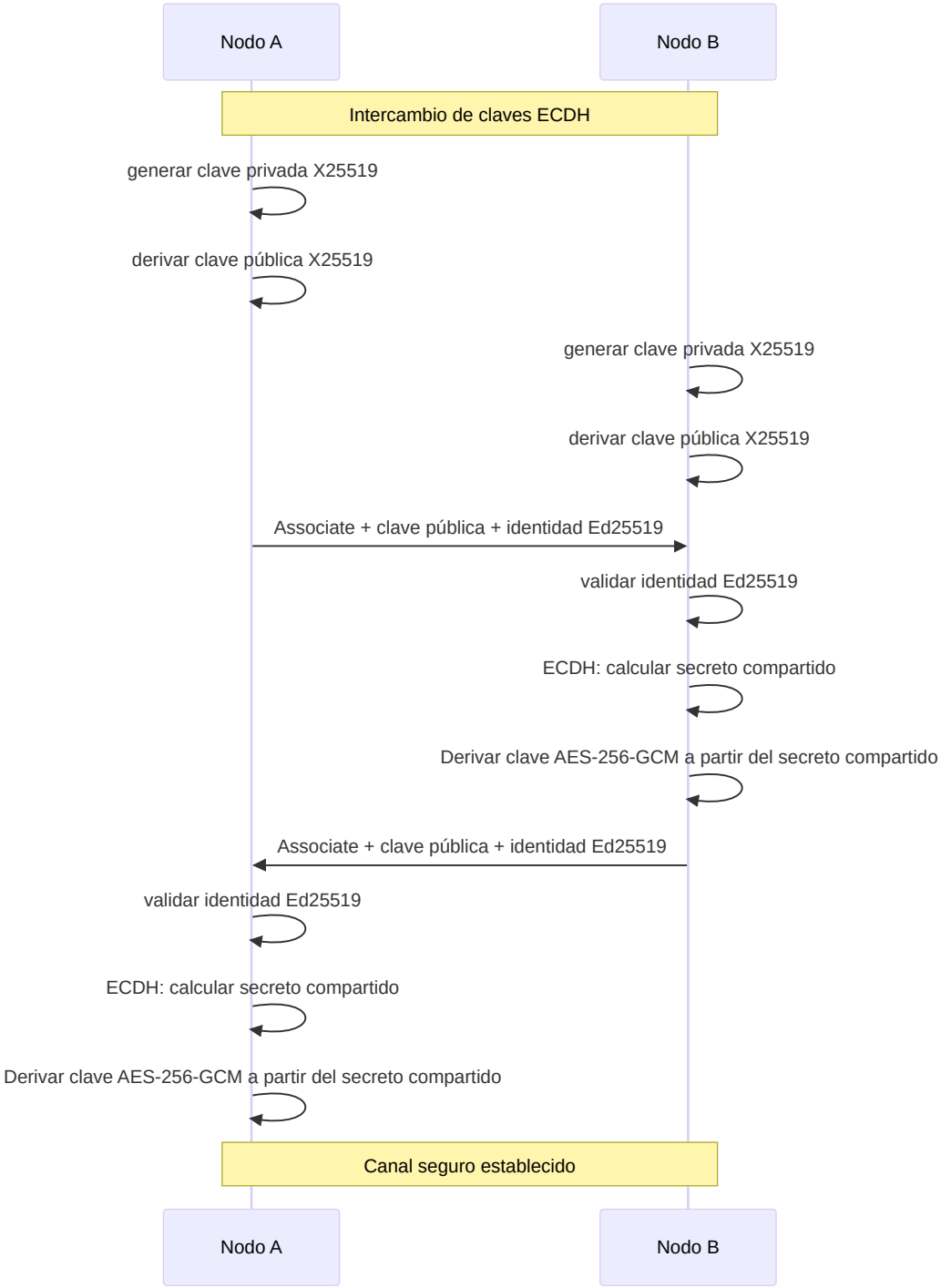

Figura 2.4: Intercambio de claves y establecimiento de un canal seguro

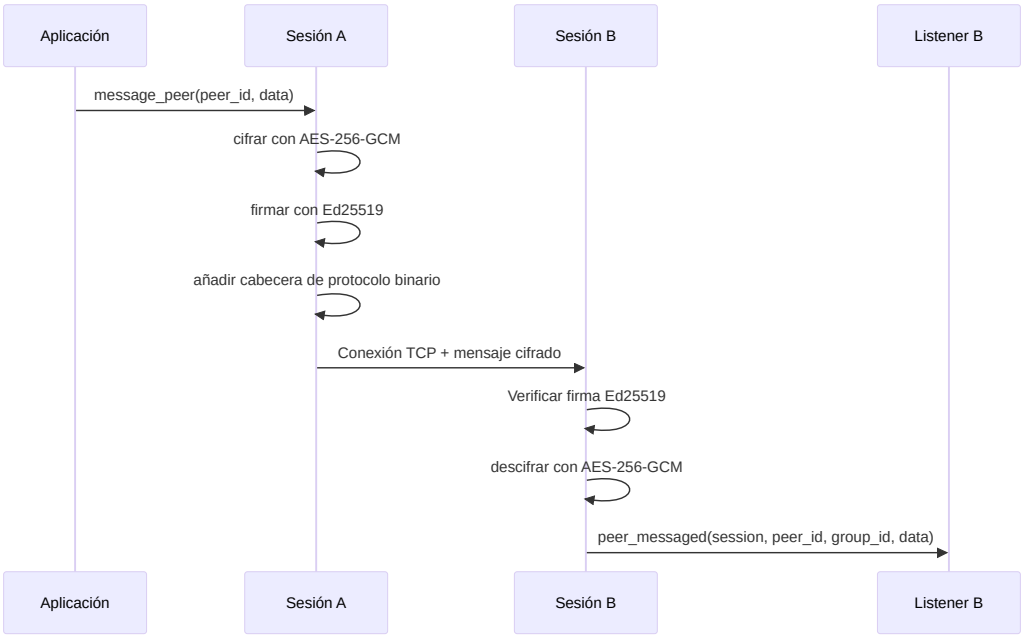

Figura 2.5: Intercambio de mensaje seguro

Anexo 3

Estimación del tamaño y esfuerzo

En el presente anexo se recoge la metodología de trabajo empleada en el desarrollo del proyecto, la planificación temporal de las tareas realizadas, las métricas empleadas para la planificación y sus estimaciones.

El proyecto se desarrollará siguiendo la metodología de trabajo ágil Scrum (ver apartado 5.1.1), una metodología que permite una planificación iterativa e incremental, facilitando la adaptación a los cambios y la mejora continua del proceso de desarrollo.

Esta metodología se adapta bien al proyecto, ya que permite que tanto el autor como los tutores puedan llevar un seguimiento del avance del proyecto, y modificar las tareas a realizar en función de los resultados obtenidos en cada iteración.

Por ejemplo, si una tarea resulta más compleja de lo previsto inicialmente, se podrá decidir si merece la pena dedicar más esfuerzo a esa tarea, o si es mejor sustituirla por una alternativa más fácil de implementar, o incluso eliminarla si no es esencial para el proyecto.

También evita la necesidad de una planificación detallada y exhaustiva al inicio del proyecto, que hubiera sido imposible dada la variabilidad de la cantidad de tiempo que el autor ha podido dedicar al proyecto (por su trabajo a tiempo completo y cargas familiares), y la gran cantidad de incógnitas que existían al comienzo del mismo.

3.1. Scrum

Este proyecto se ha desarrollado siguiendo el marco de trabajo Scrum[1]. A continuación se presenta una breve introducción a Scrum, basada parcialmente en la Guía de Scrum[45], documento que define el marco de trabajo Scrum y que se actualiza periódicamente.

[1]Aunque lo parezca, Scrum no es un acrónimo, sino un término tomado del rugby, que hace referencia a una formación de jugadores en el campo, en la que cada uno de los jugadores tiene un rol específico, y todos trabajan juntos para conseguir un objetivo común (el balón) [46]

También se explica, para cada una de las secciones, cómo se aplica esta parte del marco teórico al desarrollo del proyecto, en la práctica.

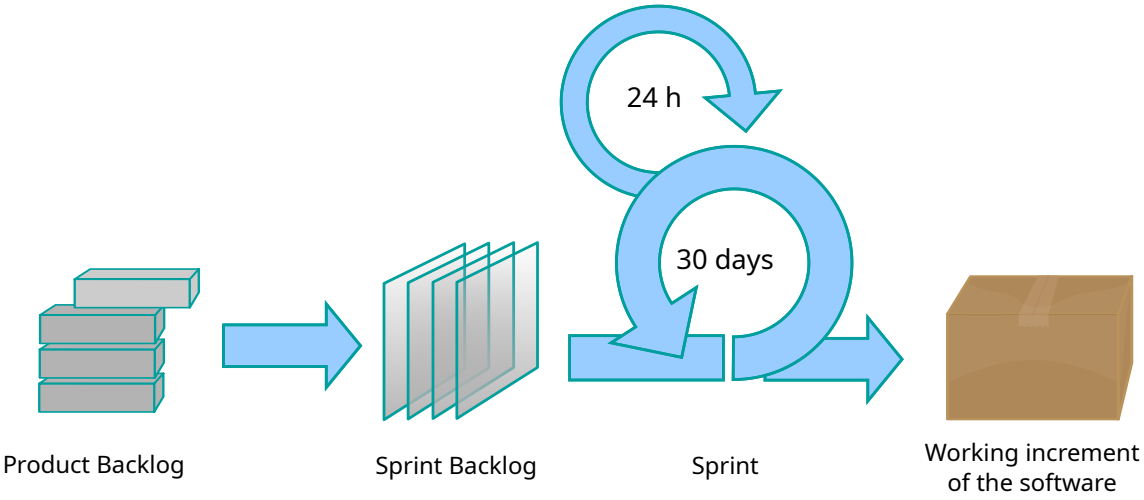

Figura 3.1: Diagrama gráfico del proceso Scrum (Fuente: Lakeworks)

3.1.1. Definición

Según los autores originales de Scrum, Schwaber y Sutherland [45], «Scrum es un marco [de trabajo] ligero que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptables para problemas complejos» (p. 3).

El enfoque del proceso Scrum es «iterativo e incremental», lo que significa que se realizan entregas periódicas de un producto que va evolucionando a lo largo del tiempo, y que se va adaptando a las necesidades del cliente.

3.1.2. Roles

En Scrum, existen tres roles principales: el *Product Owner*, el *Scrum Master* y el *Development Team*.

Product Owner

El *Product Owner* o dueño del producto es el responsable de maximizar el valor del producto resultante del trabajo del equipo Scrum. El dueño del producto es quien define el objetivo del producto, y quien gestiona el *Product Backlog*.

Scrum Master

El *Scrum Master* es el responsable de asegurar que Scrum se entiende y se aplica correctamente. Actúa de intermediario entre el *Product Owner* y el *Development Team*, facilitando la comunicación y la colaboración entre ambos, y ayudando a resolver los obstáculos que puedan encontrar durante el proceso de desarrollo.

Además, al delegársele las labores de seguimiento del flujo de trabajo, permite que el equipo de desarrollo se centre en progresar hacia el objetivo del producto.

Development Team

El *Development Team* o equipo de desarrollo es el grupo de personas que se encarga de desarrollar las funcionalidades de los incrementos del producto.

El equipo de desarrollo es autoorganizado, lo que significa que son ellos quienes deciden cómo llevar a cabo el trabajo, y son responsables de la calidad del producto que entregan. [45] (p. 7).

El equipo de desarrollo es multidisciplinar, y puede estar formado por desarrolladores, diseñadores, testers, etc. El equipo tiene todas las habilidades necesarias para generar los incrementos del producto.

Los miembros del equipo de desarrollo son iguales entre sí, y no hay jerarquías ni equipos de trabajo especializados.

El tamaño del equipo de desarrollo debe ser lo suficientemente grande como para completar el trabajo, pero lo suficientemente pequeño como para que la comunicación y la colaboración sean efectivas. La Guía de Scrum recomienda un tamaño de entre 3 y 9 personas [45] (p. 7)

3.1.3. Aplicación al Trabajo de Fin de Grado

En el caso del Trabajo de Fin de Grado, los *Product Owners* son los tutores del TFG, quienes definen el objetivo del producto y gestiona la pila del producto.

El autor cumple los roles tanto de Scrum Master como el equipo de desarrollo, asegurándose de cumplir con el marco de trabajo Scrum, hablando con los *Product Owners* para definir el objetivo del producto y la pila del producto, y desarrollando los incrementos del producto.

3.1.4. Flujo de trabajo

El trabajo en Scrum se organiza en sprints, que son «eventos de longitud fija de un mes o menos» [45] (p. 8), cuyo objetivo es entregar uno (o más) incrementos del producto. Al finalizar un sprint, se comienza el siguiente, volviendo a tener lugar todas las reuniones y eventos necesarios, y así sucesivamente hasta que se alcance el objetivo del producto.

Sprint Planning

Cada sprint comienza con una reunión de planificación, llamada *Sprint Planning*, en la que los integrantes del equipo definen el trabajo a realizar durante el sprint (confeccionando el *Sprint Backlog*) y establecen un objetivo para el mismo.

Daily Scrum

Tras la planificación, el equipo trabaja en las tareas definidas, y se reúne diariamente en una reunión llamada *Daily Scrum*, cuyo propósito es «inspeccionar el progreso hacia el Objetivo del Sprint y adaptar el Sprint Backlog según sea necesario, ajustando el próximo trabajo planeado» [45] (p. 9).

El *Daily Scrum* es una reunión breve, de máximo 15 minutos [45] (p. 9), en la que los desarrolladores comunican su progreso, los obstáculos que han encontrado y lo que planean hacer a continuación.

Sprint Review

Durante la Sprint Review (en español, revisión del sprint), que ocurre al final de cada sprint, como penúltimo evento, participan todos los roles de Scrum, y se revisa el trabajo realizado durante el sprint, el equipo presenta el incremento del producto.

Entonces se recoge la retroalimentación del dueño del producto o Product Owner, y se discute el progreso hacia el objetivo del producto y qué hacer a continuación.

Sprint Retrospective

Finalmente, tras la revisión del sprint, se realiza una reunión retrospectiva del sprint o Sprint Retrospective, en la que el equipo reflexiona sobre el trabajo realizado, y sobre qué cambios hacer para mejorar su eficacia y eficiencia de cara al siguiente sprint.

En cada sprint se planifica el trabajo a realizar (*Sprint Planning*), se trabaja y revisa el trabajo realizado (*Daily Scrum*) y, al final (pero aún dentro del sprint), se revisa el trabajo realizado (*Sprint Review*) y se reflexiona sobre el trabajo realizado y cómo mejorar para el siguiente sprint (*Sprint Retrospective*).

Aplicación al Trabajo de Fin de Grado

El caso del Trabajo de Fin de Grado es un caso particular, ya que el equipo de desarrollo está formado por una sola persona, el autor. Por esto, no se han realizado reuniones diarias (*Daily Scrum*).

Las reuniones de revisión del sprint (Sprint Review) se han llevado a cabo semanalmente, y han consistido en una revisión del trabajo realizado durante la semana, así como una revisión del progreso hacia el objetivo del producto, y una discusión sobre qué hacer a continuación.

3.1.5. Artefactos

Los artefactos de Scrum son los elementos que se utilizan para gestionar el trabajo y el progreso del equipo, y son tres: el *Product Backlog*, el *Sprint Backlog* y el *Incremento*. Cada uno de ellos se utiliza para conseguir llegar a una meta específica, y se actualizan periódicamente a lo largo del proceso de desarrollo.

Product Backlog o pila del producto

La pila del producto «es una lista emergente y ordenada de lo que se necesita para mejorar el producto» [45] (p. 11). Las tareas que se incluyen en la pila del producto son las que, durante la planificación del sprint, se seleccionan para formar parte del *Sprint Backlog*.

El objetivo del producto es un objetivo a largo plazo sobre el que se puede planificar qué tareas incluir en la pila del producto. Surge de la interacción con el dueño del producto o *Product Owner*.

Sprint Backlog o pila del sprint

La pila del sprint «se compone del objetivo sprint (por qué), el conjunto de elementos de trabajo pendiente de producto seleccionados para el Sprint (qué), así como un plan accionable para entregar el incremento (cómo)» [45] (p. 12).

Es decir, las tareas que se van a realizar durante el sprint, y que se seleccionan de la pila del producto durante la planificación del sprint. La desarrollan los desarrolladores, y se revisa (y reforma si fuese necesario) durante el *Daily Scrum*.

Increment o incremento del producto

Los incrementos son cada uno de los pasos (incrementales) que se dan para alcanzar el objetivo del producto. Son «aditivo[s] a todos los incrementos anteriores y verificado[s] a fondo, asegurando que todos los incrementos funcionen juntos» [45] (p. 12).

Son, además, utilizables, lo que significa que se pueden entregar al cliente, y que este puede utilizarlos para obtener valor del producto. Durante cada sprint, como se ha visto anteriormente, se pueden entregar uno o varios incrementos del producto.

Aplicación al Trabajo de Fin de Grado

Los artefactos Scrum se han mantenido intactos al aplicar el marco teórico al TFG, aunque no se ha mantenido un backlog continuamente durante la duración del proyecto, ya que sólo hay un desarrollador y generalmente se han planeado los sprints continuamente en base a la anterior.

Una planificación y estimación más detallada por adelantado hubiera sido tal vez beneficiosa pero, dada la naturaleza de investigación especialmente durante los primeros meses del proyecto, hubiera sido de utilidad limitada.

3.1.6. Estimaciones de tiempo

Antes de comenzar los sprint, el equipo de desarrollo debe estimar el tiempo que tardará en completar cada tarea de la pila del producto. Esto se puede hacer calculando el tiempo en horas que se tardará, directamente, o bien mediante unidades de medida relativas, como puntos de historia (*story points*) o tamaños de camiseta (*t-shirt sizes*).

Comenzando por los *story points*, según escribe Derek Davidson [20], «un punto de historia es una unidad de medida relativa, decidida y utilizada por equipos Scrum individuales, para proporcionar estimaciones relativas del esfuerzo necesario para completar los requisitos».

El uso de puntos de historia es una práctica común en Scrum, y permite estimar el esfuerzo necesario para completar una tarea de forma relativa, comparándola con otras tareas, en lugar de absoluta.

La ventaja de utilizar puntos de historia en vez de una estimación clásica de tiempo es que, según el coautor de la Guía de Scrum, Sutherland [50] «los puntos de historia dan estimaciones más precisas, reducen el tiempo de planificación drásticamente, predicen las fechas de lanzamiento con mayor precisión, y ayudan a los equipos a mejorar su desempeño».

Los tamaños de camiseta son otra forma de estimar el esfuerzo necesario para completar una tarea. Son muy parecidos a los puntos de historia, pero en lugar de utilizar una escala numérica, se utilizan tamaños de camiseta (XS, S, M, L, XL, XXL) para representar el esfuerzo necesario para completar la tarea.

Esta forma de estimación es más intuitiva, y puede ser más fácil de entender para personas que no están familiarizadas con Scrum o con la estimación de tiempo en general. Sin embargo, puede ser menos precisa que los puntos de historia, ya que no se puede comparar directamente el esfuerzo necesario para completar una tarea con el de otra.

Además, resultan más complicados de utilizar a la hora de desarrollar gráficos, ya que no se pueden utilizar directamente para calcular el trabajo pendiente o el trabajo realizado, y requieren una conversión a puntos de historia para este fin.

Aplicación al Trabajo de Fin de Grado

Para el Trabajo de Fin de Grado, se han utilizado *t-shirt sizes* como medida de estimación de esfuerzo, y no se ha realizado una contabilización exhaustiva del tiempo empleado en cada tarea.

Esto se debe a que se considera que lo importante es el esfuerzo necesario para completar una tarea, y no el tiempo que se tarda en completarla, siendo el tiempo una medida complicada de estimar, y que puede variar mucho, ya no solo dependiendo del esfuerzo invertido, sino también de factores externos como el el tiempo disponible para trabajar en la tarea.

3.2. Planificación temporal

En el presente capítulo se detallará la planificación temporal del proyecto, así como las métricas que se han utilizado para su elaboración.

Dadas las restricciones temporales, la planificación temporal ha sido bastante dinámica. Las reuniones de *Sprint Planning* se utilizaron para decidir las tareas a

completar durante el sprint, y estimar su esfuerzo, generalmente tratando de dividir las tareas de tal manera que pudieran ser ejecutadas durante el *sprint* en la medida de lo posible.

A continuación se presenta un resumen de alto nivel de las fases del proyecto, seguido de un resumen sprint a sprint.

3.2.1. Resumen general

El proyecto se ha desarrollado de manera relativamente consistente desde Octubre de 2024 a Agosto de 2025, con algunos parones intermitentes (generalmente por la amplia carga de trabajo del autor), especialmente en Abril de 2025.

Exploración inicial (Octubre de 2024)

Se realizó una exploración inicial de las capacidades de comunicación peer-to-peer de Android y Linux.

Se eligió tentativamente WiFi Direct como capa de transporte inicial, y se desarrolló un prototipo[2] de aplicación para Android, tras una primera familiarización con Android Studio.

Se determinó que para el primer prototipo se usaría Linux como plataforma soportada (ver apartado 12.2.1 y apartado 11.1.1).

Desarrollo inicial en Linux (Diciembre de 2024 a Enero de 2025)

Se realizaron los bloques iniciales usando D-Bus para comunicarse con wpa_supplicant. Una buena parte de esta fase fue diagnosticar problemas con los que no se contaron inicialmente (ver apartado 11.1.1 y apartado 7.2).

Desarrollo del protocolo principal (Febrero a Mayo de 2025)

Se consiguió un entorno de pruebas estable, se desarrolló todo el protocolo binario y un intercambio de mensajes básico usando TCP.

Se desarrolló también la implementación inicial de una interfaz de usuario para pruebas usando GTK.

Se determinaron las abstracciones principales, y se refactorizó la implementación de Linux con el objetivo de introducir soporte para Android a continuación.

Integración con Android (Junio a Julio de 2025)

Se desarrolló una aplicación de pruebas básica para Android, junto a toda la implementación de la librería y la integración con Java usando la JNI.

También se hicieron ajustes al protocolo para acomodar las limitaciones / restricciones en Android.

[2]prototipo: `https://github.com/emilio/android-wifip2p-test`

Finalmente, se ajustó la aplicación de pruebas para convertirla en la aplicación de demostración (el juego de 2048).

Criptografía y seguridad (Julio a Agosto de 2025)

Se implementó la identificación de clave pública y firma de mensajes inicialmente, y posteriormente se realizaron los ajustes necesarios al protocolo para realizar el intercambio de claves y el paso de mensajes cifrado.

Finalización de la documentación (Agosto de 2025)

Se habían hecho varias partes de la documentación como tareas auxiliares en otros sprints, y se habían documentado las cosas importantes de manera no estructurada en un fichero de texto en el repositorio, pero fue en Agosto cuando se formalizó toda la documentación.

3.2.2. Desglose detallado

A continuación se detallan los sprints donde hubo actividad. Generalmente la actividad puede ser comprobada via el log del sistema de control de versiones.

Sprint 1: Del 1 al 6 de Octubre de 2024

Se investigaron las diferentes alternativas para implementar la capa de transporte. Se eligió probar WiFi Direct como capa de transporte para el prototipo por las razones detalladas en el apartado 7.2.2.

Sprint 2: Del 7 al 13 de Octubre de 2024

Se desarrolló un prototipo de aplicación en Android que conectara dos dispositivos y enviara un mensaje entre ellos.

Tras la reunión de revisión del sprint se decidió afrontar la implementación inicial en Linux.

Sprint 3: Del 15 al 21 de Diciembre de 2024

Se creó el repositorio principal (independiente al prototipo). Se investigó el estado de la comunicación via D-Bus en Linux, y se crearon las primeras interfaces usando zbus, consiguiendo un listado de dispositivos adyacentes.

Las pruebas iniciales revelaron problemas inesperados (descritos en el apartado 7.2). Se encontraron problemas con zbus y se propusieron algunas soluciones (ver apartado 12.3.1).

Sprint 4: Del 30 de Diciembre de 2024 al 5 de Enero de 2025

Este sprint por razones obvias (navidad) no hubo reunión de *Sprint Planning*. Se depuraron los problemas de comunicación con D-Bus.

Sprint 5: Del 6 al 12 de Enero de 2025

Se hizo mucho progreso en la depuración, y se consiguió conectar el ordenador de sobremesa del autor con su portátil (ambos corriendo Linux), descubriendo problemas nuevos, como que WiFi direct en sí mismo no proporciona la dirección IP de los dispositivos, por lo que tiene que ser derivada de alguna otra manera. Ver apartado 7.2.5 para algunos de los detalles al respecto.

Sprint 6: Del 20 al 26 de Enero de 2025

Se mejoró la demo de DBus para soportar reintento de conexiones. También se depuró wpa_supplicant para intentar entender algunos de los problemas descritos anteriormente.

Sprint 7: Del 10 al 16 de Febrero de 2025

Se consiguió establecer una comunicación UDP entre dispositivos via direcciones de link local [36] (sección 5.3).

También se creó la infraestructura para la documentación del proyecto (portada del proyecto, `Makefile`, secciones iniciales con LaTeX...).

Sprint 8: Del 17 al 23 de Febrero de 2025

Se hizo progreso en poder probar la capa de transporte. Se descubrió la razón por la que el código funcionaba en uno de los ordenadores del autor pero no en el otro (ver apartado 7.2.5).

Se empezó a estructurar el código para prepararlo para ser reutilizado.

Sprint 9: Del 24 de Febrero al 2 de Marzo de 2025

Se avanzó en la documentación, creando el glosario y compartiendo la estructura con otros alumnos.

Sprint 10: Del 3 al 9 de Marzo de 2025

Se creó la infraestructura para la bibliografía de la memoria, y se hizo algo de progreso al respecto.

Se dividió la estructura del código en una biblioteca y una aplicación de ejemplo.

Sprint 11: Del 24 al 30 de Marzo de 2025

Se implementó el protocolo binario de paso de mensajes y se implementaron notificaciones para el manejo del grupo en la librería.

Sprint 12: Del 1 al 7 de Abril de 2025

Se elaboró una interfaz de paso de mensajes independiente a la capa de transporte.

Sprint 13: Del 5 al 11 de Mayo de 2025

Se depuró un error de memoria y se mejoró la infraestructura de pruebas para integrar rr (ver apartado 5.2.2) y ASan.

Sprint 14: Del 12 al 18 de Mayo de 2025

Se implementó la gestión básica de grupos, y se implementó funcionalidad para auto-unirse a un grupo en wpa_supplicant (ver apartado 12.3.1).

Sprint 15: Del 26 de Mayo al 1 de Junio de 2025

Se implementó una interfaz básica de pruebas usando GTK, para poder probar situaciones con más de dos dispositivos por grupo físico más fácilmente.

Sprint 16: Del 9 al 15 de Junio 2025

Se creó la aplicación básica de Android con un esqueleto básico del backend para Android.

Sprint 17: Del 16 al 22 de Junio 2025

Se integró mucho más profundamente la librería con Java, incluyendo poder incluir y llamar al código de Rust desde la aplicación. La biblioteca aún no era funcional.

Sprint 18: Del 23 al 29 de Junio 2025

Se consiguió una integración básica en Android y se creó infraestructura para poder testear Android con un sólo teléfono (comunicando el dispositivo Android con Linux) para poder hacer progreso sin tener dos dispositivos.

Se ajustó el descubrimiento de direcciones para soportar IPv4 también, ya que Android usa DHCP + IPv4 por defecto.

Sprint 19: Del 7 al 13 de Julio 2025

Se hizo algo de progreso en tener la librería funcional en Android, arreglando la gestión de grupos.

Sprint 20: Del 14 al 20 de Julio 2025

Se implementó la identidad y firma de mensajes, y se investigó el cifrado de mensajes también. Se implementaron varias APIs de Android también.

Sprint 21: Del 21 al 27 de Julio 2025

Se implementaron muchas mejoras de interfaz en Android y se implementó el juego 2048, que se demostró a los tutores en la retrospectiva del *sprint*.

Sprint 22: Del 5 al 11 de Agosto de 2025

Se empezó con la redacción más completa de la memoria, integrando también varias mejoras (soporte para código, etc) a la misma.

Sprint 23: Del 12 al 18 de Agosto de 2025

Se implementó y documentó el cifrado de mensajes punto a punto y el intercambio de claves. Se envió a `wpa_supplicant` una mejora necesaria para ello (ver apartado 12.3.1), y se verificó que la aplicación de Android funcionaba con mensajes cifrados.

Todo el tiempo desde entonces se ha empleado en la documentación del trabajo.

3.3. Conclusión

El uso de la metodología inspirada en Scrum en este proyecto ha permitido una planificación y desarrollo ágil del mismo, con una buena adaptación a los cambios y una mejora continua del producto.

No hay ninguna duda de que el uso de Scrum ha sido una buena elección para el proyecto. Las frecuentes reuniones de revisión de los sprints (semanales) han permitido una buena comunicación con los tutores y atajar muchos problemas antes de que se convirtieran en grandes obstáculos.

En la práctica, siendo un desarrollo individual, se han podido tomar atajos que en una aplicación más estricta de la metodología no sería posible (como saltarse sprints, o replanificar a mitad de sprint). Algunos de estos atajos recuerdan más a Kanban [11] que a Scrum, más orientado a un flujo de trabajo continuo.

El autor usa metodologías ágiles en su trabajo, y considera que usar una versión más estricta del Scrum como se podía haber hecho (requiriendo estructurar mucho más el backlog, anotar cada tarea con estimaciones, no replanificar a mitad de *sprint*) hubiera sido un error, especialmente dadas las restricciones temporales. Usar metodologías no ágiles tampoco hubiera sido factible, por las mismas razones.

Anexo 4

Plan de seguridad

La biblioteca ha sido diseñada para proveer seguridad *en profundidad*. Ya se han tocado muchos de estos detalles en la memoria (ver apartado 9.5, figura 2.4). Este anexo servirá como repaso general a los varios mecanismos de seguridad que se implementan para garantizar tanto la integridad como la privacidad de los mensajes.

4.1. Seguridad de la capa de transporte

La librería intencionadamente (ver cuadro 1.8,) *no asume que la capa de transporte sea segura*. Esto quiere decir que si remplazáramos WiFi Direct (la capa de transporte del prototipo) por una radio no cifrada, las propiedades de seguridad de la librería serían las mismas.

Dicho eso, existen medidas de seguridad a la hora de usar WiFi Direct en particular que tal vez sean dignas de mención.

Linux restringe el acceso a la gestión de redes P2P (ver apartado 7.2.4), y Android provee requiere de interacción para crear usuarios (ver figura 9.5).

WiFi-Direct generalmente utiliza WPS como medida de seguridad para prevenir que cualquier miembro externo se una: Es la aplicación o el sistema operativo el que tiene que aceptar la conexión.

4.2. Autenticidad de los mensajes

La autenticidad de los mensajes se garantiza con el firmado de los mismos usando la clave de identidad de la sesión (que es una clave ed25519[1]).

Como se ha discutido anteriormente (ver apartado 9.5), esta clave no se presume efímera.

[1] ed25519: `https://ed25519.cr.yp.to/`

4.3. Privacidad de los mensajes

La privacidad de los mensajes se garantiza con el cifrado AES-256-GCM, usando una clave de cifrado efímera para cada par de nodos que se comunican entre sí. Esta clave efímera se genera usando X25519 [31], una método de ECDH.

Adicionalmente, se usa un contador como protección ante ataques de reproducción (este es el `ring::aead::NonceSequence`). Sin embargo, no estoy convencido de que merezca la pena en la práctica y podría ser re-evaluado en un futuro ya que, al menos con la implementación actual, puede hacer que paquetes perdidos acaben rompiendo la comunicación, efectivamente.

4.4. Integridad de los mensajes

La integridad de los mensajes se garantiza por partida doble. Primero, con la susodicha firma. Segundo, con el cifrado AES-256-GCM.

Anexo 5

Otros anexos

Por conveniencia para el lector, se adjunta junto a la memoria la documentación de la librería tanto para Linux como para Android, también disponible en el sitio web personal del autor[1], generados con `cargo doc` y `cargo doc --target x86_64-linux-android` respectivamente.

La razón por la que hay dos directorios diferentes es que rustdoc no lidia bien con la compilación condicional[2].

[1] sitio web personal del autor: `https://crisal.io/ngn`

[2] no lidia bien con la compilación condicional: `https://github.com/rust-lang/rust/issues/1998`