UNIVERSITAT DE BARCELONA

Trabajo final de grado

GRADO DE INFORMÁTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

COMUNICACIÓN DESCENTRALIZADA Y SEGURA A TRAVÉS DE TOR

Autor: Joel Otero Martín

Director:	Dr. Raul Roca
Realizado en:	Departament de Matemàtiques i Informàtica

Barcelona, 12 de junio de 2023

Resumen

Este proyecto de TFG tiene como objetivo aprovechar la capacidad de Tor para ofrecer privacidad y anonimato en línea, y desarrollar un protocolo P2P distribuido y anónimo que ofrezca una alternativa segura y descentralizada a los servicios de mensajería y compartición de archivos en línea. El proyecto aborda importantes desafíos técnicos y de seguridad, y tiene el potencial de ofrecer una solución valiosa para aquellos usuarios que buscan proteger su privacidad.

Resum

Aquest projecte de TFG té com a objectiu aprofitar la capacitat de Tor per oferir privacitat i anonimat en línia, i desenvolupar un protocol P2P distribuït i anònim que ofereixi una alternativa segura i descentralitzada als serveis de missatgeria i compartició de fitxers en línia. El projecte aborda importants desafiaments tècnics i de seguretat, i té el potencial d'ofereix una solució valuosa per aquells usuaris que busquen protegir la seva privacitat.

Abstract

This Bachelor's thesis project aims to leverage Tor's capability to provide online privacy and anonymity, and to develop a distributed and anonymous P2P protocol that offers a secure and decentralized alternative to online messaging and file-sharing services. The project addresses significant technical and security challenges, and has the potential to offer a valuable solution for users seeking to protect their privacy.

Agradecimientos

En primer lugar quiero agradecer a mis amigos y familiares, quienes me han brindado su apoyo incondicional y su ánimo en todo momento, haciéndome sentir arropado y motivado durante todo este proceso. Sus palabras de aliento y su confianza en mí han sido fundamentales para lograr este objetivo.

También me gustaría agradecer a todos aquellos profesionales y expertos en la materia que me han brindado su colaboración. Sus conocimientos y experiencia han sido de gran ayuda para la realización de este trabajo.

Finalmente, quisiera extender mi reconocimiento a la Facultad de Matemáticas e Informática de la Universidad de Barcelona, por brindarme la oportunidad de desarrollar este proyecto de investigación y por ofrecerme una formación de calidad en la que he adquirido los conocimientos y habilidades necesarios para enfrentar nuevos retos en el futuro.

¡Gracias a todos por hacer posible este logro!

Índice

1. Introducción

Este proyecto se centrará en el desarrollo de DeProtocol, una libreria de python lista para utilizar por terceras personas para construir sus propias aplicaciones manteniendo un protocolo de comunicación común.

1.1. Motivación

En la era digital en la que vivimos, la privacidad y la seguridad en línea son cada vez más importantes. La vigilancia masiva, la censura y el seguimiento en línea son preocupaciones comunes para muchos usuarios, lo que ha llevado a un aumento en la demanda de herramientas y servicios que protejan la privacidad y la identidad en línea.

En este contexto, la red Tor ha emergido como una herramienta popular para proteger la privacidad y el anonimato en línea. Tor es una red de comunicaciones distribuida y anónima que permite a los usuarios navegar por la web de manera anónima y segura. El enrutamiento de la red Tor se basa en la idea de usar múltiples capas de cifrado y enrutamiento para ocultar las conexiones y los datos de los usuarios, lo que hace que sea difícil para los atacantes rastrear y monitorear las actividades de los usuarios.

A pesar de los beneficios que ofrece Tor, la mayoría de los servicios en línea todavía se basan en un modelo centralizado, lo que significa que los datos y las comunicaciones de los usuarios se almacenan y controlan en un servidor central. Esto plantea problemas de privacidad y seguridad, ya que los usuarios deben confiar en el proveedor del servicio para proteger sus datos y mantener su privacidad. Además, los servicios centralizados también son vulnerables a ataques y censura, lo que puede poner en peligro la privacidad y la libertad de expresión de los usuarios.

Para abordar estos problemas, se propone desarrollar un protocolo de comunicación peer-to-peer (P2P) seguro y anónimo que funcione sobre la red Tor. La idea es aprovechar la capacidad de Tor para ocultar las conexiones y los datos de los usuarios, y crear una red P2P distribuida que ofrezca privacidad y anonimato a los usuarios.

En resumen, este proyecto de TFG tiene como objetivo explorar y desarrollar una alternativa segura y descentralizada a los servicios de mensajería y compartición de archivos en línea, utilizando Tor como plataforma para una red P2P distribuida y anónima. La idea es proporcionar una alternativa a los servicios centralizados que ofrecen un mayor grado de privacidad y seguridad para los usuarios.

El proyecto aborda importantes desafíos técnicos y de seguridad, como la resolución de problemas relacionados con la escalabilidad, la redundancia y el enrutamiento en una red P2P distribuida, así como la protección de la privacidad y la identidad de los usuarios en una red anónima como Tor. Además, es importante identificar y mitigar posibles vulnerabilidades de seguridad en el protocolo y en la aplicación, como ataques de denegación de servicio, suplantación de identidad y divulgación de información sensible.

1.2. Objetivos

Para este proyecto, hemos establecido varios objetivos principales, otros extras se han ido definiendo en el transcurso del desarrollo del proyecto pero por motivos de limitación de tiempo se han quedado fuera del foco principal.

Los objetivos de este proyecto son los siguientes:

- **Investigación y evaluación de desafíos técnicos y de seguridad:** El objetivo principal de este proyecto es investigar y evaluar los desafíos técnicos y de seguridad asociados con la creación de un protocolo P2P distribuido y anónimo sobre la red Tor. La red Tor ofrece un alto nivel de privacidad y anonimato en línea, pero su uso para servicios P2P presenta desafíos únicos. Esto incluye garantizar la anonimidad de los usuarios y la privacidad de sus comunicaciones en un entorno P2P distribuido. Asimismo, la implementación de un protocolo P2P seguro y escalable sobre la red Tor también presenta desafíos técnicos, como la gestión de conexiones y la escalabilidad del sistema.
- **Diseño e implementación del protocolo P2P distribuido y anónimo:** El segundo objetivo de este proyecto es diseñar e implementar un protocolo P2P distribuido y anónimo que aborde los desafíos técnicos y de seguridad identificados. El protocolo se implementará utilizando las librerías de Tor en Python y se integrará con la red Tor para proporcionar una alternativa segura y descentralizada a los servicios de mensajería y compartición de archivos en línea. Se explorarán diferentes soluciones para garantizar la privacidad y anonimato de los usuarios en el entorno P2P distribuido, incluyendo el uso de cifrado de extremo a extremo, la identificación de usuarios mediante pseudónimos y la implementación de mecanismos de firmado y verificación.
- **Garantizar la escalabilidad del protocolo:** Otro objetivo clave del proyecto es garantizar la escalabilidad del protocolo. A medida que más desarrolladores comiencen a utilizar el protocolo para sus aplicaciones, este debe tener la capacidad de ser extendido y integrado en otros proyectos sin comprometer la privacidad y la seguridad de los usuarios. Además, la escalabilidad es importante para permitir la adopción del protocolo a gran escala, lo que puede ser crucial para el éxito a largo plazo del proyecto.
- **Extra: Evaluación exhaustiva de los resultados:** Durante el desarrollo nos hemos dado cuenta de la posiblidad de expansión de dicho protocolo a otros campos, como el uso habitual del mismo para mensajería instantanea y compartición de archivos. Por eso mismo un objetivo extra sería la evaluación exhaustiva de la eficacia del protocolo, haciendo un estudio de rendimiento, mercado y seguridad para determinar si el protocolo podría llegar a ser una alternativa viable y efectiva en nuestras comunicaciones del día a día.

- **Extra: Anvanzar en el campo de seguridad y privacidad online:** Además de tener en cuenta el bajo uso actual de Tor para las comunicaciones diarias y las limitadas facilidades para su implementación, también nos hemos propuesto contribuir al avance del campo de seguridad y privacidad en las comunicaciones online que surgen en el día a día. Reconocemos la importancia de fortalecer la protección de los usuarios en sus interacciones en línea, y nuestro objetivo es brindar soluciones que promuevan una mayor seguridad y privacidad en estas comunicaciones cotidianas.

1.3. Planificación

Este proyecto abarca múltiples objetivos de gran importancia, cuyo desarrollo requiere un margen de tiempo adecuado para hacer frente a posibles contratiempos imprevistos. Teniendo esto en cuenta, hemos elaborado un plan de desarrollo detallado, como se muestra en la Figura 1, donde se desglosan las diversas tareas que se han ido completando en el transcurso de nuestro proyecto. Es importante destacar que el diseño y la implementación han tomado más tiempo del inicialmente previsto, debido a ciertas variaciones en la definición de objetivos y alcance. En concreto, se produjeron cambios en los objetivos, como la eliminación del desarrollo de un chat, y en su lugar, se priorizó la creación de una biblioteca para utilizar nuestro protocolo. Estas adaptaciones en el enfoque del proyecto han contribuido a su progreso y éxito.

	Tarea	Duración	Inicio
1	Elaboración del proyecto: DeProtocol	7 meses	Noviembre
2	Información y desarrollo de memória	7 meses	Noviembre
3	Investigación y recopilación de información	4 meses	Noviembre
4	Definición de los objetivos y alcance del proyecto	3 semanas	Febrero
5	Desarrollo de la memória	3 meses	Marzo
6	Diseño y implementación del protocolo	4 meses	Febrero
7	Diseño de la arquitectura y estructura del proyecto	1 mes	Febrero
8	Implementación del protocolo y pruebas	2 meses	Marzo
9	Análisis de los resultados y ajustes necesarios	2 semanas	Mayo
10	Diseño de un front-end para pruebas	1 semana	Mayo
11	Implementación del front-end de prueba	1 semana	Mayo

Figura 1: Diagrama de Gantt sobre la planificación de DeProtocol.

La planificación se ha estructurado en dos secciones principales, cada una abordando aspectos específicos del proyecto. La primera sección se centra principalmente en la recopilación de información y el desarrollo de la memoria, identificada como Tarea 2 en nuestro plan. La segunda sección (Tarea 6), por su parte, se enfoca en el diseño e implementación del protocolo y la biblioteca correspondientes. Al dividir el proceso de esta manera, podemos dedicar la atención necesaria a cada etapa del proyecto, asegurando una gestión efectiva y una ejecución exitosa.

Empezamos con las diferentes tareas relativas a la primera sección:

Tarea 3. **Investigación y recopilación de información:** Inicialmente, llevaremos a cabo un minucioso análisis de la tecnología conocida como "The Onion Router"(TOR), con el propósito de examinar detalladamente su flujo de ejecución, protocolo de encriptación y enrutamiento. De esta manera, podremos identificar y comprender a fondo el funcionamiento de esta red. Asimismo, exploraremos las distintas aplicaciones y bibliotecas disponibles que permiten interactuar con dicha red, evaluando su viabilidad e idoneidad para integrarlas en nuestro proyecto de comunicación descentralizada. Este enfoque exhaustivo nos brindará una base sólida para tomar decisiones fundamentadas y optimizar nuestra implementación.

Tarea 4. **Definición de los objetivos y alcance del proyecto:** Uno de los aspectos de mayor relevancia radica en la adecuada y viable definición de los objetivos que aspiramos alcanzar mediante nuestro proyecto. Es fundamental contar con una perspectiva realista acerca del alcance que poseemos para su desarrollo, teniendo en consideración las diversas tecnologías a las que tenemos acceso. La correcta delimitación de estos elementos resulta de vital importancia, ya que sentará las bases para el éxito y la eficacia de nuestra iniciativa, permitiéndonos gestionar adecuadamente los recursos disponibles y maximizar los resultados obtenidos. En este sentido, es imperativo analizar y evaluar meticulosamente las capacidades tecnológicas a nuestra disposición, asegurándonos de seleccionar aquellas que mejor se adecuen a nuestras necesidades y posibilidades.

Tarea 5. **Desarrollo de la memória:** Como elemento crucial dentro del proyecto, resulta imperativo contar con un amplio margen de tiempo para establecer y desarrollar una memoria exhaustiva que refleje de manera evidente el trabajo llevado a cabo en este proyecto. A través de esta memoria, se documentarán y presentarán de forma detallada las decisiones tomadas, los diseños desarrollados, la implementación realizada y los análisis efectuados, así como los resultados obtenidos. Esta completa y bien estructurada memoria constituirá una pieza fundamental para la comprensión y valoración del alcance y los logros alcanzados en este proyecto.

Teniendo bien definidas estas tareas, ahora vamos con las tareas que permitiran el desarrollo, diseño e implementación de nuestro protocolo:

Tarea 7. **Diseño de la arquitectura y estructura del proyecto**: Dentro de esta tarea específica, nos dedicaremos a un proceso detallado y meticuloso para diseñar las diversas arquitecturas necesarias que el proyecto demanda. Este enfoque abarcará una amplia gama de aspectos clave, incluyendo los flujos de comunicación, la conexión con la red, el intercambio de datos, las estructuras de datos pertinentes y los patrones de diseño que garanticen la extensibilidad y permitan que nuestra biblioteca sea utilizada por terceros de manera efectiva. Además, se prestará especial atención a la implementación de mecanismos sólidos de encriptación que salvaguardarán la integridad y la seguridad de los datos manejados. Al adoptar este enfoque exhaustivo, nos aseguramos de

abordar todos los aspectos cruciales para el éxito del proyecto, proporcionando una base sólida y confiable para su implementación y utilización futura. Este proceso de diseño cuidadoso y minucioso permitirá alcanzar resultados óptimos y promover la adaptabilidad y la escalabilidad del sistema en su conjunto.

Tarea 8. **Implementación del protocolo y pruebas:** Durante esta etapa crucial del proyecto, se dará vida a todo el trabajo previo a través de la implementación práctica. Aquí, cada idea y concepto diseñado se materializará en código concreto, permitiendo así que la visión del proyecto se haga realidad. Además, es importante reconocer que el desarrollo de una aplicación conlleva la posibilidad de encontrar nuevos desafíos y generar ideas adicionales en el proceso. Por tanto, se prevé y se incluyen modificaciones de diseño con el objetivo de lograr de manera más efectiva nuestros objetivos. Esta flexibilidad nos brinda la oportunidad de adaptarnos a los cambios y mejoras necesarias para asegurar un resultado óptimo.

Tarea 9. **Análisis de los resultados y ajustes necesarios:** Una vez finalizada nuestra versión definitiva, es crucial llevar a cabo una exhaustiva comprobación para asegurarnos de que hemos obtenido el resultado esperado: una biblioteca en Python que pueda ser utilizada para desarrollar nuevas aplicaciones. Además, se realizarán análisis adicionales para evaluar la eficacia de nuestro proyecto. Durante esta etapa, también se realizan ajustes y correcciones en el código e implementación, abordando posibles aspectos que pudieron haber sido pasados por alto. Esta fase de revisión y mejora nos permite garantizar la calidad y funcionalidad óptima de nuestra biblioteca, así como detectar y solucionar posibles problemas o mejoras adicionales necesarias para su despliegue exitoso.

Tarea 10. **Diseño de un front-end para pruebas:** Aunque no se había contemplado inicialmente, hemos decidido agregar esta tarea para diseñar la arquitectura de un pequeño front-end que demostrará la aplicación de nuestra biblioteca en un entorno real".

Tarea 11. **Implementación del front-end de prueba:** Dentro de un plazo de tiempo reducido, llevaremos a cabo una implementación mínima de la arquitectura diseñada con el objetivo de mostrar un front-end funcional que se ajuste a nuestro proyecto.

1.4. Estado del arte

La comunicación segura y privada a través de la red es un tema de gran importancia en la actualidad, especialmente en un mundo cada vez más interconectado en el que la privacidad y la protección de datos personales son esenciales. En este sentido, la red Tor ha demostrado ser una herramienta útil para garantizar el anonimato y la privacidad de los usuarios al enmascarar la dirección IP de origen de los paquetes de datos y encriptar el tráfico que circula por la red.

En cuanto a la implementación de protocolos peer-to-peer (p2p), existen numerosas alternativas disponibles, como BitTorrent, Kademlia y Chord, que han demostrado ser eficientes y escalables para la transferencia de archivos en una red descentralizada. Sin embargo, muchos de estos protocolos no ofrecen una protección adecuada de la privacidad y la identidad de los usuarios.

Recientemente, se han propuesto varias soluciones para abordar estos problemas de privacidad en las redes p2p. Por ejemplo, se ha desarrollado el protocolo GNUnet, que utiliza técnicas de anonimización y encriptación para garantizar la privacidad y la seguridad de los usuarios en la red. Asimismo, el protocolo Onion Routing ha sido utilizado para diseñar redes p2p como I2P y Ricochet, que también se enfocan en la privacidad y el anonimato.

En el ámbito de la programación, Python es un lenguaje de programación muy popular para el desarrollo de aplicaciones y herramientas. Existen numerosas bibliotecas y herramientas disponibles en Python para la implementación de protocolos de red y la comunicación entre nodos.

La librería será compatible con los sistemas operativos Windows, Linux y MacOS, y se integrará con otras librerías de encriptación, verificación y firma de comunicaciones. Asimismo, se utilizarán herramientas de testing como pytest y behave para garantizar la calidad y robustez del código.

En resumen, la implementación de una librería de Python que permita a los desarrolladores construir sus propias aplicaciones de mensajería y comunicación P2P sobre la infraestructura de Tor, es una solución interesante y necesaria en el contexto actual de la seguridad y privacidad en línea. La librería propuesta hará uso de las herramientas y técnicas más avanzadas disponibles en la plataforma de Tor, lo que garantizará su eficiencia, seguridad y confiabilidad.

2. Tecnologias

En la sección de Tecnologías, nuestro objetivo es brindar una introducción a cada una de las tecnologías que vamos a utilizar en el proyecto. Esta categoría servirá como punto de partida para comprender y contextualizar cada tecnología, lo que facilitará las explicaciones y discusiones posteriores.

2.1. Peer to Peer

2.1.1. Introducción

Una red peer-to-peer (P2P) es un tipo de red de computadoras descentralizada donde todos los nodos de la red tienen la misma capacidad y responsabilidad para proveer y consumir servicios. En una red P2P, los dispositivos conectados pueden actuar tanto como clientes como servidores, lo que significa que cada nodo puede proporcionar servicios y recursos a otros nodos de la red, y también puede solicitar servicios y recursos de otros nodos.

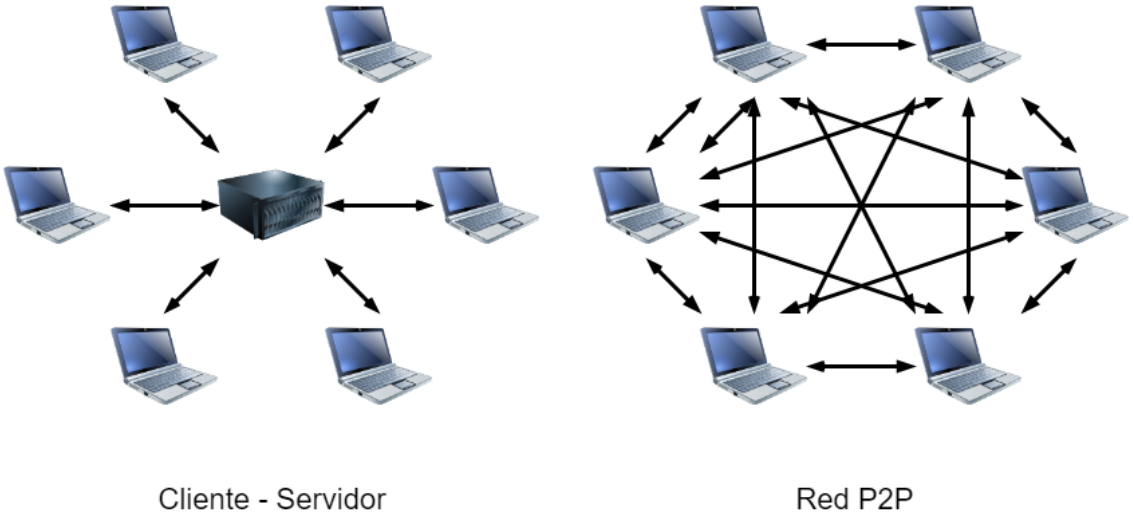

Figura 2: Representación de una red Cliente-Servidor y una red P2P.

A diferencia de las redes centralizadas, donde hay un servidor central que controla el tráfico de la red, las redes P2P son descentralizadas y no dependen de un único punto de fallo (Figura 2). En cambio, la carga de trabajo se distribuye entre todos los nodos de la red, lo que permite que la red funcione de manera más eficiente.

2.1.2. Utilidad

Las redes P2P se utilizan en una variedad de aplicaciones, como compartir archivos, juegos en línea, mensajería instantánea y telefonía VoIP. En una red P2P, los nodos pueden conectarse directamente entre sí o a través de una red intermedia, como una red overlay. Estas redes overlay proporcionan una capa adicional de abstracción y permiten a los nodos comunicarse de manera más efectiva.

En una red P2P, los nodos pueden actuar como clientes y solicitar recursos, como archivos o información, de otros nodos. Los nodos también pueden actuar como servidores y proporcionar recursos a otros nodos. Esto permite que la red funcione de manera más eficiente, ya que los recursos se distribuyen entre todos los nodos de la red, y no dependen de un servidor central.

2.1.3. Escalabilidad

Una de las ventajas de las redes P2P es su capacidad de escalar de manera efectiva, ya que cada nodo en la red puede actuar como un punto de conexión adicional para otros nodos. Además, las redes P2P son más resistentes a los ataques, ya que no dependen de un único punto de fallo.

2.1.4. Aplicación real

Son muchas las aplicaciones existentes hoy día que utilizan esta tecnología para compartir la información y recursos entre sus usuarios. Algunos ejemplos de los más populares son BitTorren, Skype, Bitcoin, Dropbox y muchos otros. Estas aplicaciones tienen millones de usuarios en todo el mundo y demuestran la versatilidad y el potencial de la tecnología P2P. Desde compartir archivos hasta llamadas de voz y video, pasando por transacciones financieras y almacenamiento en la nube, el uso de la tecnología P2P ofrece una alternativa descentralizada y eficiente a los modelos centralizados tradicionales.

2.2. The Onion Router

2.2.1. Introducción

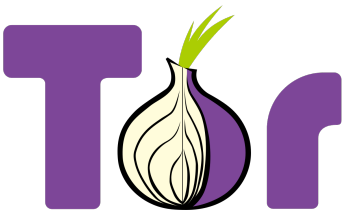

Figura 3: Logo de TOR.

En la actualidad, la privacidad y la seguridad en línea son dos de los temas más importantes en el ámbito de la tecnología y la comunicación. Con el creciente uso de internet y las tecnologías digitales, se han creado numerosas herramientas y técnicas para proteger la privacidad de los usuarios y garantizar la seguridad de sus datos. Una de estas herramientas es Tor (Figura 3), una red de comunicaciones descentralizada que se utiliza para mantener el anonimato y la privacidad de las comunicaciones en línea.

2.2.2. ¿Qué es Tor?

Tor es una red de comunicaciones en línea que se utiliza para proteger la privacidad y la seguridad de los usuarios en internet. Se basa en una serie de servidores que actúan como nodos en la red, que se encargan de enrutar el tráfico de datos de forma anónima. Tor es un acrónimo de "The Onion Router", y se llama así porque utiliza múltiples capas de cifrado, como las capas de una cebolla, para proteger el tráfico de datos.

2.2.3. Contexto histórico

El proyecto Tor se inició en el año 2002 como una iniciativa de la Marina de los Estados Unidos para proteger la privacidad y la seguridad de las comunicaciones gubernamentales en línea. Desde entonces, el proyecto ha sido desarrollado y mantenido por la Tor Project, una organización sin fines de lucro dedicada a la investigación y desarrollo de tecnologías de privacidad en línea.

2.2.4. ¿Como funciona?

La red Tor funciona mediante la utilización de una serie de nodos, que son servidores que se encargan de enrutar el tráfico de datos a través de la red. Cada vez que un usuario se conecta a la red Tor, su conexión es enrutada a través de una serie de nodos aleatorios, que sólo conocen la identidad del nodo anterior y posterior. De esta manera, se oculta la ubicación y la identidad del usuario final.

La seguridad y privacidad en la red Tor se logra mediante la utilización de técnicas de cifrado de extremo a extremo para proteger la integridad de los datos transferidos. Además, los nodos de la red Tor están configurados de tal manera que no pueden conocer la identidad o la ubicación del usuario final, lo que garantiza un alto nivel de privacidad y anonimato. Esta es ampliamente utilizada en todo el mundo para proteger la privacidad y seguridad en línea, especialmente en áreas con vulnerabilidad en la libertad de expresión y derechos humanos.

2.2.5. Servicios ocultos

Los servicios ocultos de Tor son una parte fundamental de la red Tor y se basan en una tecnología llamada Hidden Service Protocol (HSP), que permite a los servidores ofrecer servicios sin revelar su dirección IP real. En lugar de utilizar direcciones IP públicas, los servicios ocultos utilizan direcciones .onion, que son cadenas de caracteres alfanuméricos generadas criptográficamente.

Cuando un servidor configura un servicio oculto en Tor, genera una clave criptográfica llamada clave de servicio y la guarda en su sistema. Esta clave se utiliza para calcular la dirección .onion única asociada al servicio. La dirección .onion es esencialmente un hash de la clave de servicio, lo que garantiza que solo aquellos que poseen la clave correcta puedan acceder al servicio.

Cuando un usuario desea acceder a un servicio oculto de Tor, utiliza un cliente de Tor para conectarse a la red. El cliente solicita la dirección .onion del servicio deseado y la utiliza para establecer una conexión encriptada con el servidor que aloja el servicio. Toda la comunicación entre el usuario y el servidor se enruta a través de la red Tor, lo que proporciona un alto nivel de anonimato y privacidad.

Los servicios ocultos de Tor han sido utilizados para una variedad de propósitos legítimos, como la protección de la libertad de expresión en países represivos, la comunicación segura y privada para periodistas y activistas, y la creación de sitios web anónimos. Sin embargo, también han surgido preocupaciones sobre el uso de los servicios ocultos para actividades ilegales, como el tráfico de drogas, la pornografía infantil y la venta de armas ilegales.

Es importante tener en cuenta que aunque los servicios ocultos de Tor proporcionan un alto grado de anonimato y privacidad, no son completamente invulnerables. Hay técnicas de análisis de tráfico y ataques sofisticados que pueden comprometer la identidad de los usuarios o los servidores ocultos. Además, es fundamental tener precauciones adicionales al utilizar servicios ocultos, como asegurarse de que las comunicaciones sean encriptadas end-to-end y evitar revelar información personal que pueda vincularse a la identidad real.

En resumen, los servicios ocultos de Tor son una parte esencial de la red Tor que permite a los usuarios acceder y proporcionar servicios en línea de forma anónima y sin revelar su identidad. Si se utilizan de manera responsable, pueden ser una herramienta valiosa para proteger la privacidad y la libertad de expresión en el mundo digital. Sin embargo, también es necesario tener en cuenta los riesgos y tomar medidas adicionales para mantener la seguridad y evitar el uso indebido de estos servicios.

Es crucial que los usuarios comprendan la importancia de seguir buenas prácticas de seguridad al utilizar los servicios ocultos de Tor. Algunas de estas prácticas incluyen mantener actualizado el software de Tor, utilizar una versión segura del cliente de Tor, evitar descargar archivos o hacer clic en enlaces sospechosos, y utilizar conexiones encriptadas siempre que sea posible.

Además, es fundamental recordar que la tecnología de los servicios ocultos de Tor es neutra y no está diseñada específicamente para actividades ilegales. Su propósito

principal es proteger la privacidad y el anonimato en línea, y ha sido utilizado en contextos legítimos por periodistas, defensores de derechos humanos y ciudadanos preocupados por su privacidad.

Las autoridades y los proveedores de servicios de Internet también juegan un papel importante en la supervisión y la lucha contra el uso ilegal de los servicios ocultos de Tor. La cooperación entre las partes interesadas, incluidos los organismos encargados de hacer cumplir la ley y las organizaciones que promueven la seguridad y la privacidad en línea, es esencial para abordar los abusos y proteger a los usuarios legítimos de posibles amenazas.

En conclusión, los servicios ocultos de Tor son una parte integral de la red Tor, que permite a los usuarios acceder y proporcionar servicios en línea de forma anónima y segura. Aunque han surgido preocupaciones debido a su potencial uso ilegal, es importante reconocer que su propósito principal es proteger la privacidad y el anonimato en línea. Al utilizar estos servicios de manera responsable y seguir buenas prácticas de seguridad, podemos aprovechar su potencial para proteger nuestros derechos digitales y promover un entorno en línea más seguro y privado.

2.2.6. Stem

Las librerias de Stem son una colección de herramientas y recursos desarrollados en Python que proporcionan una interfaz para interactuar con la red Tor. Estas librerias están diseñadas para facilitar la implementación y el desarrollo de aplicaciones que utilizan la red Tor, permitiendo a los desarrolladores aprovechar las funcionalidades y características de Tor de una manera más sencilla.

Stem se centra en brindar un conjunto de herramientas que facilitan la programación y la automatización de tareas relacionadas con la red Tor. La libreria ofrece una API (Interfaz de Programación de Aplicaciones) completa y bien documentada que permite a los desarrolladores controlar y administrar nodos Tor, enviar solicitudes a través de la red Tor, obtener información sobre circuitos y conexiones, y realizar una amplia gama de operaciones relacionadas con la red.

Una de las características más destacadas de las librerias de Stem es su capacidad para interactuar con los servicios ocultos de Tor. Permite a los desarrolladores configurar y administrar servicios ocultos, generar direcciones .onion únicas y establecer conexiones seguras con ellos. Esto es especialmente útil para aquellos que desean crear y hospedar servicios web en la red Tor manteniendo su anonimato y privacidad.

Además de su funcionalidad básica, Stem también proporciona una serie de utilidades y herramientas auxiliares. Esto incluye funciones para analizar y manipular el protocolo de enrutamiento de Tor, realizar pruebas de circuitos y conexiones, realizar análisis de tráfico y monitorear el comportamiento de los nodos de la red Tor.

Las bibliotecas de Stem son apreciadas en la comunidad de desarrollo de Tor por su estabilidad, flexibilidad e integración con otras herramientas. Su diseño modular y personalizable satisface las necesidades específicas de los desarrolladores.

2.2.7. Aplicación en el proyecto

En mi proyecto de comunicación peer-to-peer, he aplicado de manera exitosa las tecnologías de los servicios ocultos de Tor, el enrutador Tor y la biblioteca Stem, logrando crear un entorno altamente seguro y anónimo para la comunicación entre los usuarios.

La implementación de los servicios ocultos de Tor ha sido fundamental en mi proyecto. Estos servicios permiten a los usuarios establecer conexiones y acceder a recursos de manera completamente anónima. Al utilizar direcciones .onion únicas y generadas criptográficamente, se garantiza que la identidad y la ubicación de los usuarios permanezcan ocultas, lo que es esencial en un entorno peer-to-peer donde la privacidad es primordial. Los servicios ocultos de Tor han demostrado ser una solución confiable para evitar la exposición de información personal sensible y proteger la confidencialidad de las comunicaciones.

Asimismo, la integración del enrutador Tor ha sido esencial para garantizar la privacidad y el anonimato en mi proyecto. El enrutador Tor enruta el tráfico de Internet a través de una red de nodos encriptados, lo que dificulta el seguimiento y la identificación de los usuarios. Esta tecnología crea una capa adicional de seguridad al ocultar las direcciones IP reales y mantener el anonimato en las comunicaciones. Al utilizar el enrutador Tor, he asegurado que las transmisiones de datos sean confidenciales y estén protegidas contra cualquier intento de interceptación o vigilancia no autorizada.

Además, la biblioteca Stem ha sido una herramienta invaluable en el desarrollo de mi proyecto. Con su API completa y bien documentada, he podido interactuar y controlar los nodos de Tor de manera eficiente y efectiva. La biblioteca Stem ha simplificado tareas como el establecimiento de conexiones, la gestión de circuitos y la administración de solicitudes, lo que ha mejorado la eficiencia y la seguridad de mi aplicación peer-to-peer. Gracias a Stem, he podido aprovechar al máximo las funcionalidades de la red Tor y adaptarlas a las necesidades específicas de mi proyecto.

En conjunto, la aplicación de los servicios ocultos de Tor, el enrutador Tor y la biblioteca Stem en mi proyecto de comunicación peer-to-peer ha permitido construir un entorno sólido y seguro para la interacción de los usuarios. Estas tecnologías han brindado anonimato, privacidad y confidencialidad a las comunicaciones, evitando la exposición de información personal y garantizando la protección de la identidad de los participantes. Gracias a esta implementación, los usuarios pueden comunicarse y compartir información de manera segura, confiando en que su privacidad está salvaguardada en todo momento.

La integración de servicios ocultos de Tor, el enrutador Tor y la biblioteca Stem en mi proyecto P2P ha creado un entorno de comunicación seguro, anónimo y confiable. Estas tecnologías garantizan la privacidad de los usuarios y la confidencialidad de los datos transmitidos, maximizando las funcionalidades de la red Tor y brindando una experiencia de comunicación protegida.

2.3. Git

En mi proyecto, he aprovechado la potencia y la colaboración que ofrece Git en combinación con GitHub para mantener un historial de cambios y gestionar el desarrollo de manera efectiva. Esta combinación ha sido fundamental para asegurar la calidad del software y facilitar la entrega de versiones funcionales a través de releases bien definidos.

En primer lugar, he utilizado Git como sistema de control de versiones para registrar y mantener un historial de todos los cambios realizados en el código fuente. Con Git, he podido realizar seguimiento de cada modificación, lo cual resulta extremadamente útil para rastrear y solucionar problemas, así como para revertir cambios si es necesario. Además, Git ha permitido una colaboración eficiente, ya que múltiples desarrolladores pueden trabajar simultáneamente en diferentes ramas y fusionar sus contribuciones de manera ordenada y controlada.

GitHub ha sido la plataforma elegida para alojar mi repositorio Git y aprovechar sus numerosas funcionalidades adicionales. Una de las características clave de GitHub que he utilizado es la capacidad de crear releases. He realizado tres versiones principales de mi proyecto, cada una representando un hito significativo en el desarrollo. La primera versión se enfocó en obtener una funcionalidad mínima, lo que permitió probar y validar conceptos clave. La segunda versión fue un hito importante, ya que estableció un protocolo sólido y bien definido para la comunicación peer-to-peer. Finalmente, la última versión, la versión final, incluyó una interfaz completa para realizar pruebas y testeos exhaustivos.

GitHub Actions ha sido una herramienta invaluable en mi proyecto para mantener una integración continua. Mediante la creación de flujos de trabajo automatizados con GitHub Actions, he logrado asegurar que cada cambio en el código sea probado y validado de forma automática. Por ejemplo, he utilizado Actions para ejecutar pruebas de comportamiento en cada push o pull request realizado al repositorio. Esto ha garantizado que el código funcione correctamente y cumpla con los requisitos establecidos antes de ser fusionado con la rama principal. La integración continua ha mejorado la calidad y la estabilidad del proyecto, al tiempo que ha ahorrado tiempo y esfuerzo al automatizar tareas repetitivas.

En resumen, la combinación de Git y GitHub ha sido fundamental en el desarrollo de mi proyecto. Git ha permitido mantener un historial de cambios completo y realizar un seguimiento preciso de cada modificación. GitHub, por su parte, ha proporcionado una plataforma colaborativa para alojar el repositorio y aprovechar las funcionalidades adicionales, como la creación de releases y la integración continua con GitHub Actions. Gracias a esta combinación, he podido entregar versiones funcionales de manera ordenada y garantizar la calidad del software mediante pruebas automatizadas.

2.4. Cryptodome

En mi proyecto, he utilizado la biblioteca Cryptodome como una herramienta fundamental para la implementación de criptografía robusta y segura. Cryptodome, que es una bifurcación (fork) de la popular biblioteca PyCrypto, proporciona una amplia gama de algoritmos criptográficos y funciones criptográficas para asegurar la confidencialidad, la integridad y la autenticidad de los datos en mi aplicación.

La utilización de Cryptodome me ha permitido aprovechar al máximo las capacidades criptográficas en mi proyecto. La biblioteca ofrece algoritmos avanzados como AES, RSA, SHA, HMAC, entre otros, para cifrado simétrico y asimétrico, así como funciones hash y de autenticación. Estos algoritmos y funciones criptográficas cumplen con los estándares de seguridad establecidos y proporcionan una protección sólida contra ataques maliciosos.

Una de las características destacadas de Cryptodome es su enfoque en la implementación segura de algoritmos criptográficos. La biblioteca ha sido diseñada teniendo en cuenta las mejores prácticas y directrices de seguridad, y ha sido sometida a revisiones de código exhaustivas para garantizar su integridad y fiabilidad. Al utilizar Cryptodome, puedo tener la confianza de que las operaciones criptográficas realizadas en mi proyecto están protegidas contra posibles vulnerabilidades y debilidades en la implementación.

Además, Cryptodome ofrece una interfaz de programación amigable y fácil de usar, lo que facilita la integración de las funciones criptográficas en mi aplicación. La biblioteca proporciona métodos claros y documentados para realizar operaciones criptográficas, como el cifrado y el descifrado de datos, la generación y verificación de firmas digitales, y la creación y verificación de claves. Esto me ha permitido implementar de manera eficiente y efectiva los algoritmos criptográficos necesarios en mi proyecto.

En resumen, la utilización de Cryptodome en mi proyecto ha sido fundamental para garantizar la seguridad y la confidencialidad de los datos. Gracias a su amplia gama de algoritmos y funciones criptográficas, puedo implementar mecanismos de cifrado y autenticación sólidos en mi aplicación. La atención dedicada a la seguridad en la implementación de Cryptodome me brinda la confianza necesaria en la robustez de las operaciones criptográficas realizadas. En conjunto, Cryptodome ha demostrado ser una biblioteca confiable y poderosa para la criptografía en mi proyecto, permitiéndome proteger los datos sensibles y salvaguardar la integridad de la información en todo momento.

2.5. PySocks

En mi proyecto, he utilizado la biblioteca PySocks como una herramienta clave para establecer conexiones a través de proxies SOCKS en mi aplicación. PySocks es una biblioteca de Python que facilita la comunicación a través de este protocolo, permitiendo una capa adicional de anonimato y seguridad en las conexiones de red.

La implementación de Socks en mi proyecto ha sido de gran utilidad para garantizar conexiones seguras y protegidas. La biblioteca proporciona una interfaz sencilla y fácil de usar para establecer conexiones a través de proxies SOCKS, permitiéndome ocultar la dirección IP real y enmascarar la ubicación de los usuarios. Esto es especialmente relevante en entornos donde se requiere anonimato, como en aplicaciones peer-to-peer o al acceder a servicios remotos de forma segura.

Al utilizar PySocks, he podido aprovechar las funcionalidades de los proxies SOCKS para enrutar el tráfico de red a través de intermediarios. Esto brinda una capa adicional de seguridad al ocultar la dirección IP y proteger la identidad del usuario. Además, PySocks permite trabajar con diferentes versiones del protocolo SOCKS, lo que brinda flexibilidad y compatibilidad con una variedad de escenarios de red.

La biblioteca PySocks también ofrece características avanzadas, como la autenticación de proxies y la capacidad de especificar diferentes tipos de proxies SOCKS (SOCKS4, SOCKS4a, SOCKS5). Estas funcionalidades adicionales me han permitido adaptar y personalizar las conexiones de red según los requisitos específicos de mi proyecto, brindando una mayor flexibilidad y control sobre las comunicaciones.

En resumen, la incorporación de la biblioteca PySocks en mi proyecto ha sido de gran ayuda para establecer conexiones seguras y anónimas a través de proxies SOCKS. La capacidad de ocultar la dirección IP y enrutar el tráfico a través de intermediarios ha mejorado significativamente la privacidad y la seguridad en las comunicaciones. Con la simplicidad y la potencia de PySocks, he podido integrar fácilmente esta funcionalidad en mi aplicación y asegurar que las conexiones de red sean confiables y protegidas.

2.6. Diseño y arquitectura

En esta sección, se describe en detalle el diseño y la arquitectura del protocolo P2P con cifrado end-to-end utilizando la red Tor en el proyecto DeProtocol. Se abordan los diferentes aspectos de la arquitectura, desde la descripción general hasta las decisiones de diseño clave y la gestión de conexiones.

2.6.1. Descripción general de la arquitectura

La arquitectura se basa en un enfoque descentralizado en el que los nodos participantes actúan tanto como clientes como servidores. Cada nodo tiene la capacidad de establecer conexiones P2P con otros nodos y transmitir datos de forma directa. La arquitectura sigue una estructura de red overlay, donde los nodos se conectan directamente entre sí formando una red virtual superpuesta a la infraestructura de red subyacente.

2.6.2. Decisiones de diseño

Para implementar el protocolo P2P en DeProtocol, se ha seleccionado Python como lenguaje de programación principal. Esta elección se basa en las ventajas que ofrece Python, como su amplia variedad de bibliotecas criptográficas y de red, su facilidad de uso y su comunidad activa de desarrolladores. En particular, se utilizan las siguientes librerías:

- **Stem**: En el proyecto DeProtocol, la librería Stem ha sido utilizada para actuar como un proxy en cada nodo, permitiendo la apertura de servicios ocultos mediante la asignación de una dirección .onion. Esto ha posibilitado la conexión anónima y segura de otros pares a través de la red Tor. Además, Stem ha facilitado la creación y control de circuitos encriptados, estableciendo rutas seguras entre nodos y garantizando la confidencialidad de las comunicaciones en el protocolo P2P.
- **PyCryptodome**: En el proyecto DeProtocol, la librería PyCryptodome ha sido utilizada para implementar el cifrado y descifrado de las comunicaciones, así como para firmar y verificar los mensajes transmitidos entre los nodos. Se han utilizado algoritmos criptográficos confiables, como AES, RSA y elíptica-curva, proporcionados por PyCryptodome. Esto ha permitido garantizar la seguridad y confidencialidad de las comunicaciones en el protocolo P2P. Mediante la generación de claves públicas y privadas, PyCryptodome ha facilitado el cifrado y descifrado de datos en bloques, utilizando el modo PKCS1_OAEP, asegurando así la integridad y autenticidad de los datos transmitidos. En resumen, PyCryptodome ha sido fundamental en el proyecto DeProtocol al proporcionar las funcionalidades de encriptación, descifrado, firma y verificación de mensajes necesarias para garantizar la seguridad de las comunicaciones entre los nodos.

- **Behave:** Behave es una librería de pruebas de aceptación escrita en Python que se ha utilizado para realizar pruebas de extremo a extremo en el protocolo P2P con cifrado end-to-end utilizando la red Tor. Con Behave, se han creado escenarios y pasos de prueba en un lenguaje legible por humanos llamado Gherkin. Estas pruebas han permitido verificar la funcionalidad principal del proyecto, como la correcta inicialización y comunicación efectiva entre dos nodos. La integración de Behave con la librería DeProtocol ha facilitado la escritura y ejecución de pruebas de aceptación, asegurando que la funcionalidad central de la librería se mantenga sin errores a medida que se realicen cambios o adiciones en el código.

- **Pytest:** Pytest es un framework de pruebas en Python que se ha utilizado para realizar pruebas unitarias en el proyecto DeProtocol. Se han diseñado pruebas unitarias utilizando Pytest para las partes críticas y menos moldeables de la librería, como los paquetes y clases fundamentales del protocolo P2P. Estas pruebas unitarias han verificado el correcto funcionamiento de estas partes clave y han evitado la introducción de errores al realizar modificaciones o cambios en el código. La utilización de Pytest ha simplificado la escritura y ejecución de pruebas unitarias, asegurando la calidad y robustez del proyecto.

- **PyQt5:** A pesar de no haberse considerado inicialmente, se ha decidido agregar una demostración adicional mediante la implementación de un front-end básico utilizando PyQt5. Si bien se utilizará PyQt5 para esta tarea, no se profundizará en los detalles del desarrollo del front-end en sí.

2.6.3. Diseño del Protocolo P2P en TOR

El diseño del protocolo P2P se basa en un enfoque convencional donde los clientes asumen los roles de cliente y servidor. Cada cliente abre un socket en modo de escucha en un puerto específico para esperar conexiones entrantes. El objetivo principal del protocolo es garantizar la seguridad, privacidad y anonimato de las comunicaciones entre los nodos. Para lograrlo, el protocolo utiliza la red Tor, que se compone de una serie de relays.

El proceso del servidor para crear el servicio oculto (Figura 4) empieza por la creación de un socket. Este esta conectado a través de un proxy a la red de Tor. Una vez tenga conexión con esta se va a realizar una petición a diferentes nodos de la red, para que estos sean su punto de introducción, una vez estos acepten se enviará la información junto a la clave publica previamente generada por el protocolo Tor hacia un descriptor de la red el cual hará una función parecida a la de un servidor DNS.

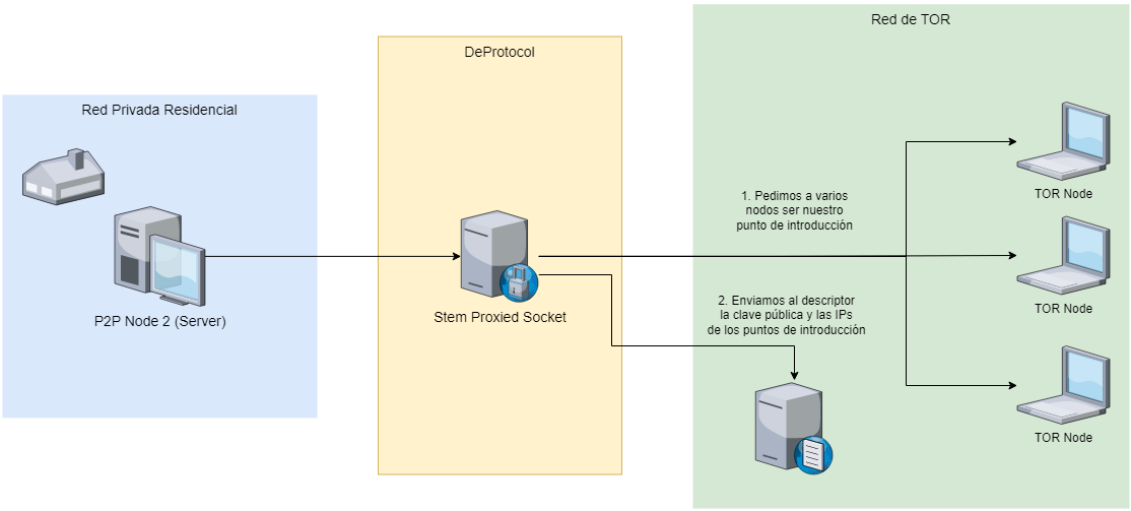

Figura 4: Creación del Servicio Oculto TOR en DeProtocol.

El proceso de establecimiento de conexión comienza con cada cliente estableciendo un circuito a través de la red Tor. El circuito consta de varios relays que se encargan de enrutar el tráfico de forma anónima. Cada mensaje enviado entre los nodos se cifra en varias capas, una para cada relay del circuito. Esto garantiza que los relays solo puedan descifrar la capa correspondiente a su nodo, y la información se mantiene segura y confidencial durante el tránsito.

Una vez que el circuito está establecido, el cliente abre un hidden service utilizando la dirección onion generada por Tor. El hidden service actúa como un punto de entrada para las conexiones entrantes. La dirección onion es única y se genera de manera criptográficamente segura, lo que garantiza que solo los nodos autorizados puedan establecer una conexión.

Cuando un segundo nodo desea establecer una conexión con el primer nodo, utiliza la dirección onion para dirigir su tráfico hacia el hidden service. El tráfico se enruta a través de la red Tor utilizando los relays, cifrando cada capa del mensaje para

mantener la confidencialidad. Cada relay solo conoce el nodo anterior y el siguiente en el circuito, lo que hace que sea extremadamente difícil rastrear el origen y destino de la comunicación.

Además de la encriptación en cada etapa, el protocolo P2P se beneficia de la arquitectura descentralizada de la red Tor. La información se transmite a través de múltiples relays en diferentes ubicaciones geográficas, lo que dificulta aún más el seguimiento y la identificación de los nodos involucrados en la comunicación. Esto proporciona un alto grado de anonimato y protección de la identidad de los usuarios.

En cuanto a la dirección IP, el protocolo P2P oculta la dirección real de los nodos al utilizar la red Tor. En lugar de revelar la dirección IP del cliente, la comunicación se realiza a través de los relays de la red Tor, lo que oculta la ubicación y cualquier rastro directo hacia los nodos. Esto brinda una capa adicional de seguridad y protección de la privacidad.

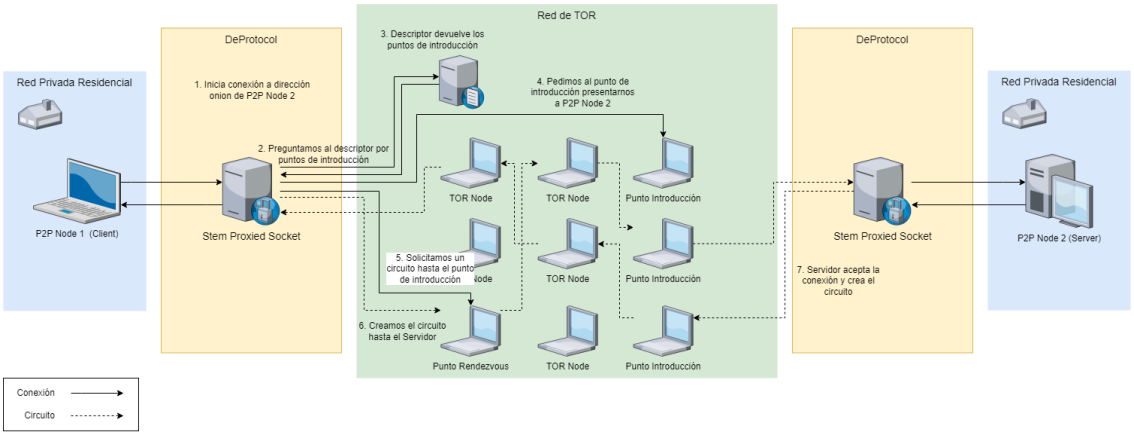

Figura 5: Flujo de conexión Cliente-Servidor en DeProtocol.

En resumen, el diseño peer-to-peer (P2P) que utilizamos en nuestra plataforma presenta numerosos beneficios al ofrecer una conexión totalmente descentralizada. Una de las ventajas más destacadas es la protección de la privacidad al ocultar la dirección IP real de los nodos mediante el uso de la red Tor. En lugar de revelar directamente la dirección IP del cliente, la comunicación se realiza a través de los relays de la red Tor, lo que garantiza la ocultación de la ubicación y cualquier rastro directo hacia los nodos.

Este enfoque brinda una capa adicional de seguridad al dificultar la identificación y el rastreo de los nodos involucrados en la conexión. Al utilizar el protocolo P2P y la red Tor, los usuarios pueden disfrutar de una conexión segura y anónima, lo que protege su privacidad y los resguarda de la vigilancia no autorizada.

2.6.4. Diseño de la Seguridad y Encriptación de las Comunicaciones

Pese a la gran seguridad y encriptación de Tor, adicionalmente se ha diseñado e implementado un sistema de seguridad robusto utilizando métodos de encriptación, firma y verificación. Añadiendo así una última capa de cifrado end-to-end entre los clientes del protocolo. A continuación, se detallan los componentes y métodos utilizados para garantizar la seguridad de los datos:

- **Generación de Claves:** Para asegurar el intercambio de datos cifrados, se utiliza el algoritmo RSA (Rivest-Shamir-Adleman), el cual genera un par de claves pública y privada con una longitud de 2048 bits. Es importante destacar que este valor puede ser incrementado para aumentar aún más la seguridad del protocolo, aunque esto conlleva un incremento en el costo computacional. Sin embargo, dado que esta capa de cifrado se considera la última barrera de protección, se ha decidido mantener el valor de 2048 bits como un equilibrio entre seguridad y eficiencia. La clave privada se mantiene en estricto secreto, mientras que la clave pública se comparte con las partes involucradas en el intercambio de datos, permitiendo así el cifrado y descifrado adecuado de la información.
- **Firma y Verificación:** Para garantizar la integridad y autenticidad de los datos, se utiliza una función de firma digital basada en el algoritmo PKCS1_v1_5 y el hash SHA256. La función, tomaremos el mensaje como entrada y este será firmado utilizando la clave privada correspondiente. El hash SHA256 se calcula a partir del mensaje para obtener una representación única del contenido. Por otro lado, el cliente receptor se encarga de verificar la firma digital utilizando la clave pública asociada. Calcula el hash SHA256 del mensaje recibido y compara este valor con la firma proporcionada. Si coinciden, se garantiza la integridad y autenticidad del mensaje.
- **Serialización de Claves:** Para facilitar el intercambio y almacenamiento de claves, dispondremos de dos funciones, una estará encargada de cargar una clave en formato base64 y convertirla en un objeto de clave válido. La segunda serilizará una clave en un formato base64 para su posterior almacenamiento y transmisión.

2.6.5. Diseño del protocolo de Comunicación entre Nodos y paquetes

Para facilitar la comunicación entre los nodos, hemos desarrollado un protocolo basado en el esquema UDP (User Datagram Protocol). Dado que la red de Tor, si bien ofrece ventajas en términos de privacidad y anonimato en línea, presenta limitaciones en cuanto a velocidad debido a diversos factores, como el enrutamiento entre múltiples nodos y las restricciones de ancho de banda. Por lo tanto, hemos optado por una transmisión de datos prácticamente asincrónica.

Nuestro protocolo se compone de diferentes tipos de paquetes que permiten el intercambio de datos entre los nodos. Estos paquetes están diseñados de manera modular y flexible, centrándose en el contenido del payload, que se encuentra en formato JSON. Los diferentes tipos de paquetes son los siguientes:

- **Handshake:** El paquete de saludo o "handshake.es esencial para iniciar una conexión entre dos clientes. En este paquete se transmiten datos fundamentales para establecer una comunicación segura. Incluimos la dirección Onion generada por nuestro servicio oculto en la red de Tor, así como el nombre de usuario que nos identifica. También compartimos una imagen de perfil, clave pública y un indicador de inicio de comunicación. Este paquete no se cifra y, al recibirlo, el cliente crea automáticamente una nueva conexión y responde con un paquete similar compartiendo su información.

- **Message:** El propósito de este paquete es transmitir información sin esperar una respuesta inmediata. Contendrá la hora exacta en la que se generó el mensaje, así como su contenido. Para asegurar la privacidad y la integridad del mensaje, este se cifrará y firmará. De esta manera, garantizamos que el mensaje permanezca confidencial y que cualquier modificación mínima resulte en una falla de la firma, asegurando así su inmutabilidad.

- **Keepalive:** Con el objetivo de mantener activo el socket de conexión, hemos desarrollado un protocolo adicional basado en un paquete especial. Este paquete no se cifrará y no contendrá ningún contenido en su carga útil. Su única función será realizar un "ping.a intervalos de tiempo predefinidos para evitar que el tiempo de espera de conexión entre los clientes se agote. De esta manera, garantizamos que la comunicación se mantenga estable y que la conexión no se interrumpa debido a inactividad.

La estructura básica de cada paquete consta de una cabecera de 8 bytes de datos (Figura 6). Esta cabecera incluye campos para la versión del protocolo, el tipo de paquete, el número de secuencia y el tamaño del payload contenido. El cuerpo del paquete, es decir, el payload en formato JSON, tiene un tamaño dinámico y puede variar según la información transmitida.

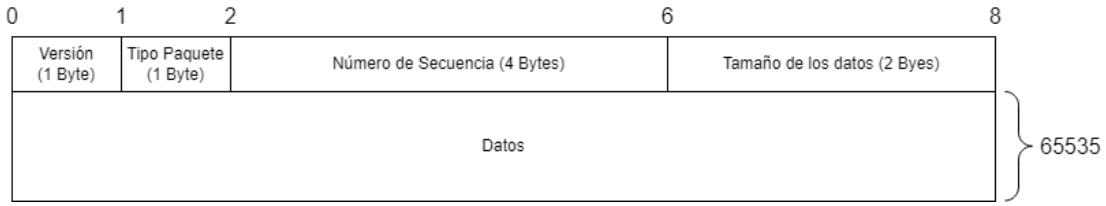

Figura 6: Estructura de un paquete de DeProtocol.

Nuestros paquetes se estructuran de la siguiente forma:

- **Versión:** El primer byte se reserva para identificar la versión del protocolo DeProtocol que se está utilizando. Esta característica es fundamental para garantizar la compatibilidad entre diferentes implementaciones del protocolo. Al utilizar un byte para representar la versión, se pueden representar hasta 255 versiones distintas de DeProtocol. Esto proporciona una amplia flexibilidad para adaptar y evolucionar el protocolo a lo largo del tiempo.
- **Tipo:** El segundo byte se utiliza para definir el tipo de paquete en el contexto del protocolo DeProtocol. Al disponer de un byte completo, se pueden definir hasta 255 tipos diferentes de paquetes. Esta funcionalidad permite clasificar y distinguir diversos tipos de datos transmitidos, como mensajes, comandos, respuestas o notificaciones, según las necesidades específicas del sistema que implementa el protocolo.
- **Número de Secuencia:** El número de secuencia se emplea para establecer y rastrear el orden en el que se envían los paquetes dentro del protocolo DeProtocol. Con 4 bytes (32 bits) asignados a esta finalidad, se pueden representar hasta 4,294,967,295 números de secuencia únicos. Esta capacidad masiva de numeración proporciona una gran escalabilidad y robustez al protocolo, permitiendo un seguimiento preciso y confiable del flujo de los paquetes en aplicaciones que requieren una secuencia precisa de eventos.
- **Tamaño de los datos:** El séptimo y octavo byte se reservan para indicar el tamaño del payload o datos que se transmiten en el paquete DeProtocol. Con 2 bytes (16 bits) destinados a este propósito, se puede representar un rango de tamaños desde 0 hasta 65,535 bytes. Esta capacidad de almacenamiento proporciona suficiente flexibilidad para enviar y recibir datos de diversos tamaños dentro del límite máximo establecido por el protocolo.

El objetivo principal de nuestro protocolo es garantizar una comunicación eficiente y confiable entre los nodos, superando las limitaciones de velocidad y ancho de banda de la red de Tor. Al utilizar un enfoque desincronizado y modular, buscamos minimizar la latencia y maximizar la flexibilidad para adaptarnos a diferentes necesidades de comunicación. Esto permitirá a los usuarios aprovechar al máximo las capacidades de nuestra biblioteca y personalizarla según sus requerimientos individuales.

A través de un protocolo basado en UDP, una estructura modular de paquetes y la utilización de payloads en formato JSON, proporcionamos una solución robusta y adaptable para la comunicación entre nodos en la red de Tor. Nuestra arquitectura garantiza una transmisión de datos eficiente y segura, preservando la privacidad y el anonimato en línea de los usuarios involucrados en la comunicación.

2.6.6. Diseño general de DeProtocol

DeProtocol incorpora diversas secciones para administrar al cliente a través de comandos, inicializar los binarios necesarios y registrar eventos y comandos. Si bien hemos profundizado en las secciones más relevantes, como la conexión, encriptación y comunicación, no entraremos en detalles sobre el diseño de las demás secciones, ya que no resulta relevante para comprender los objetivos técnicos de nuestro proyecto. No obstante, se ha desarrollado un diagrama de ejecución en la Figura 7 que ofrece una visión general del flujo que sigue nuestra aplicación desde su inicio hasta su final.

En dicho diagrama, se puede observar el estado infinito de espera de nuevos paquetes, donde paralelamente podemos utilizar la consola para enviar paquetes nosotros mismos, o más específicamente, en nuestra implementación, mensajes.

El diagrama de ejecución nos proporciona una comprensión superficial de cómo se desarrolla el funcionamiento de nuestra aplicación, centrándose en los aspectos relevantes para nuestra implementación y los objetivos técnicos que perseguimos.

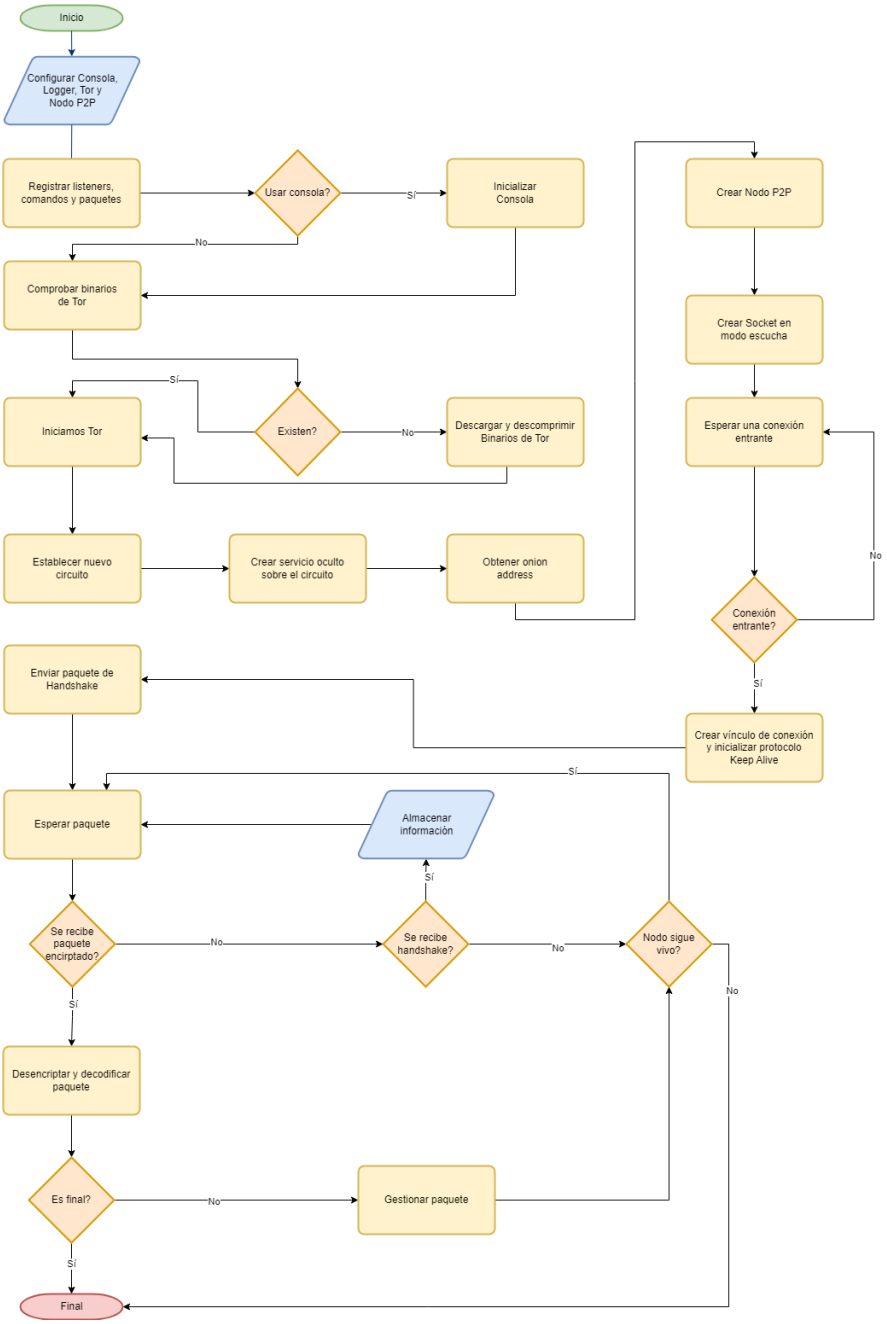

Figura 7: Flujo de ejecución de DeProtocol.

3. Implementació, código y ejecución

En el análisis detallado que sigue, examinaremos exhaustivamente las partes clave del proyecto, proporcionando una visión profunda de su implementación y funcionamiento. Estas secciones son fundamentales para comprender en profundidad cómo se han abordado los desafíos específicos y cómo se ha logrado la funcionalidad requerida. Sin embargo, es importante mencionar que, debido a la extensión del proyecto y el enfoque en las partes más relevantes, algunas secciones pueden ser tratadas de manera más superficial o incluso no ser abordadas en este contexto. Esto no implica que su importancia sea menospreciada, sino que nos enfocaremos en los aspectos que requieren mayor atención y explicación detallada.

3.1. Estructura del proyecto

En neustro proyecto contenemos la implementación de tres proyectos distintos:

- **DeProtocol** El primero de ellos es la librería "DeProtocol", la cual engloba todas las funcionalidades y reglas indispensables para lograr una comunicación eficiente entre diferentes sistemas.
- **Proyecto de pruebas:** El segundo proyecto consiste en un conjunto de pruebas que hemos desarrollado para verificar la efectividad y la robustez del protocolo implementado. Estas pruebas nos permiten comprobar que el protocolo cumple con los requisitos establecidos y que es capaz de manejar diferentes escenarios y casos de uso.
- **Aplicación de ejemplo:** El último proyecto añadido recientemente se llama 'sample_gui_app' y se trata de una aplicación de interfaz gráfica de usuario de ejemplo. Aunque no profundizaremos en este proyecto en particular en este trabajo, su inclusión demuestra la versatilidad y la aplicabilidad del protocolo implementado.

La estructura del proyecto "DeProtocol"se organiza de la siguiente manera:

- **api:** Contiene los archivos relacionados con la API del proyecto. Es el punto de entrada para el desarrollo de terceras aplicaciones a DeProtocol.
- **app:** Contiene la lógica principal de la aplicación
  - **console:** Contiene los archivos relacionados con la interfaz de consola.
    - **command:** Contiene los comandos disponibles para utilizar a través de la consola.
  - **listeners:** Contiene los archivos de escucha de eventos.
  - **setup:** Contiene los archivos de configuración de la aplicación.
- **event:** Contiene los archivos relacionados con el sistema de eventos.
  - **events:** Contiene las clases de eventos específicos.
- **network:** Contiene los archivos relacionados con la gestión de la red.
  - **peer_networking:** Contiene los archivos relacionados con el protocolo peer to peer.
  - **protocol:** Contiene los archivos relacionados con el protocolo de comunicación.
    - **packets:** Contiene los archivos de paquetes que utiliza el protocolo para comunicarse.
    - **payloads:** Contiene el formato de los payloads que contendrá cada paquete en funcion del tipo.
- **utils:** Contiene los archivos de utilidad para el proyecto.

La estructura del proyecto de "test"se estructura de la siguiente forma:

- **deprotocol:** Contiene los archivos relacionados con las pruebas unitarias del proyecto "DeProtocol".
  - **api:** Contiene los archivos relacionados con las pruebas de la API que se utilizará de forma externa, y nuestro objetivo principal.
  - **network:** Contiene los archivos relacionados con las pruebas de red.
    - **protocol:** Contiene los archivos relacionados con las pruebas del protocolo específico, importante para que garantizar el funcionamiento de las comunicaciones.
- **features:** Contiene el apartado de pruebas de aceptación, que son las pruebas que garantizan la funcionalidad especificada.
  - **steps:** Contiene los archivos necesarios para los diferentes pasos que hay en las pruebas de aceptación.

El proyecto 'sample_gui_app' consiste de una carpeta que contiene los diseños de las interfaces implementadas y un archivo "main"donde se contiene todo el código que la construye y hace funcionar.

3.2. Implementación de Deprotocol y la API

Con el propósito de facilitar la interacción con nuestra API, hemos desarrollado un archivo denominado 'client.py', el cual alberga la clase Client. Esta clase sirve como el punto de entrada principal para los usuarios externos que deseen utilizar nuestra biblioteca dentro del proyecto DeProtocol.

A continuación, nos adentraremos en una explicación detallada de cada uno de los métodos que conforman esta clase, analizando tanto su implementación como su utilidad en el contexto de nuestro proyecto.

3.2.1. Inicialización del Cliente

Primeramente encontramos en el método inicializador de la clase 'Client' (Figura 8) una definición donde creamos un objeto DeProtocol, a partir de ahora la base de toda la lógica de la aplicación, y el controlador base de cualquier operación.

Figura 8: Métodos inicializadores del cliente.

A su vez encontramos el método 'start', este se encargara de hacer una llamada al controlador (clase DeProtocol) para inicializar cada uno de los componentes que forman la lógica de nuestro proyecto como se observa en la figura 9.

En el método 'on_start' perteneciente al controlador, podemos observar que recogemos dos parametros, la dirección a utilizador como servidor proxy, y el puerto. Después de esto hay tres llamadas para registrar diferentes componentes de nuestro protocolo, primero esta el registro de los eventos, el registro de los comandos y por último de los paquetes.

```
werogg +1
def on_start(self, proxy_host='127.0.0.1', proxy_port=9050):
    self.register_default_events()
    self.register_default_commands()
    self.register_default_packets()

    self.setups = {
        'console': ConsoleSetup(self),
        'logger': LoggerSetup(),
        'tor': TorSetup(self, proxy_host, proxy_port),
        'peer_networking': P2PNodeSetup(self, NODE_HOST, NODE_PORT)
    }

    for setup in self.setups.values():
        setup.setup()

    self.node.onion_address = self.setups['tor'].tor_service.get_address()

    Logger.get_logger().info(f"Starting {APP_NAME} version {APP_VERSION}, running on {platform.system()}")

    event = DeProtocolReadyEvent()
    self.listeners.fire(event)
```

Figura 9: Inicializador del controlador.

Las llamadas realizadas son esenciales para cargar en memoria y hacer disponibles los diversos eventos, comandos o paquetes de nuestro protocolo.

A continuación, encontramos los "setups", que constituyen una de las fases más importantes en el flujo de nuestra aplicación. Estos se basan en una clase genérica llamada 'SetupABC', que contiene un método abstracto llamado 'setup'. Esta estructura nos permite crear nuevos objetos que heredan de la clase genérica, lo que les permite heredar su método y ejecutar fácilmente diferentes tareas de inicialización. Estas tareas incluyen el lanzamiento de instancias de los siguientes objetos: 'Node', que es la base del protocolo peer-to-peer; 'ConsoleUI', que es la interfaz de consola opcional actualmente disponible en nuestra aplicación; el objeto 'logger', que recopila información sobre la ejecución; y finalmente, 'TorService' y 'TorUtils', que nos permiten descargar y ejecutar las herramientas y binarios necesarios para establecer conexiones a la red de Tor. En el diagrama de clases mostrado en la Figura 10, se ilustran las herencias mediante flechas blancas y las instancias de cada objeto se representan con rombos negros.

Parte de estas instancias heredan directamente del sistema de multihilo, por este motivo es necesario estos 'iniciadores' que marcan el lanzamiento del hilo y servicio.

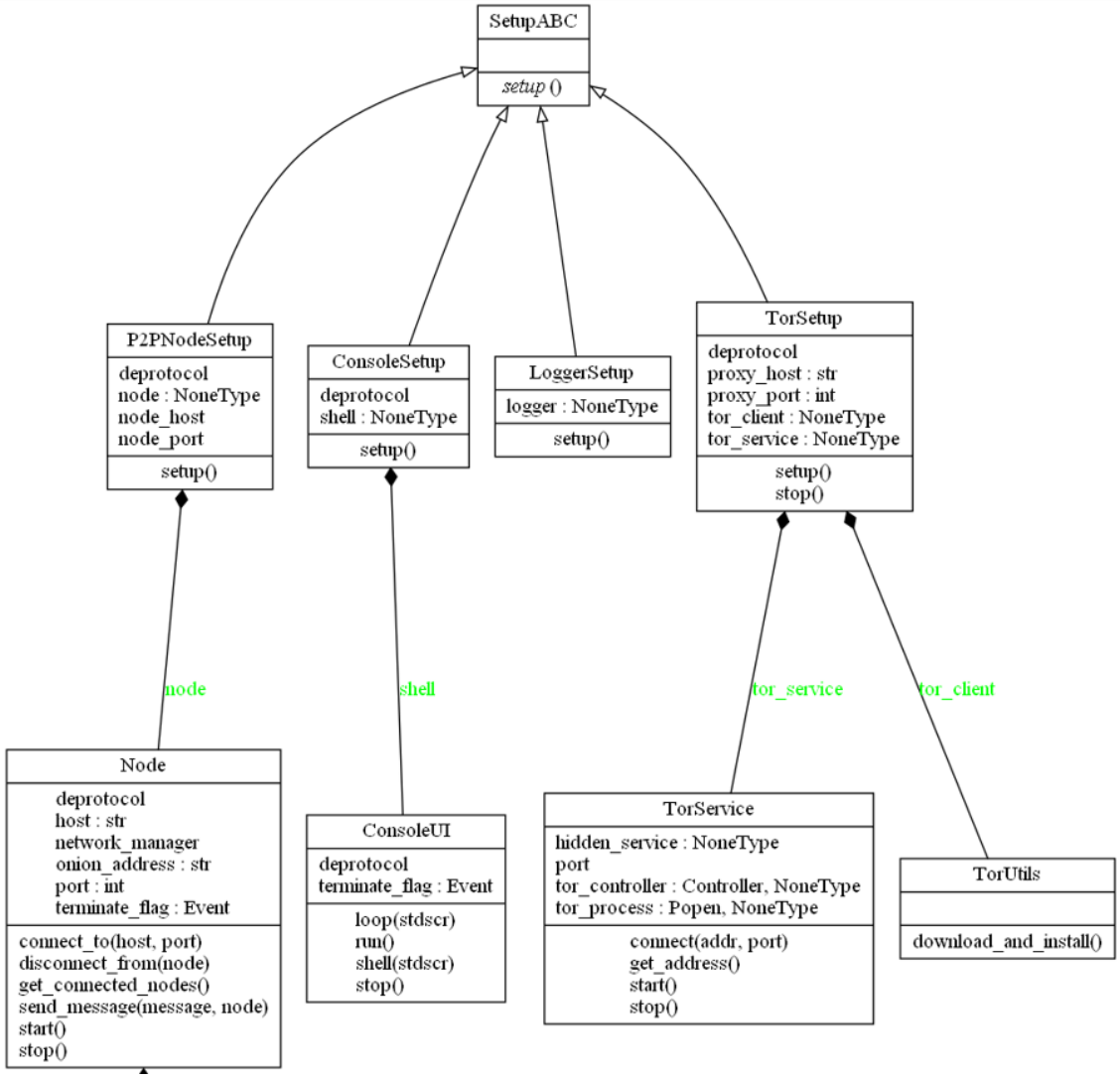

Figura 10: Diagrama de clases del proceso de inicialización.

En la definición de la clase 'Node', encontramos varias propiedades necesarias para establecer conexiones peer-to-peer. Estas propiedades incluyen la dirección 'host', la dirección 'onion_address', el puerto y el indicador de terminación, que determina cuándo se debe detener una instancia. Además, la clase 'Node' contiene dos instancias de objetos: 'deprotocol', que se refiere al controlador, y 'network_manager', que actúa como una capa intermedia para la gestión de nuestra red. A través de la clase 'Node', podemos realizar acciones como 'connect_to' para establecer una conexión, 'disconnect_from' para desconectarnos, 'get_connected_nodes' para obtener una lista de conexiones activas o inactivas, 'send_message' para enviar un mensaje a un nodo específico, y finalmente, dos métodos para iniciar y detener el servicio. Es importante destacar que cada función delega la lógica a la instancia del objeto responsable de esa tarea.

En el esquema de la Figura 11, se muestra la composición de la gestión de conexiones en nuestro protocolo peer-to-peer. La clase 'Node' instancia un 'NetworkManager', que como su nombre indica, se encarga de gestionar las conexiones de red. Realiza comprobaciones de validez y crea conexiones a través del 'ConnectionHandler', que actúa como intermediario y solo se encarga de crear y cerrar conexiones. Estas conexiones están representadas por la clase 'NodeConnection', la cual será explicada en la sección correspondiente.

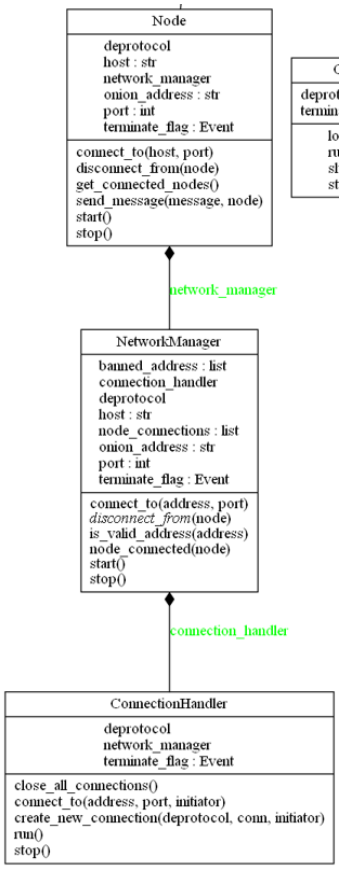

Figura 11: Diagrama de clases del protocolo peer-to-peer.

En nuestro sistema, nos encontramos con la clase 'ConsoleUI', que es instanciada por 'ConsoleSetup'. Esta clase tiene la responsabilidad de gestionar la interfaz de usuario en la consola. En esencia, 'ConsoleSetup' inicia un hilo que se ejecuta en un ciclo infinito. Durante este ciclo, se redirige la salida de los registros a la consola y se esperan comandos de entrada para controlar la aplicación. En la próxima sección, exploraremos a fondo el Sistema de consola y comandos.

En contraste, el 'logger' se inicia mediante 'LoggerSetup'. No se trata de un servicio que requiere una ejecución constante. En su lugar, se basa en el patrón de diseño 'singleton', lo que significa que se crea una única instancia del objeto. Esta instancia del logger incluye varias configuraciones que nos permiten mantener un registro ordenado de todo lo sucedido dentro de la aplicación.

Por último, pero no menos importante, tenemos la clase 'TorSetup'. En la próxima sección profundizaremos en esta clase. Su propósito es facilitar la descarga de los binarios desde una dirección especificada en la configuración. Además, permite iniciar el servicio de Tor directamente con la configuración adecuada para ser utilizado en nuestra aplicación.

3.2.2. Inicialización de Tor

En esta sección de nuestro trabajo, nos centramos en analizar uno de los componentes del diagrama representado en la Figura 10. Observamos que la clase 'TorSetup' es responsable de crear instancias de 'TorService' y 'TorUtils'. Nos interesa particularmente el método 'setup' que se muestra en la Figura 12, ya que contiene dos acciones adicionales además de la creación de la instancia del servicio.

```python
def setup(self):
    # Configure socks to use Tor proxy by default
    socks.setdefaultproxy(PROXY_TYPE, self.proxy_host, self.proxy_port)
    Logger.get_logger().debug(
        f'Default proxy configuration set: {PROXY_TYPE} - {self.proxy_host}:{self.proxy_port}')

    # Download and install Tor Client
    self.tor_client = TorUtils()
    self.tor_client.download_and_install()

    # Start Tor Service
    self.tor_service = TorService(9051)
    self.tor_service.start()
    Logger.get_logger().info("Tor Service started correctly!")
```

Figura 12: Método 'setup' de la clase 'TorSetup'.

En la primera línea de código del método 'setup', se realiza una llamada a 'socks.setdefaultproxy' con varios parámetros que se obtienen directamente de la configuración. Esta llamada tiene como propósito establecer un servidor proxy por defecto. En términos básicos, un servidor proxy actúa como un intermediario entre el cliente y la conexión a la que se está accediendo. Al utilizar este método, estamos redirigiendo todas las conexiones que pasan a través de nuestra aplicación para que se enrutan a través de este servidor proxy.

El objetivo de establecer un servidor proxy por defecto es controlar y dirigir el tráfico de red de manera centralizada. Al hacer que todas las conexiones pasen a través de este servidor proxy, podemos aplicar medidas de seguridad adicionales, filtrar contenido no deseado o monitorear el tráfico para realizar análisis o auditorías. Esta configuración nos permite tener un mayor control sobre las conexiones salientes desde nuestra aplicación y mejorar la seguridad y privacidad de los usuarios.

Dentro de nuestra implementación, tenemos la clase 'TorUtils' que desempeña un papel importante. En esta sección, creamos una instancia de esta clase y realizamos una llamada al método 'download_and_install'. Esta acción tiene como objetivo gestionar los binarios de Tor de manera externa.

Cuando ejecutamos nuestro proyecto, verificamos la existencia de los binarios de Tor en la carpeta del proyecto. En caso de que los binarios estén presentes, los descomprimimos y los dejamos listos para su ejecución. Sin embargo, si los binarios no están disponibles localmente, recurrimos a la dirección especificada en el archivo 'settings.py' para descargarlos desde allí. La Figura 13 muestra cómo se realiza este proceso.

Al realizar esta verificación y gestión de los binarios de Tor, aseguramos que nuestro proyecto pueda funcionar de manera independiente, sin depender de la presencia de los binarios en el entorno de desarrollo. De esta manera, garantizamos la portabilidad y flexibilidad de nuestra aplicación al permitir la descarga automática de los binarios desde una fuente externa cuando sea necesario.

```python
class TorUtils:
    werogg +1
    @staticmethod
    def download_and_install():
        os.makedirs(DATA_DIR, exist_ok=True)
        os.makedirs(BIN_DIR, exist_ok=True)

        if os.path.isfile(TOR_BINARIES_DIR):
            os.remove(TOR_BINARIES_DIR)
            Logger.get_logger().warning(
                'A tor installation was found in your system, if DeProtocol is not working please delete tor.tar.gz'
            )

        response = requests.get(TOR_BINARIES_URL)
        Logger.get_logger().trace(f'requests_get: Requesting tor binaries from [{TOR_BINARIES_URL}]')
        Logger.get_logger().info("Tor Client binaries downloaded")
        with open(TOR_BINARIES_DIR, 'wb') as f:
            f.write(response.content)

        try:
            with tarfile.open(TOR_BINARIES_DIR, 'r:gz') as tar:
                members = tar.getmembers()
                for member in tqdm(members):
                    tar.extract(member, path=BIN_DIR)
            Logger.get_logger().info(f'Tor Client binaries were successfully decompressed on {BIN_DIR}')
        except Exception as exc:
            Logger.get_logger().error(exc)
```

Figura 13: Método 'download_and_install' de la clase 'TorUtils'.

3.2.3. Creación del servicio oculto

Una vez hemos preparado los binarios de Tor, tal como se explicó en la sección anterior de este trabajo, procedemos a llevar a cabo el levantamiento del servicio. Esta tarea es llevada a cabo por el método 'start' de la clase 'TorService' (Figura 14, el cual se encuentra dentro del archivo 'tor_network.py'.

```
def start(self):
    try:
        self.tor_process = stem.process.launch_tor_with_config(
            config={
                'SocksPort': '9050',
                'SocksPolicy': 'accept *',
                'ControlPort': str(self.port),
                'DataDirectory': TOR_DATA_DIR,
                'HiddenServiceDir': HIDDEN_SERVICE_DIR,
                'HiddenServicePort': f'{HIDDEN_SERVICE_VIRTUAL_PORT} {HIDDEN_SERVICE_HOST}:{HIDDEN_SERVICE_FORWARD_PORT}'
            },
            tor_cmd=TOR_BINARIES_PATH,
            init_msg_handler=self._print_bootstrap_lines,
            take_ownership=True
        )
    except Exception as e:
        Logger.get_logger().error(e)

    self.tor_controller = stem.control.Controller.from_port(port=self.port)
    self.tor_controller.authenticate()
    self.tor_controller.new_circuit()

    bytes_read = self.tor_controller.get_info("traffic/read")
    bytes_written = self.tor_controller.get_info("traffic/written")

    Logger.get_logger().trace(f'tor_traffic: Tor relay has read {bytes_read} bytes and written {bytes_written}.')

    self.hidden_service = self.tor_controller.create_ephemeral_hidden_service(
        {'80': '127.0.0.1:65432'}, await_publication=True
    )
    Logger.get_logger().debug(f"Hidden service created with address: {self.hidden_service.service_id}.onion")
```

Figura 14: Método 'start' de la clase 'TorService'.

En este proceso, utilizamos la librería 'stem' para interactuar con el servicio de Tor. Mediante la llamada a 'stem.process.launch_tor_with_config', podemos iniciar el servicio de Tor de forma local, utilizando las configuraciones previamente establecidas. Esta llamada permite que el servicio se inicie con los parámetros necesarios y en consonancia con las preferencias de configuración que hemos definido.

Es importante mencionar que, en esta etapa, el servicio de Tor no es accesible externamente. Para habilitar las conexiones externas, es necesario realizar algunas operaciones adicionales. En primer lugar, creamos un controlador en nuestro código mediante la llamada a 'stem.control.Controller.from_port'. Esta operación nos permite manejar y controlar el servicio de Tor. Especificamos el puerto a través del cual deseamos establecer la comunicación con el servicio.

Una vez que hemos obtenido una instancia del controlador, procedemos a autenticar nuestra identidad dentro de la red de Tor mediante la llamada a 'tor_controller.authenticate'. Este proceso de autenticación garantiza que nuestra aplicación esté autorizada para realizar operaciones en la red de Tor y establecer conexiones con otros nodos.

Posteriormente, creamos un circuito hacia nuestro servicio utilizando la llamada a 'tor_controller.new_circuit'. Un circuito en Tor es una serie de nodos a través de los cuales se enruta el tráfico de red de forma anónima. El funcionamiento detallado de la creación de un circuito se puede encontrar en el apartado correspondiente del diseño del sistema. En resumen, la creación del circuito implica establecer una ruta segura y encriptada para las comunicaciones.

Una vez que hemos establecido el circuito, es posible habilitar la accesibilidad externa a nuestro servicio. Para ello, utilizamos la llamada a 'tor_controller.create_ephemeral_hidden_service'. Este método genera una clave pública que se utiliza para cifrar las conexiones entrantes a nuestro servicio y, a su vez, se convierte en la dirección de Tor que se utilizará para acceder a nuestro servicio desde el exterior.

3.2.4. Creación de la conexión en escucha

Como mencionamos anteriormente, la clase 'NetworkManager' incluye una instancia de 'ConnectionHandler', la cual se ejecuta en un hilo separado y se encarga de gestionar las conexiones. 'ConnectionHandler' hereda del sistema multihilo, por lo tanto, hemos realizado una sobreescritura del método 'run' para establecer un bucle continuo de funcionamiento.

El método 'run', que podemos observar en la Figura 16 se encarga de coordinar el funcionamiento principal de 'ConnectionHandler'. En primer lugar, se crea un contexto utilizando la clase 'Socket', que se encarga de configurar y establecer el socket necesario para la escucha de conexiones. Dentro del bucle, se verifica si la bandera de terminación 'terminate_flag' no está activada. En caso de ser así, el hilo continúa su ejecución.

```
Joel Otero
def run(self):
    with Socket(self.network_manager.host, self.network_manager.port) as sock:
        while not self.terminate_flag.is_set():
            try:
                conn, _ = sock.accept()
                new_connection = self.create_new_connection(self.deprotocol, conn, False)
                self.network_manager.node_connections.append(new_connection)
                Logger.get_logger().info("Connection created with a client.")
                new_connection.start()
            except socket.timeout:
                continue
            except socket.error as exc:
                if exc.errno == errno.ECONNRESET:
                    Logger.get_logger().error(f"SocketClosed: {str(exc)}")
            except Exception as exc:
                Logger.get_logger().error(exc)
        self.stop()
```

Figura 15: Método 'run' de la clase 'ConnectionHandler'.

Dentro del bucle, se utiliza el método 'accept' del socket para esperar y aceptar nuevas conexiones entrantes. Esta llamada bloqueante devuelve un nuevo socket y otra información relevante sobre la conexión, pero en este caso, solo nos interesará el nuevo socket, el cual almacenamos en la variable 'conn'. A continuación, se crea una nueva instancia de 'NodeConnection' utilizando el método 'create_new_connection', pasando como argumento el objeto 'deprotocol', el socket 'conn' y el valor False para el parámetro 'initiator'. Esta instancia de 'NodeConnection' se agrega a la lista 'node_connections' del objeto 'network_manager'. Luego, se llama al método 'start' en la instancia de 'NodeConnection' recién creada, lo que inicia la ejecución de su propio hilo correspondiente.

En caso de producirse una excepción de tiempo de espera (timeout) en el socket, se continúa con la siguiente iteración del bucle sin interrumpir la ejecución del hilo. Además, se manejan excepciones relacionadas con el socket, como por ejemplo, cuando se produce un error de conexión reseteada (ECONNRESET). En caso de producirse alguna otra excepción no manejada, se registra en el 'Logger' correspondiente.

Una vez que la bandera de terminación se activa y se sale del bucle, se llama al método 'stop' para finalizar adecuadamente el hilo 'ConnectionHandler' y cerrar todas las conexiones activas.

```python
class Socket:
    werogg
    def __init__(self, host: str, port: int):
        self.host = host
        self.port = port
        self.sock = None

    werogg
    def __enter__(self):
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.sock.bind((self.host, self.port))
        self.sock.settimeout(30.0)
        self.sock.listen(1)
        return self.sock

    werogg
    def __exit__(self, exc_type, exc_val, exc_tb):
        self.sock.close()
```

Figura 16: Clase 'Socket'.

Es importante destacar la clase 'Socket' cuya función es configurar e iniciar el socket con los parametros correspondientes a nuestra configuración.

3.2.5. Establecer una conexión

Para explicar cómo se ha implementado el establecimiento de conexiones, partimos de la clase 'Client' de la API, específicamente del método 'connect'.

En este método, especificamos mediante argumentos la dirección de Tor del cliente al que nos queremos conectar. El método delega la lógica a la clase 'ConnectionHandler', que es la encargada de gestionar las conexiones.

Primero, creamos un nuevo socket y configuramos el proxy con los ajustes especificados en la configuración como vemos en la Figura 17.

```
def connect_to(self, address, port, initiator=False):
    sock = socks.socksocket()
    sock.settimeout(120)
    sock.setproxy(PROXY_TYPE, PROXY_HOST, PROXY_PORT)

    Logger.get_logger().info(f"connecting to {address} port {port}")
```

Figura 17: Creación y configuración del socket.

Después, utilizando el método 'connect' de dicho socket, abrimos la conexión. Si no ocurre ninguna excepción, creamos un objeto 'NodeConnection' con la nueva conexión. Pasamos como parámetros la instancia de DeProtocol, el socket y un booleano que indica si nosotros estamos iniciando dicha conexión como se observa en la Figura 18. El objeto de esta conexión es retornado y almacenado en 'NetworkManager'.

```
try:
    sock.connect((address, port))
except Exception as exc:
    Logger.get_logger().error(exc)

Logger.get_logger().info(
    f"NodeConnection.send: Started with client ({address}) ':{str(port)}'"
)
new_connection = self.create_new_connection(self.deprotocol, sock, initiator)
new_connection.start()

return new_connection
```

Figura 18: Código para la inicialización de la conexión hacia otro cliente.

Además, iniciamos el hilo del 'NodeConnection' utilizando el método 'start'. Esta clase se encarga de manejar una conexión única a otro cliente. Mediante esta llamada a 'start', iniciamos el protocolo 'mantente vivo' y enviamos un paquete de 'handshake' para intercambiar datos de dirección, nombre de usuario, imagen de perfil, clave pública y si estamos iniciando la conexión (Figura 19).

Por otro lado, el cliente que está en espera de una conexión recibirá la apertura del socket. Entonces, realiza las mismas llamadas para crear un nuevo 'NodeConnection'

```python
def start(self):
    self.pinger = Pinger(self)
    self.pinger.start()
    self.send_packet(
        PacketFactory.create_packet(
            PacketType.HANDSHAKE,
            HandshakePayload(address=self.deprotocol.node.onion_address,
                             nickname=self.user.nickname,
                             profile_img=self.user.profile_img,
                             public_key=cf.serialize_key(self.public_key),
                             initiator=self.initiator).serialize()))
    super().start()
```

Figura 19: Código para el inicio del protocolo 'mantente vivo' y el intercambio de saludos.

y, de la misma forma que el iniciador, inicia el protocolo 'mantente vivo' y envía sus datos a través de otro paquete de 'handshake'.

En este punto, la conexión está completamente establecida y se pueden intercambiar datos de forma segura.

3.2.6. Protocolo mantente vivo

El protocolo "mantente vivo"se implementa en la clase 'Pinger' dentro del archivo 'pinger.py'. Este protocolo es un servicio simple diseñado para mantener el socket activo. Consiste en enviar periódicamente un paquete de 'ping' a intervalos predefinidos.

Además, nuestra gestión de la conexión espera recibir al menos un paquete cada cierto tiempo, como se especifica en la variable 'dead_time' dentro de 'Pinger'. Si en algún momento se excede el tiempo de espera o se detecta que el socket está cerrado al intentar enviar el paquete de 'ping', la conexión se finaliza de inmediato con el otro cliente.

Su implementación consta de muy pocas lineas como podemos observar en la Figura 20

```python
def run(self):
    Logger.get_logger().info("Pinger Started")
    time.sleep(1)
    while not self.terminate_flag.is_set():
        ping_packet = KeepAlivePacket()
        try:
            self.node_connection.packet_handler.send_packet(ping_packet)
        except Exception as exc:
            Logger.get_logger().error(exc)
        Logger.get_logger().trace(f'pinger_run: Ping packet sent, sleeping {self.send_time} seconds...')
        time.sleep(self.send_time)
```

Figura 20: Implementación del protocolo mantente vivo.

3.2.7. Sistema de consola y comandos

El sistema de consola y comandos se estructura en tres partes diferentes.

Por una parte tenemos la consola en si, no entraremos mucho en detalle sobre su implementación, pero la encontramos en la clase 'ConsoleUI', esta clase tiene la función de realizar un bucle cuya finalización viene determinado por una bandera de terminación, el cual únicamente recoge el texto introducido por el usuario, lo analiza y delega la gestión del mismo a la clase 'CommandHandler'.

La clase 'CommandHandler' es instanciada por el controlador, como se observa en la Figura 21. Contiene un diccionario que almacena los diferentes comandos registrados. Cada comando en el diccionario está asociado a un alias y a la clase correspondiente. Esto permite agregar comandos personalizados llamando al método 'register_command' para extender la funcionalidad de la aplicación.

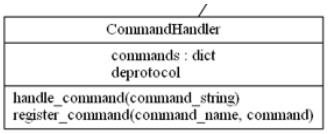

Figura 21: Diagrama de la clase CommandHandler.

En el método 'handle_command', se analiza el comando recibido y se busca la clase correspondiente en el diccionario. Luego, se delega la ejecución del comando a la clase correspondiente. Esto proporciona un enfoque modular y reutilizable para el procesamiento de comandos en la aplicación.

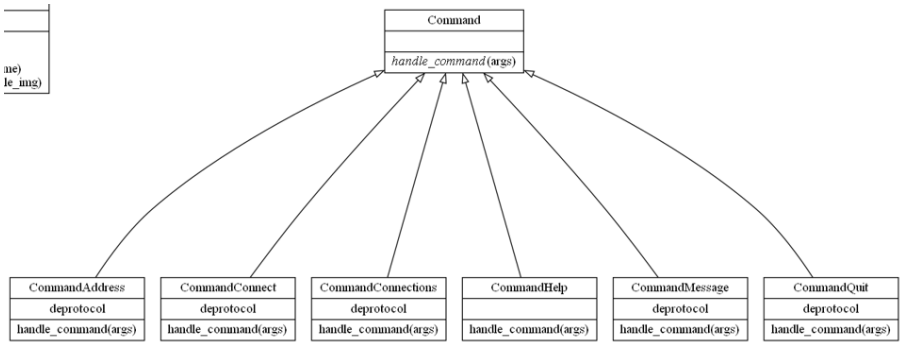

Figura 22: Diagrama de clases de los comandos por defecto.

Puedes consultar el diagrama de clases de los comandos por defecto en la Figura 22 para obtener una visualización de la estructura y las relaciones entre las clases de los comandos.

3.2.8. Sistema de registros

Nuestra implementación del sistema de registros o "logger"se basa en la clase "Logger". Este será un objeto único, siguiendo el patrón de diseño Singleton, que puede ser invocado desde cualquier parte del código para crear nuestros registros.

Durante la instanciación de la clase, se configura según los parámetros establecidos en el archivo "settings.py". Además, se encarga de crear las carpetas y archivos donde se almacenarán los registros, así como de definir el formato de cada registro. La clase también se encarga de crear el flujo de datos que se utilizará para mostrar los registros en tiempo de ejecución en la consola.

Este sistema se basa generalmente en el paquete de registro ("logging") de Python y se configura según nuestras necesidades. Sin embargo, hemos agregado un nivel adicional de registro llamado "TRACE", que nos permite realizar registros detallados para facilitar la identificación de errores de manera más eficiente y efectiva.

A lo largo del código, encontraremos llamadas a este sistema de registro de la siguiente manera:

- **Logger.get_logger().info(msg)** para registrar información.
- **Logger.get_logger().warning(msg)** para registrar advertencias cuando algo posiblemente no está funcionando como debería.
- **Logger.get_logger().error(msg)** para registrar errores.
- **Logger.get_logger().trace(msg)** para marcar una traza dentro del código y registrarla.

3.2.9. Sistema de paquetes y datos

Los paquetes por defecto, como hemos visto en el apartado de inizialicación, son registrados en el momento del inicio del protocolo. Estos paquetes son la base de nuestra comunicación. Como podemos observar en la Figura 23, los diferentes paquetes que están disponibles de forma inicial heredan directamente de 'Packet', así como sus propiedades y métodos.

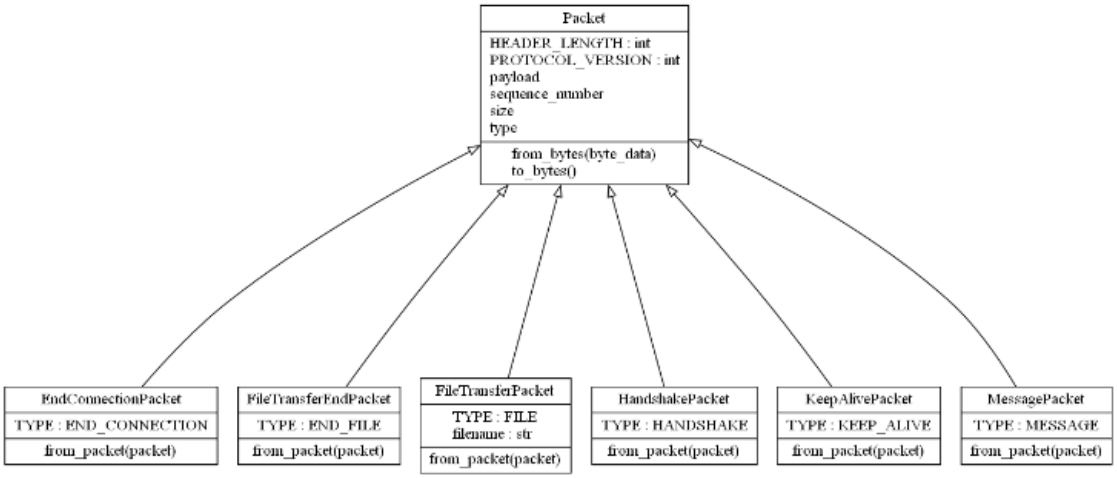

Figura 23: Diagrama de clases de los paquetes.

Cada paquete en el protocolo contiene varios elementos importantes. Esto incluye el tamaño de la cabecera en bytes, la versión del protocolo, el payload (datos específicos del paquete), el número de secuencia, el tipo de paquete y una propiedad general que se calcula en tiempo de ejecución, que es el tamaño total del paquete (suma de la cabecera y los datos).

Además, cada paquete implementa los métodos 'to_bytes' para codificar su contenido en formato de bytes y permitir su transmisión a través de la red, y 'from_packet' para reconstruir los objetos de los paquetes a partir de los bytes recibidos.

La estructuración y codificación de los datos se realiza de acuerdo a un diagrama de clases, como se muestra en la Figura 24. La clase base para los datos es 'Payload', que representa los datos en formato JSON. Cada tipo de dato tiene una implementación específica del método abstracto 'get_payload' en la clase 'Payload'. Sin embargo, todos los payloads comparten el método 'serialize', que se encarga de serializar y dar formato a los datos devueltos por 'get_payload'.

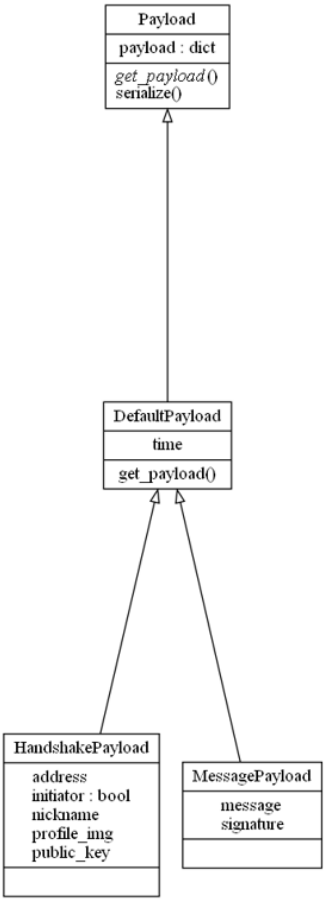

Figura 24: Diagrama de clases de los datos.

En el diagrama de clases, se puede observar que los payloads no heredan directamente de la clase 'Payload', sino de la clase 'DefaultPayload', que es una subclase de 'Payload'. La clase 'DefaultPayload' implementa el método 'get_payload', que recopila todos los datos contenidos en los atributos de cada objeto y los convierte en

un diccionario. Además, 'DefaultPayload' incluye por defecto una variable llamada 'time', que almacena el momento en el que los datos son generados.

Para la creación de los paquetes, se utiliza el patrón de diseño 'Factory'. En la clase 'PacketFactory', cada vez que deseamos crear un paquete, llamamos al método 'create_packet' con el tipo de paquete y el payload que contendrá. De esta manera, la clase se encarga de instanciar el objeto correcto según los paquetes registrados en nuestra aplicación.

Para lograr la escalabilidad del protocolo, es decir, añadir un nuevo tipo de paquete, debemos utilizar el método 'register_packet_type' que se encuentra en 'PacketFactory'. Le indicamos el enumerador correspondiente, el directorio y el nombre de la clase. Esto garantiza que el protocolo admita el nuevo tipo de paquete. Es recomendable también modificar la versión del protocolo si se añaden paquetes, para manejar posibles incompatibilidades en la aplicación.

3.2.10. Gestión de paquetes

La clase 'NodeConnection' es el punto de partida para comprender la gestión de conexiones establecidas. Cuando se establece una conexión exitosa, se realizan dos llamadas importantes: 'self.packet_handler.receive_packet' y 'self.handle_received_packet'.

Estas llamadas se delegan a la clase 'PacketHandler', cuyo propósito principal es la gestión de paquetes. Al examinar el diagrama de implementación en la Figura 25, se hace evidente la complejidad y la interconexión de esta funcionalidad, ya que implica la participación de varias clases.

La clase 'PacketHandler' desempeña un papel central en la recepción de paquetes entrantes y coordina su procesamiento. Es responsable de garantizar que los paquetes sean correctamente manejados y procesados dentro de la aplicación. La implementación detallada de esta funcionalidad involucra la colaboración de otras clases especializadas en el procesamiento y manipulación de paquetes. De esta forma se encarga de delegar las funciones de envío y recepción de paquetes a sus respectivas clases especializadas: 'PacketSender' y 'PacketReceiver'.

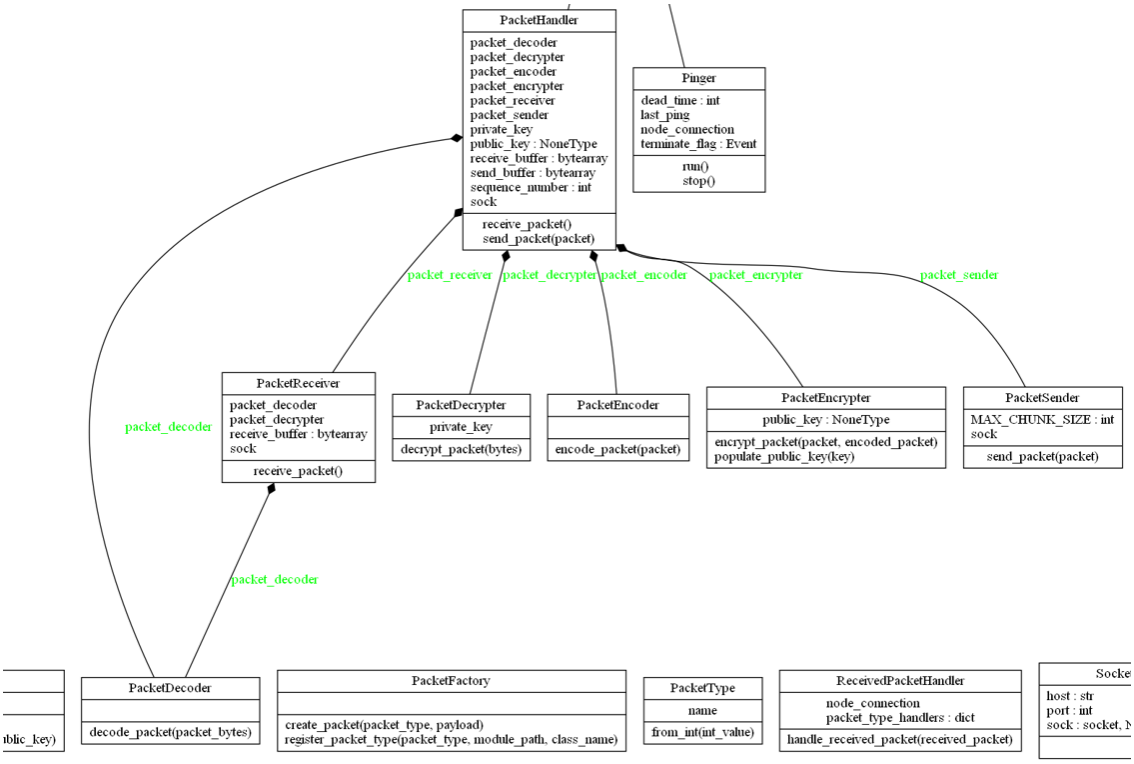

Figura 25: Diagrama de clases de la gestión de paquetes general.

La clase 'PacketReceiver' se encarga de recibir los datos entrantes y almacenarlos en una cola. Luego, realiza el análisis, desencriptado y decodificación de los paquetes para obtener el objeto de paquete correspondiente. Este objeto es devuelto para su procesamiento posterior a través del método 'handle_received_packet'.

Por otro lado, la función 'send_packet' de 'PacketHandler' realiza las llamadas necesarias a las clases 'PacketEncrypter' y 'PacketEncode' para encriptar y codificar los paquetes antes de enviarlos a través de 'PacketSender'.

El procesamiento final de los paquetes recibidos recae en la clase 'ReceivedPacketHandler'. Esta clase contiene el método 'handle_received_packet', el cual determina el tipo de paquete recibido y delega su gestión y ejecución a la clase hija correspondiente de 'PacketTypeHandler'. El diagrama de estructura de estas clases se puede observar en la Figura 26.

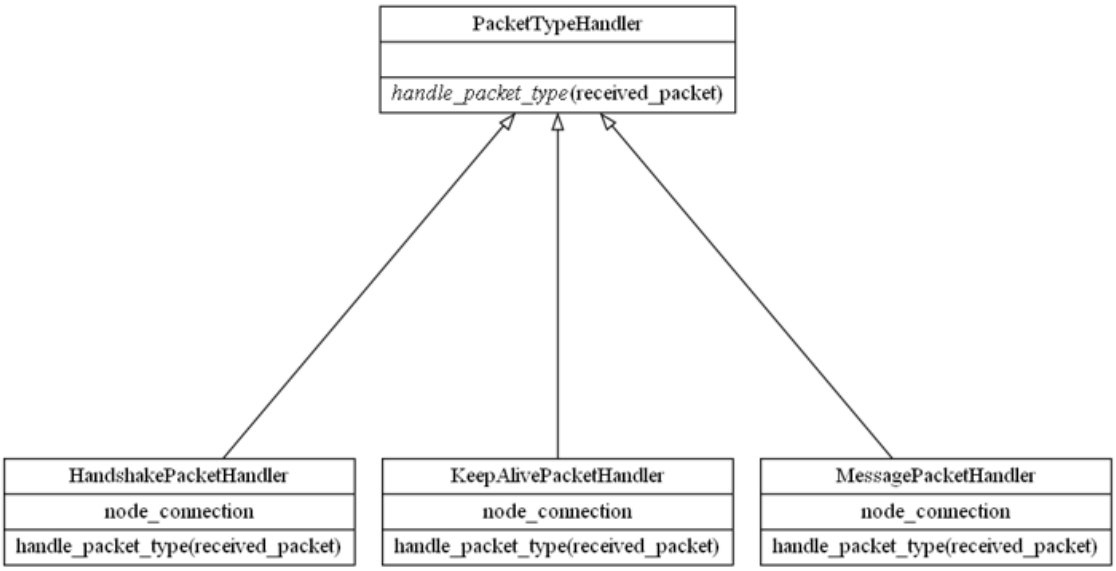

Figura 26: Diagrama de clases de la gestión de paquetes específica.

Dentro de cada método 'handle_packet_type', nos encargamos de recopilar y almacenar los datos recibidos en la sección correspondiente. Por ejemplo, en 'HandshakePacketHandler', se recopilan la dirección, el nombre de usuario, la imagen de perfil y la clave pública, y se establece una bandera que indica que el "handshake"se ha realizado con éxito. Para recopilar estos datos, utilizamos la misma estructura que hemos definido en los objetos 'Payload', por lo que es importante gestionar correctamente la versión del protocolo en caso de que haya cambios que puedan afectar los esquemas actuales.

3.2.11. Encriptación y desencriptación de los datos

Como hemos mencionado en la sección de gestión de paquetes, la responsabilidad de encriptar o desencriptar los datos recae en el 'PacketHandler'. En el diagrama de clases de la Figura 25, se puede observar que las clases encargadas de esta tarea son 'PacketEncrypter' y 'PacketDecrypter'.

Estas clases contienen la información necesaria para realizar la encriptación y desencriptación de los datos. En el caso de la encriptación, se utiliza la clave pública, mientras que en el caso de la desencriptación, se utiliza la clave privada. Una vez que se verifica que los paquetes no son del tipo 'handshake' ni 'keep_alive' (protocolo de mantenerse vivo), se realiza una llamada al archivo de utilidades 'crypto_funcs.py' para llevar a cabo la encriptación o desencriptación de los datos.

Para cifrar los datos con claves RSA de 2048 bits utilizando el algoritmo RSA, se realiza el proceso de cifrado en bloques o çhunks"de información. La razón detrás de esto es que los datos a cifrar pueden exceder el tamaño máximo permitido por las claves RSA de 2048 bits. De esta forma el cifrado RSA se aplica a cada bloque de datos de forma independiente. Así los datos se dividen en trozos más pequeños para que cada bloque pueda ser cifrado individualmente. De esta manera, se asegura que

cada bloque se ajuste al tamaño máximo permitido por las claves RSA.

Una vez que se han cifrado todos los bloques de datos, se juntan nuevamente y se realiza una codificación en formato Base64. La codificación en Base64 convierte los datos cifrados en una representación de texto legible y transportable, que puede ser enviada en formato de bytes a través de la red.

3.2.12. Firma y verificación de los mensajes

Tomando en cuenta las secciones anteriores, podemos entender que la clase 'NodeConnection' tiene la responsabilidad de gestionar una conexión específica. En esta clase se encuentra el método 'send_message', el cual se encarga de completar la información del paquete y delegar el envío al 'PacketHandler'. Durante este proceso de completar la información, se incluye la firma del mensaje a enviar junto con nuestra clave privada en el 'MessagePayload'. De esta manera, cualquier modificación en los datos resultará en una firma inválida.

A continuación, procedemos a la parte de procesar el paquete, lo cual se realiza a través del correspondiente 'PacketTypeHandler'. En el caso de un mensaje, específicamente en 'MessagePacketHandler', al procesar el paquete se verifica la firma y se almacena el resultado de esta verificación.

Para llevar a cabo la firma y verificación de los mensajes, contamos con una utilidad en la clase 'MessageAuthenticator', la cual incluye dos métodos estáticos (Figura 27: uno para realizar la firma y otro para realizar la verificación. Estos métodos utilizan las funciones 'sign' y 'verify' de nuestra utilidad 'crypto_funcs.py'.

```
def sign(message, private_key):
    """ Write a signature using a private key """
    digest = get_digest(message)
    signer = PKCS1_v1_5.new(private_key)
    sig = signer.sign(digest)

    return base64.b64encode(sig).decode("utf-8")


 werogg
def verify(message, sig, key):
    """ Verify a signature using a public key """
    digest = get_digest(message)
    verifier = PKCS1_v1_5.new(key)
    return verifier.verify(digest, base64.b64decode(sig))
```

Figura 27: Métodos de firma y verificación.

3.2.13. Sistema de eventos

Uno de los elementos clave en nuestro sistema es la escalabilidad, por lo tanto, es necesario implementar un sistema de eventos que nos permita definir y gestionar diferentes sucesos, como la disponibilidad del protocolo o la llegada de un paquete. Para ello, hemos desarrollado un sistema de escucha de eventos.

Este sistema se compone de varias partes:

Modelos de eventos: Cuando ocurre un evento, a veces es necesario agregar información adicional para proporcionar contexto. Es por eso que hemos implementado la clase base 'Event', de la cual heredan dos tipos de eventos. El primero es 'DeProtocolReadyEvent', que no contiene información y simplemente indica que el protocolo está listo para ser utilizado. Por otro lado, tenemos 'PacketReceivedEvent', que es la base de la cual heredarán el resto de eventos relacionados con la recepción de paquetes y que gestionarán la información según su responsabilidad:

- **PacketReceivedEvent:** Almacena el paquete recibido y el objeto 'NodeConnection' a través del cual se ha recibido. Los demás eventos de paquetes heredarán esta información.
- **MessageReceivedEvent:** Convierte los datos del paquete en un diccionario y utiliza esta estructura para almacenar la hora indicada en el paquete, el mensaje, si ha pasado la verificación y el nombre de usuario del cliente que lo ha enviado.
- **HandshakeReceivedEvent:** De manera similar, convierte los datos del paquete en un diccionario y almacena la dirección del cliente, el nombre de usuario, la imagen de perfil, la clave pública y la condición de iniciador de la conexión.
- **KeepAliveReceivedEvent:** Indica la recepción de un paquete de tipo 'keep_alive' del protocolo, pero no almacena ninguna información adicional.

Podemos ver su estructura en la Figura 28

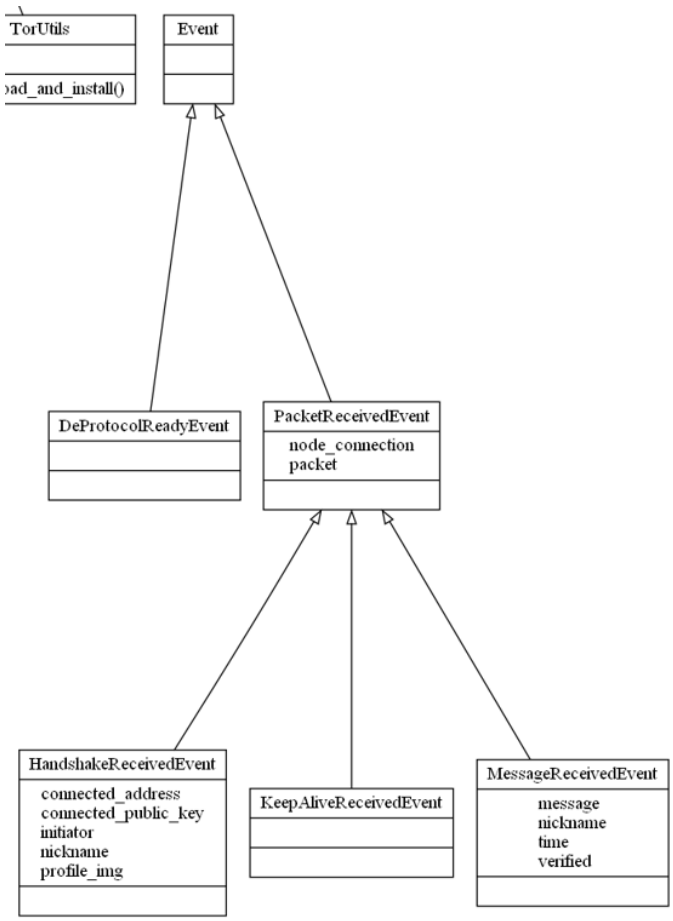

Figura 28: Diagrama de clases de los modelos de los eventos.

La segunda parte es la **escucha y gestión de los eventos**. Para poder lanzar eventos y notificar a los diferentes componentes interesados, hemos implementado las clases 'Listener' y 'Listeners'.

La clase 'Listener' es una clase abstracta que define el método 'handle_event'. Esta clase sirve como base para la creación de otros listeners que estarán atentos a eventos específicos. Los listeners deben implementar el método 'handle_event' para procesar el evento recibido. Esta clase es responsable de gestionar y registrar los listeners. Tiene un diccionario llamado 'listeners' donde se almacenan los listeners registrados, organizados por el tipo de evento al que están suscritos. Cuando se registra un listener utilizando el método 'register_listener', se valida que sea una instancia de la clase 'Listener' y se agrega a la lista correspondiente en el diccionario de listeners.

La función 'fire' se utiliza para lanzar un evento. Recibe como argumento un objeto de tipo 'Event' y busca en el diccionario de listeners los listeners asociados al tipo de evento recibido. Luego, invoca el método 'handle_event' de cada listener registrado, pasándole el evento como argumento. Podemos encontrar llamadas a este método distribuidas por todo el código.

3.3. Despliegue de la aplicación

En esta sección, describiremos los pasos necesarios para desplegar y ejecutar la aplicación en los diferentes modos disponibles. Además, abordaremos la realización de pruebas unitarias y de aceptación, así como los prerrequisitos necesarios para llevar a cabo el despliegue de manera exitosa.

3.3.1. Prerrequisitos

El proyecto ha sido desarrollado utilizando 'IntelliJ IDEA' como entorno de desarrollo principal. Recomendamos encarecidamente utilizar este entorno para facilitar la instalación y evitar posibles problemas. Sin embargo, también es posible utilizar otros entornos de desarrollo compatibles.

Partimos de la carpeta donde se encuentra el código. Antes de todo necesitamos instalar 'Python 3.10', lo podemos descargar e instalar directamente de la web oficial (https://www.python.org/downloads/).

1. Verificar la instalación de 'Python 3.10':

   ```
   python --version
   ```

2. Crear el entorno virtual (no es necesario, pero si recomendado):

   ```
   python -m venv myenv
   ```

3. Activar el entorno virtual:

   Linux/MacOS:

   ```
   source myenv/bin/activate
   ```

   Windows:

   ```
   myenv\Scripts\activate.bat
   ```

4. Instalar las dependencias:

   ```
   pip install -r requirements.txt
   ```

Una vez llegados a este punto estamos listos para lanzar el proyecto.

3.3.2. Como desplegar DeProtocol en modo consola

Para ejecutar la aplicación únicamente tenemos que ejecutar el siguiente comando desde la raíz del proyecto:

```
python src/deprotocol/app/main.py
```

Podremos ver entonces el siguiente mensaje avisando del modo en el que estamos corriendo la aplicación:

```
WARNING: Running DeProtocol in CONSOLE MODE!
```

Los diferentes servicios requeridos para el funcionamiento del protocolo empezaran a iniciarse. Una vez nuestra aplicación esté lista observaremos el siguiente mensaje:

```
INFO: Starting DeProtocol version 0.0.3, running on Windows
```

A partir de este punto ya podremos comenzar a utilizar DeProtocol, utiliza el comando 'help' para mostrar las distintas opciones.

3.3.3. Como desplegar las pruebas unitarias

Para desplegar las pruebas unitarias tenemos que ejecutar el siguiente comando:

```
pytest test/
```

Es importante especificar el directorio donde se encuentran nuestros archivos de pruebas.
Si todo está correctamente observaremos un mensaje indicando el inicio de la prueba, los tests ejecutandose y un mensaje indicando la finalización con la cantidad de tests que han pasado satisfactoriamente (Figura 29).

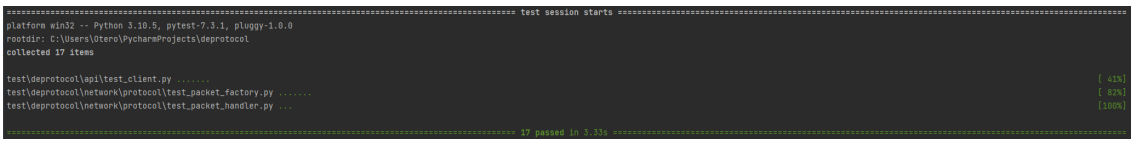

Figura 29: Ejecución satisfactoria de los tests unitarios.

3.3.4. Como desplegar las pruebas de aceptación

Para desplegar las pruebas de aceptación de forma correcta es importante partir de un entorno limpio y nuevo, de forma que no pueda quedar ningún puerto en escucha pendiente de cerrar o alguna cosa que pueda provocar inestabilidad al entorno.

Ejecutaremos el siguiente comando que lanzara las pruebas de aceptación:

```
python .\test\main.py
```

Aunque si estamos en el entorno de desarrollo 'IntellIJ IDEA', recomendamos lanzarlos desde el botón de ejecutar que observaremos en el archivo 'communication.feature' dentro de 'test/features/'.

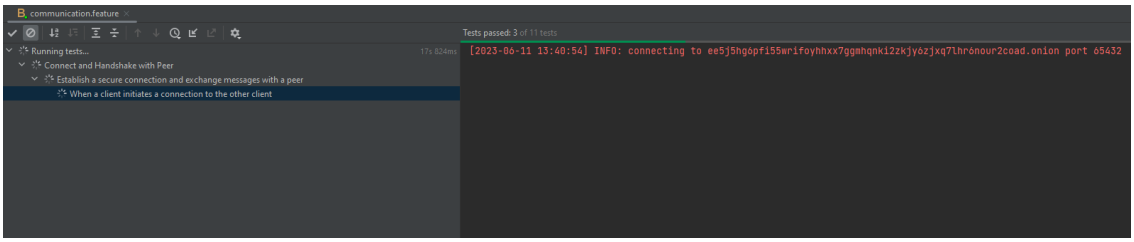

Figura 30: Ejecución de las pruebas de aceptación.

Esto nos permite tener una visión más detallada de los pasos que se están ejecutando, como se muestra en la Figura 30

3.3.5. Como desplegar la aplicación de ejemplo

Para desplegar la aplicación de ejemplo vamos a ejecutar el siguiente comando:

```
python .\src\sample_gui_app\main.py
```

Si no hemos cambiado la configuración, observaremos que nos cargará una interfaz cuando la consola muestre que nuestro protocolo ha sido iniciado correctamente.

3.3.6. Como configurar DeProtocol

Dentro del paquete 'deprotocol' podemos encontrar un archivo llamado 'settings.py' este contiene todas las configuraciones necesarias para el correcto funcionamiento del protocolo por defecto. No obstante podemos facilmente ajustarlas a nuestras necesidades.

En esta sección explicaremos las más relevantes a la hora de ejecutar nuestra aplicación:

- **USE_CONSOLE:** Esta configuación nos permite desactivar la consola interactiva de nuestra aplicación, de esta forma bloqueamos por completo el uso del protocolo por medio de comandos, de forma que únicamente se podrá manejar a través de la API.
- **PROXY_HOST / PROXY_PORT:** Permite establecer la dirección y puerto donde se encuentra el 'proxy' intermediario con la red de Tor.
- **PROXY_TYPE:** Sirve para establecer el tipo de 'proxy' a utilizar, existen variantes como SOCKS4, HTTP o HTTPS, aunque la más recomendada es la que viene por defecto SOCKS5 pues es la más común y usada porque es completamente compatible con todas las características de Tor. El resto pueden resultar en una limitación de funciones o suponer un riesgo para la seguridad del protocolo.

- **HIDDEN_SERVICE_HOST / HIDDEN_SERVICE_PORT:** Establece la dirección y puerto del servicio oculto a nivel de cliente, es donde iran redirigidas todas las conexiones.
- **HIDDEN_SERVICE_VIRTUAL_PORT:** Este puerto por contra es el expuesto a la red de Tor, todas las conexiones a este puerto iran redirigidas al servicio oculto presente en el cliente.
- **NODE_HOST / NODE_PORT:** Esta configuración establece la dirección y el puerto de nuestra puerta de enlace principal a nivel de cliente. Es la configuración del "servidor"peer-to-peer.
- **DEBUG:** Este valor booleano nos permite obtener información adicional tanto en los registros como durante la ejecución del programa. Su objetivo es facilitar el desarrollo y mejorar el análisis de errores de manera más eficiente. Al activarlo, se obtendrá información detallada y adicional que puede ser útil para identificar y solucionar problemas durante el desarrollo del software.
- **TRACE:** Este valor, similar a 'DEBUG', activará el nivel de traza en los registros. Al activarlo, cada pequeña llamada relevante para la búsqueda de errores será marcada y registrada.
- **TOR_BINARIES:** En esta sección, se presentan diversas configuraciones para ubicar los servidores desde los cuales descargaremos los binarios de Tor. También se indica la ubicación del directorio donde se encontrarán los archivos una vez que se hayan descomprimido.
- **TEST_SLOW_TIMEOUT:** El valor especificado en segundos indica el tiempo de espera utilizado por nuestras pruebas de aceptación para verificar si una llamada se ha realizado correctamente. Se incluye un pequeño margen de tiempo para tener en cuenta las posibles variaciones en los tiempos de comunicación con la red de Tor, que pueden depender del momento en que se realicen las pruebas.

3.4. Prueba de ejecución

En esta sección, se llevará a cabo la prueba de ejecución del protocolo en modo consola, así como de la aplicación de ejemplo. La prueba se realizará siguiendo los pasos descritos a continuación.

3.4.1. DeProtocol en modo consola

Para realizar la prueba del protocolo en modo consola, primero debemos abrir nuestro entorno virtual y ejecutar DeProtocol siguiendo los pasos explicados anteriormente utilizando el siguiente comando:

```
python src/deprotocol/app/main.py
```

Esto abrirá la consola interactiva y se iniciará el proceso de carga del protocolo. Una vez completado, se mostrará un mensaje indicando que el protocolo está listo para su uso. La apariencia de la consola será similar a la mostrada en la Figura 31.

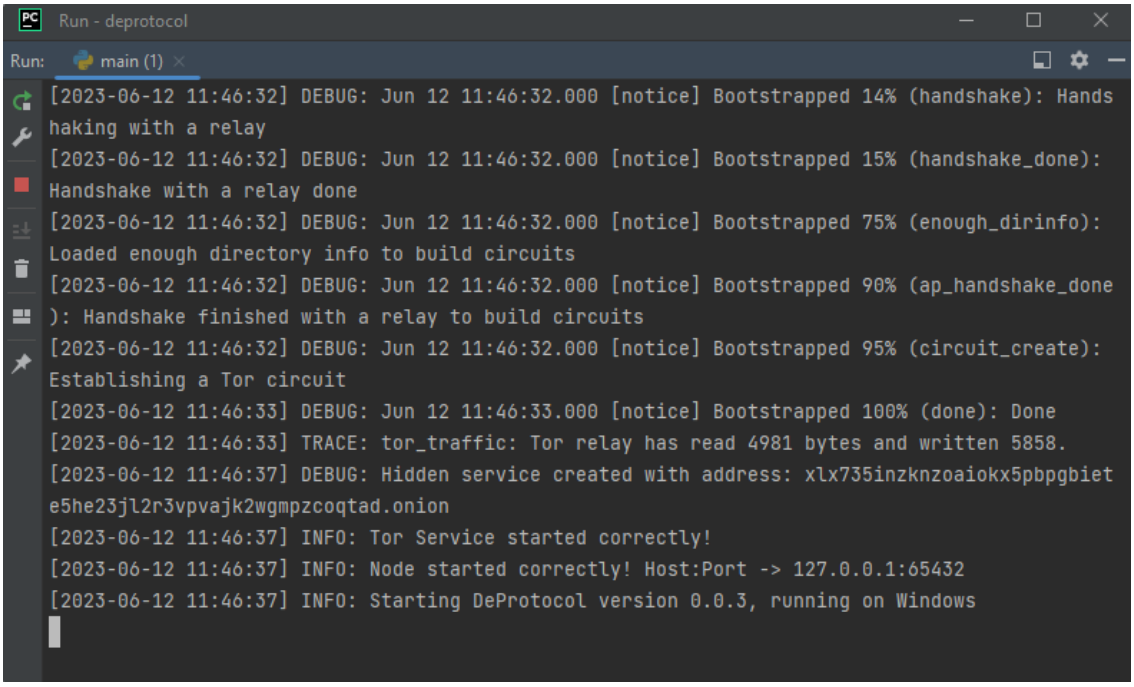

Figura 31: Cliente 1: Consola mostrando que el protocolo está listo para funcionar.

En el segundo cliente que se encuentra en ejecución en una máquina virtual, escribiremos el comando **address** tal como se muestra en la Figura 32. Esto nos permitirá observar y copiar la dirección de Tor al portapapeles para su posterior uso.

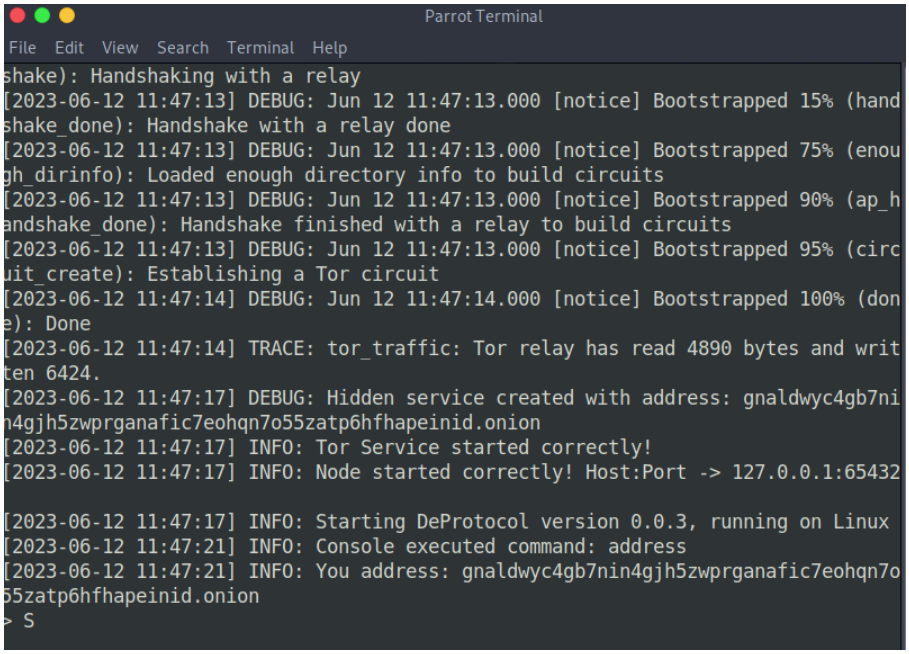

Figura 32: Cliente 2: Ejecución del comando 'address' en la consola.

Ahora, en el primer cliente, utilizaremos el comando **connect ¡dirección¿** para establecer una conexión con la dirección previamente obtenida. Esto se ilustra en la Figura 33.

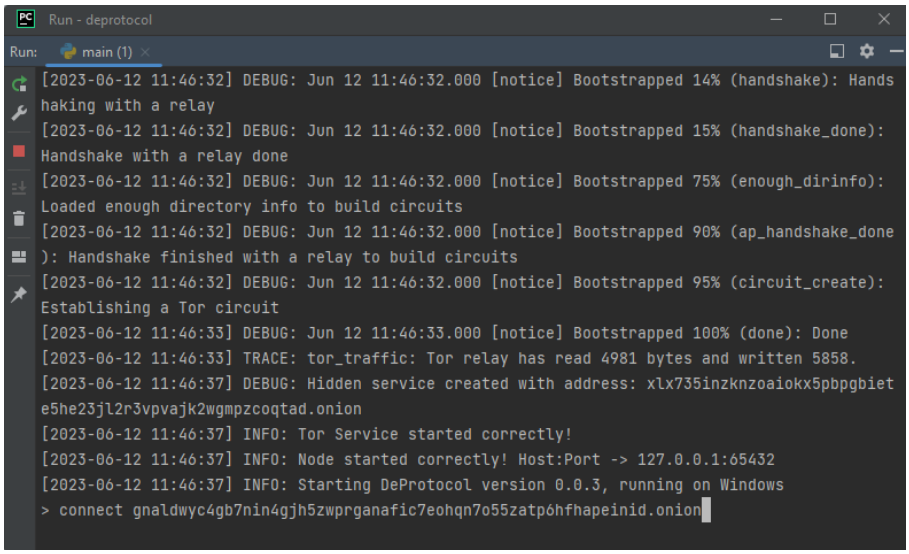

Figura 33: Cliente 1: Ejecución del comando 'connect' en la consola.

Una vez iniciada la conexión, veremos el mensaje 'NodeConnection.send: Started with client' (Figura 34). Es importante tener en cuenta que en este punto la conexión no está lista. Debemos esperar al intercambio de saludos, el cual se muestra en la Figura 35.

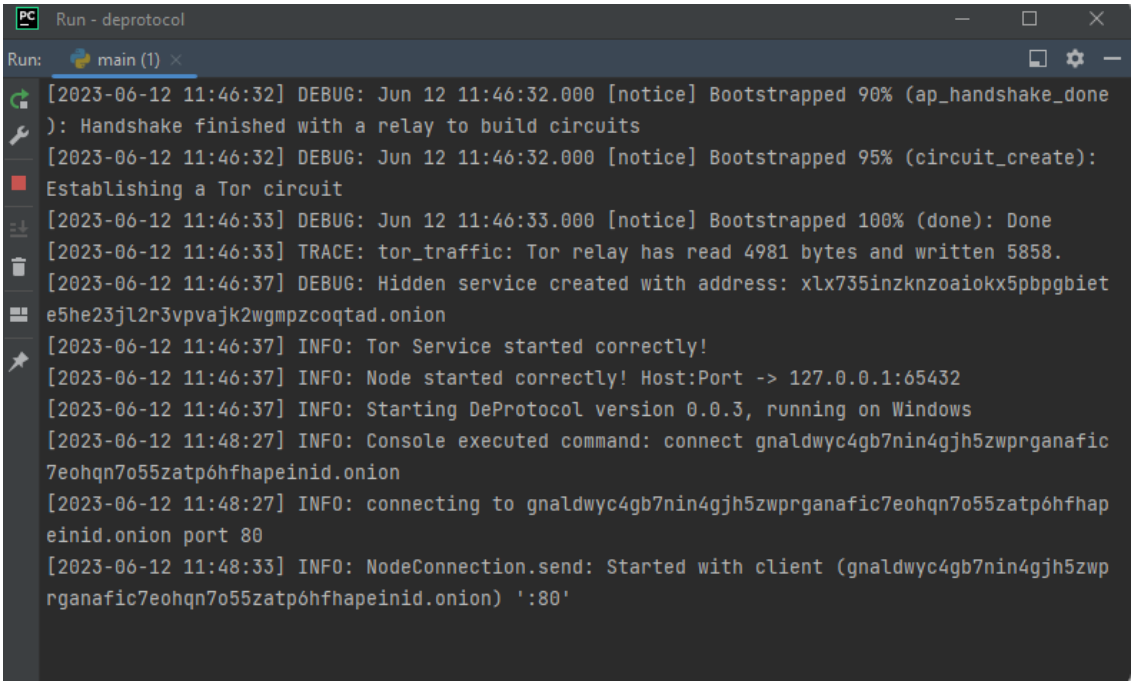

Figura 34: Cliente 1: Información en consola del establecimiento de conexión.

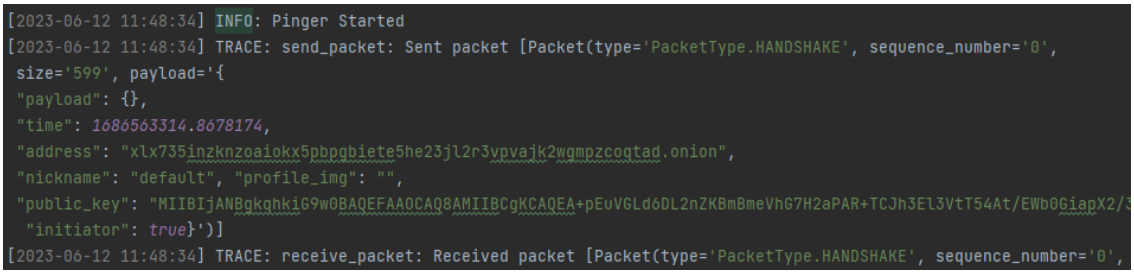

Figura 35: Cliente 1: Mensaje recogido del registro mostrando el intercambio de saludos.

Ahora estamos listos para establecer la comunicación. Para ello, ejecutamos el comando **connections**, el cual mostrará una lista de conexiones junto con sus índices, como se muestra en la Figura 36. De esta manera, sabremos con qué conexión debemos comunicarnos, en este caso, sería la conexión 0.

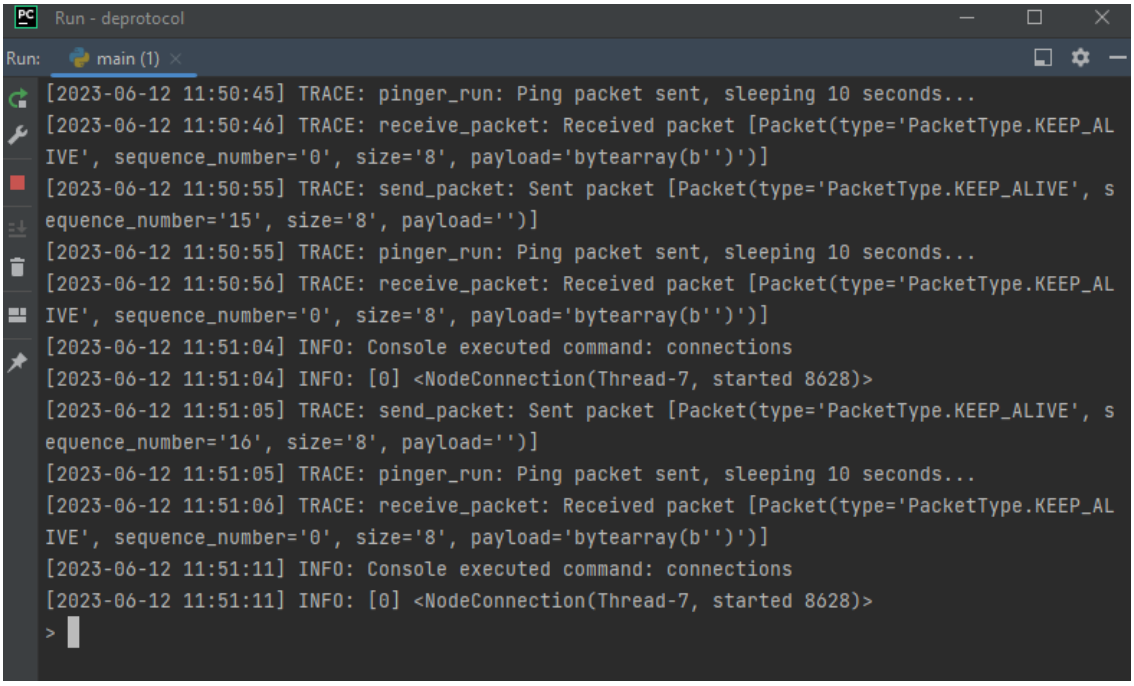

Figura 36: Cliente 1: Resultado del comando 'connections' en consola.

Para enviar un mensaje, simplemente ejecutamos el comando **message ¡id¿¡Mensaje¿**. Por ejemplo, para enviar el mensaje .Esto es una prueba!.a la conexión 0, utilizaríamos el comando **message 0 Esto es una prueba!**, como se muestra en la Figura 37. Podremos observar la recepción del mensaje en el segundo cliente, como se muestra en la Figura 38.

Figura 37: Cliente 1: Envio de un mensaje al nodo 0 a través del comando 'message'.

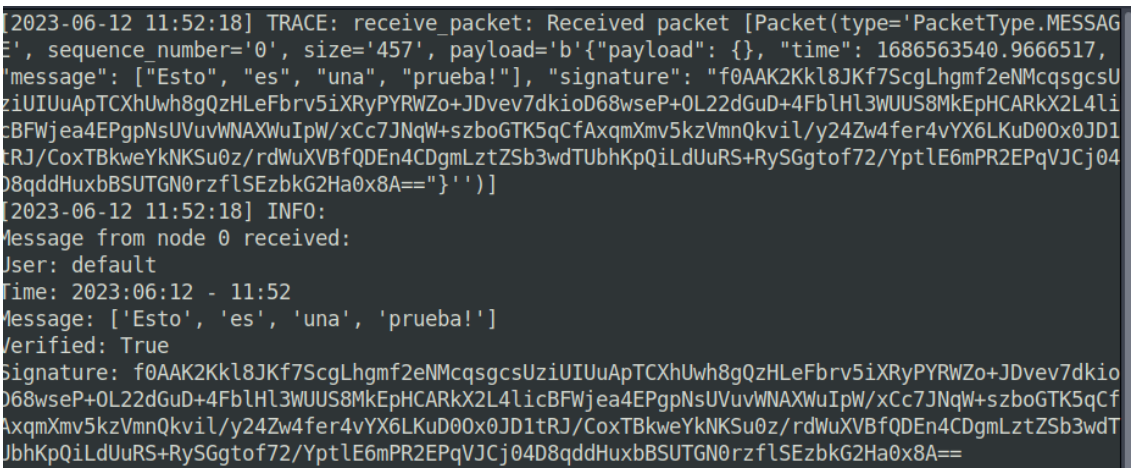

Figura 38: Cliente 2: Recepción del mensaje enviado por Cliente 1.

3.5. Aplicación de ejemplo

Hemos desarrollado una aplicación de ejemplo utilizando la API de DeProtocol para probar el uso y la escalabilidad de la librería. Puedes lanzar fácilmente esta aplicación siguiendo los pasos explicados en la sección de despliegue. Utiliza el siguiente comando:

```
python .\src\sample_gui_app\main.py
```

Una vez ejecutado, se abrirá la consola y el protocolo comenzará a ejecutarse. Una vez que esté listo, se abrirá la interfaz de la aplicación, como se muestra en la Figura 39. En este momento, deberás copiar la dirección de Tor que se muestra en la interfaz.

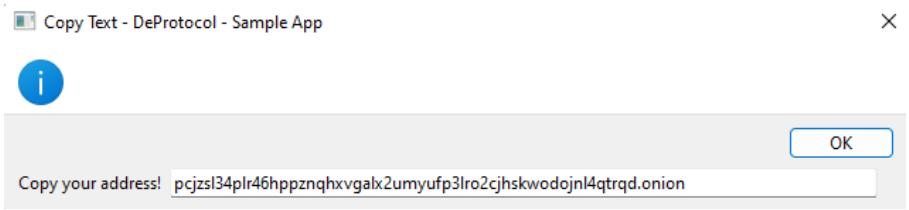

Figura 39: Aplicación de ejemplo mostrando la dirección de Tor establecida.

Después de hacer clic en 'OK', se abrirá la interfaz del chat (Figura 40), la cual es muy sencilla e intuitiva de usar. En primer lugar, configuraremos nuestro nombre y foto de perfil siguiendo los pasos ilustrados en la Figura 41. Para ello, accederemos a la sección Çonexiones.[en] la parte superior de la interfaz, y luego haremos clic en Çonfigurar". Allí podremos ingresar nuestro nombre de usuario y también tenemos la opción de cargar una imagen de perfil. Es importante destacar que la información se transmitirá, aunque actualmente está diseñado para futuras implementaciones de imágenes en formato "bytearray".

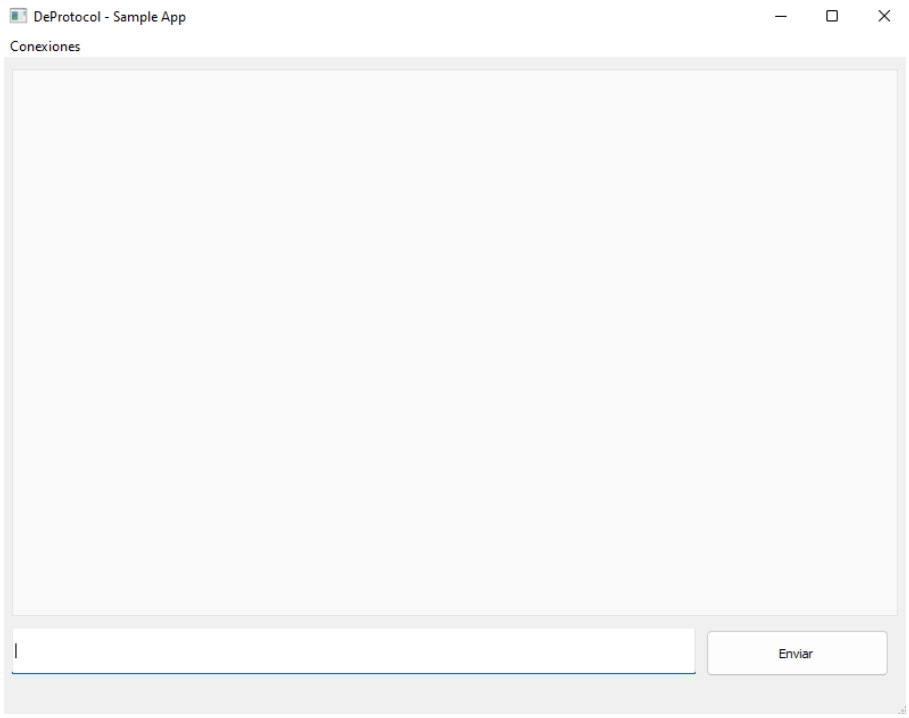

Figura 40: Pantalla principal de la aplicación de ejemplo.

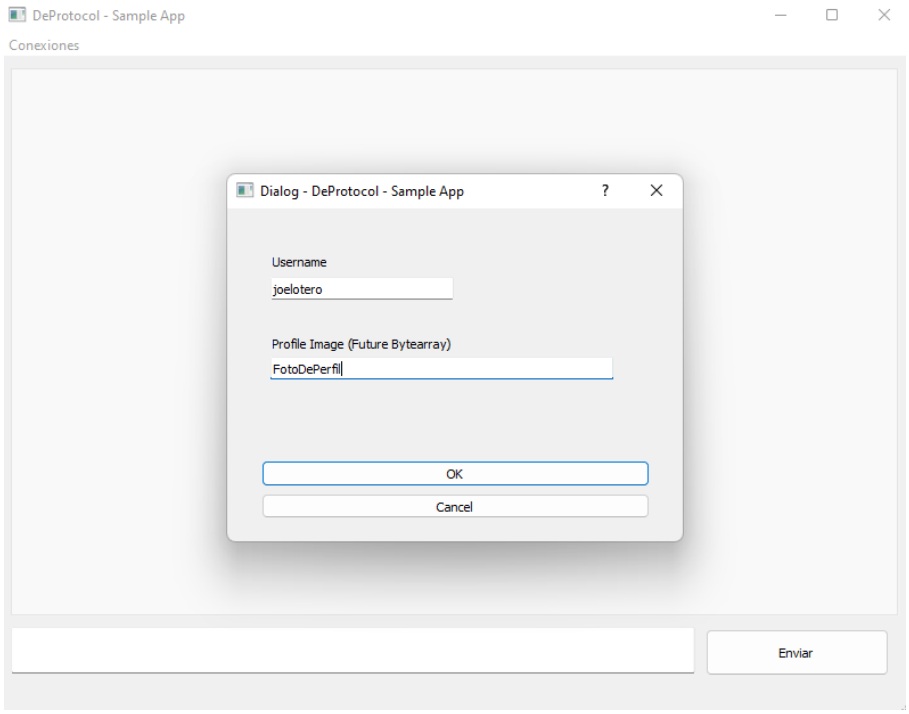

Figura 41: Pantalla de connfiguración de la aplicación de ejemplo.

Una vez que hayamos seleccionado 'OK', los valores de configuración se guardarán. Ahora, solo nos queda hacer clic nuevamente en 'Conexiones' y luego en 'Nueva...'. En este punto, veremos dos campos. El primer campo, denominado **nombre**, nos permite asignar un sobrenombre a nuestro chat. Si no introducimos ningún valor, se utilizará la dirección que deberemos ingresar en el segundo campo.

Ahora, solo nos queda esperar. Cuando la conexión esté lista, se abrirá una nueva pestaña con el chat correspondiente. Ahí podremos escribir un mensaje y hacer clic en el botón 'Enviar' (Figura 42) para enviarlo.

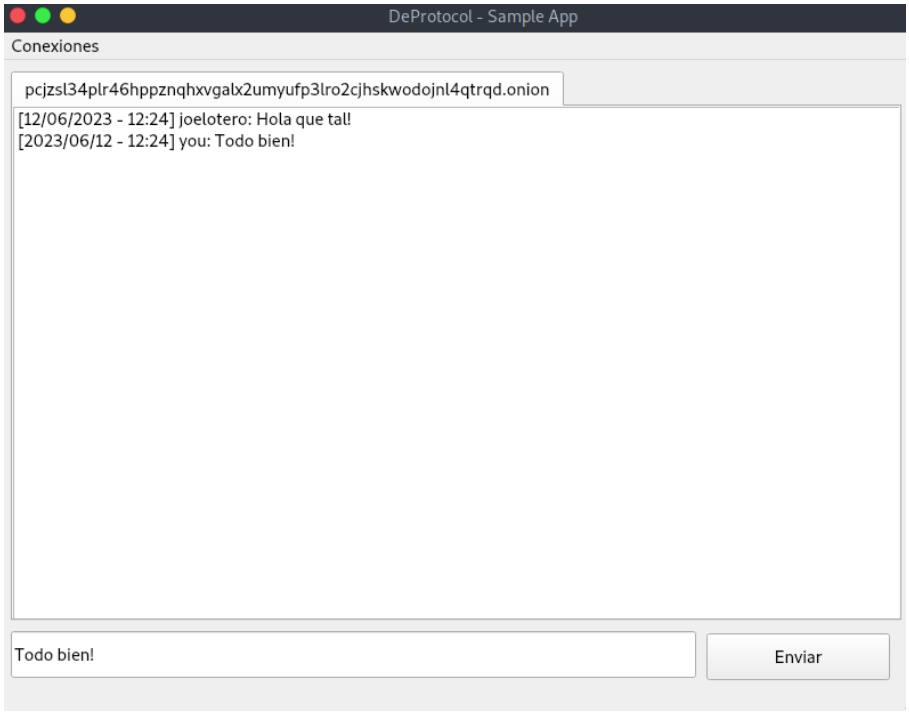

Figura 42: Intercambio de mensajes en la aplicación de ejemplo.

3.5.1. Aplicaciones compiladas

Como parte de la prueba, se ha creado una carpeta denominada 'compiled_deprotocol' que alberga cuatro versiones precompiladas del software denominado deprotocol. Estas versiones son especialmente diseñadas para funcionar en entornos Windows. Entre ellas se incluyen 'deprotocol.exe', que representa la ejecución del protocolo en modo consola, y 'derotocol-ui' y 'deprotocol-ui-console', que son dos aplicaciones de ejemplo con y sin consola, respectivamente.

Además, se ha desarrollado una versión adicional denominada 'wheel' de deprotocol, la cual puede ser instalada como un paquete de Python. Esto permite aprovechar la funcionalidad de la biblioteca deprotocol en el proceso de desarrollo, facilitando así la integración y utilización del software en proyectos basados en Python.

4. Análisis de los resultados

Durante el desarrollo de este proyecto, se han ido realizando pruebas exhaustivas para evaluar el funcionamiento y la eficacia del protocolo P2P distribuido y anónimo implementado. A continuación, se presenta un análisis de los resultados obtenidos en las pruebas, teniendo en cuenta diferentes aspectos, como la plataforma de implementación, los entornos de prueba y el rendimiento del protocolo.

4.1. Plataformas

En cuanto a la plataforma, se han realizado pruebas satisfactorias en dos sistemas operativos principales: Windows y Linux. Windows ha sido el sistema operativo principal para el desarrollo, el protocolo pudo implementarse y ejecutarse sin problemas significativos. Sin embargo, en el caso de Linux, se encontraron algunas limitaciones y se detectó una menor estabilidad en comparación con Windows. Esto se debe en parte a que las pruebas en Linux fueron menos extensivas y, por lo tanto, no se puede garantizar plenamente la estabilidad del protocolo en esta plataforma. Cabe mencionar que el protocolo también se ha implementado para Darwin (el núcleo de macOS), pero no se ha podido probar en este entorno debido a limitaciones de tiempo y recursos.

4.2. Entornos

Para evaluar el funcionamiento del protocolo, se han llevado a cabo pruebas en diferentes entornos:

4.2.1. Máquina local

En primer lugar, se han realizado pruebas en un entorno simulado en una misma máquina local. Sin embargo, se observaron problemas debido a que el entorno simulado se volvía inestable en algunos casos, lo que afectaba el funcionamiento del protocolo. Esto se debió, en parte, al uso de los mismos puertos y recursos por parte del entorno simulado, lo que provocaba conexiones entre componentes del protocolo en la misma máquina. A pesar de estas limitaciones, se pudo comprobar que el protocolo era funcional en dicho entorno, aunque se recomendaba su uso en entornos más estables para obtener resultados más confiables.

4.2.2. Máquina virtual

En un segundo escenario, se han realizado pruebas entre la máquina local y una ejecución en una máquina virutal, en este caso el funcionamiento del protocolo ha sido el esperado, permitiendo establecer conexiones e intercambiar información sin problema alguno.

4.2.3. Red local

En el tercer escenario, se realizaron pruebas en una red local, donde el protocolo demostró un rendimiento óptimo y un funcionamiento sin problemas. Las comunicaciones entre nodos se llevaron a cabo correctamente, y se verificó que la privacidad y el anonimato se mantenían de acuerdo con los principios de la red Tor. Esta prueba en un entorno controlado permitió validar la funcionalidad del protocolo y confirmar que cumplía con los objetivos establecidos.

4.2.4. Diferentes redes

Además de las pruebas en una red local, se realizaron pruebas en redes totalmente diferentes para simular un entorno más realista. Estas pruebas también se llevaron a cabo con éxito, y el protocolo funcionó de manera efectiva en esta configuración. Sin embargo, se observó que el rendimiento del protocolo era variable, principalmente debido a la naturaleza de la red Tor utilizada para la comunicación. Los tiempos de respuesta podían variar y, en algunos casos, superar varios segundos, lo cual es un efecto esperado en redes Tor. Además, se identificó que en ocasiones la comunicación podía experimentar errores debido a la propia naturaleza de Tor.

5. Conclusiones

Durante el desarrollo de este proyecto, se han logrado alcanzar los objetivos principales establecidos, así como se han identificado metas adicionales que podrían ser exploradas en futuras etapas. A continuación, se presentan las principales conclusiones obtenidas en cada objetivo abordado.

En primer lugar, se ha llevado a cabo una exhaustiva investigación y evaluación de los desafíos técnicos y de seguridad asociados con la creación de un protocolo P2P distribuido y anónimo sobre la red Tor. La red Tor, conocida por su capacidad de proporcionar anonimato en línea, presenta desafíos únicos cuando se utiliza para servicios P2P. Sin embargo, a través del estudio y análisis de estos desafíos, se han identificado soluciones viables y se ha logrado implementar con éxito la comunicación a través de Tor en el protocolo desarrollado. Esto demuestra una comprensión sólida de los aspectos técnicos y de seguridad relacionados con la utilización de Tor en un entorno P2P.

En segundo lugar, se ha logrado diseñar e implementar un protocolo P2P distribuido y anónimo que aborda los desafíos identificados. El protocolo ha sido implementado utilizando las librerías de Tor en Python y se ha integrado con la red Tor para garantizar la privacidad y anonimato de los usuarios. Se han utilizado técnicas de cifrado de extremo a extremo, identificación de usuarios mediante pseudónimos, y mecanismos de firmado y verificación para proteger la integridad y autenticidad de los mensajes. El éxito en la implementación de estas características demuestra una sólida capacidad de diseño y desarrollo de protocolos seguros.

En tercer lugar, se ha logrado garantizar la escalabilidad del protocolo desarrollado. Se ha tenido en cuenta la capacidad de expansión y la integración en otros proyectos sin comprometer la privacidad y la seguridad de los usuarios. Esto es fundamental para permitir la adopción del protocolo a gran escala y asegurar su viabilidad y efectividad en el mundo real. La implementación de un sistema de eventos y la utilización de patrones de diseño adecuados han contribuido a la creación de un protocolo altamente escalable y modular, capaz de adaptarse a diferentes necesidades y requerimientos.

Además de los objetivos principales, se han identificado metas adicionales que podrían ampliar el alcance y el impacto del proyecto. Entre ellas se encuentra la evaluación exhaustiva de los resultados, incluyendo un estudio de rendimiento, mercado y seguridad, con el fin de determinar la viabilidad del protocolo como una alternativa efectiva en las comunicaciones cotidianas. Esta evaluación permitiría identificar posibles mejoras y ajustes necesarios para optimizar el protocolo y adaptarlo a diferentes casos de uso.

Por último, se ha reconocido la importancia de contribuir al avance del campo de seguridad y privacidad en las comunicaciones online. El proyecto ha abordado el desafío de fortalecer la protección de los usuarios en sus interacciones en línea, y se ha trabajado en la creación de soluciones que promuevan una mayor seguridad y privacidad en estas comunicaciones cotidianas. Este enfoque en la protección de los usuarios refleja un compromiso con la mejora continua y el avance de la seguridad

y privacidad en el ámbito digital.

En conclusión, este proyecto ha logrado satisfactoriamente implementar la comunicación a través de Tor en un protocolo P2P distribuido y anónimo, demostrando un sólido conocimiento de los desafíos técnicos y de seguridad involucrados. El diseño y la implementación del protocolo han sido realizados de manera efectiva, brindando un sistema escalable y altamente seguro. Aunque se han cumplido los objetivos principales, se han identificado oportunidades para futuras investigaciones y mejoras, incluyendo una evaluación exhaustiva de los resultados y una mayor contribución al campo de seguridad y privacidad en las comunicaciones online. En general, este proyecto ha sentado las bases para una alternativa segura y descentralizada en las comunicaciones P2P, fomentando la protección de los usuarios en su interacción en línea.

5.1. Ampliaciones y Trabajo Futuro

A lo largo del desarrollo de este proyecto, se han identificado varias áreas que podrían ser objeto de ampliación y trabajo futuro. Estas ampliaciones tienen como objetivo mejorar y expandir las funcionalidades del protocolo P2P distribuido y anónimo implementado, así como explorar nuevos horizontes en el campo de la seguridad y la privacidad en línea. A continuación, se detallan algunas de las posibles ampliaciones y áreas de trabajo futuro:

5.1.1. Mejora de la eficiencia y rendimiento del protocolo

Una de las áreas clave para el trabajo futuro es la mejora de la eficiencia y el rendimiento del protocolo. A medida que el protocolo se utilice en escenarios reales y se enfrente a cargas de trabajo más altas, será fundamental optimizar su desempeño. Esto puede incluir la optimización de algoritmos, la reducción del tiempo de respuesta y la minimización del consumo de recursos. Una evaluación exhaustiva de la escalabilidad y el rendimiento del protocolo en diferentes entornos y configuraciones ayudaría a identificar posibles cuellos de botella y áreas de mejora.

5.1.2. Ampliación de características y funcionalidades

El protocolo actualmente implementado proporciona una base sólida para la comunicación P2P distribuida y anónima. Sin embargo, existen diversas oportunidades para ampliar las características y funcionalidades del protocolo. Por ejemplo, se podría considerar la implementación de un sistema de búsqueda y descubrimiento de nodos en la red, lo que permitiría a los usuarios encontrar y conectarse de manera más eficiente.

5.1.3. Expansión a otros casos de uso

Si bien el protocolo ha sido desarrollado inicialmente para su uso en mensajería y compartición de archivos, existe la posibilidad de expandir su aplicación a otros casos de uso. Por ejemplo, se podría considerar su implementación en entornos de transacciones financieras, donde la privacidad y la seguridad son fundamentales. Además, se podría explorar su integración en aplicaciones de colaboración en tiempo real, como salas de chat y videoconferencias, donde la confidencialidad y la autenticidad de las comunicaciones son críticas. Estas expansiones permitirían una mayor adopción y utilidad del protocolo en diferentes contextos.

5.1.4. Estudio de mercado y viabilidad

Para determinar la viabilidad y efectividad del protocolo en el mundo real, sería necesario realizar un estudio de mercado exhaustivo. Este estudio podría incluir la evaluación de las necesidades y demandas del mercado, la identificación de competidores y la investigación de posibles modelos de negocio. Además, se podría realizar un análisis de costo-beneficio para evaluar la viabilidad económica de implementar y mantener el protocolo a gran escala. Estos estudios proporcionarían información valiosa para tomar decisiones informadas sobre el futuro desarrollo y la adopción del protocolo.

5.1.5. Investigación continua en seguridad y privacidad en línea

Dado el constante avance de las amenazas y los desafíos en el ámbito de la seguridad y la privacidad en línea, es fundamental continuar la investigación en este campo. Esto incluye estar al tanto de las últimas tendencias y tecnologías, así como participar en comunidades académicas y colaborar con expertos en la materia. Además, se podría explorar la aplicación de técnicas de aprendizaje automático y análisis de datos para mejorar la detección de actividades maliciosas y la protección de los usuarios. La investigación continua garantizará que el protocolo se mantenga actualizado y a la vanguardia en términos de seguridad y privacidad.

6. Bibliografía

Referencias

[1] James, C.: What's a Peer-to-Peer (P2P) Network? ,
https://www.computerworld.com/article/2588287/
networking-peer-to-peer-network.html

[2] Tor Project — Anonymity Online ,
https://www.torproject.org/es/about/history/

[3] What is the Tor Browser? And how it can help protect your identity,
https://www.csoonline.com/article/3287653/
what-is-the-tor-browser-how-\it-works-and-how-it-can-help-you\
-protect-your-identity-online.html

[4] Tor Hidden Services: A Systematic Literature Review,
https://www.mdpi.com/2624-800X/1/3/25

[5] Biryukov, A., Pustogarov, I., & Weinmann, R. (2013). Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization.
https://doi.org/10.1109/sp.2013.15

[6] Tor Project. (s.f.). Stem: Python controller library for Tor
https://stem.torproject.org

[7] Jonsson, J., Kaliski, B. S. (2003). Public-Key Cryptography Standards (PKCS) 1: RSA Cryptography Specifications Version 2.1. https://doi.org/
10.17487/rfc3447

[8] Legrand, A. (2020). Cryptodome: A pure-python library for cryptographic recipes.
https://www.pycryptodome.org

[9] Kaufman, C., Perlman, R., Speciner, M. (1995). Network Security: Private Communication in a Public World.
http://connections-qj.org/article/network-security-private-\
communication-public-world

[10] Meurer, M., How To Use A Proxy With Python Requests? (2021)
https://www.scrapingbee.com/blog/python-requests-proxy/

[11] Irshad, M., Adapting Behavior Driven Development (BDD) for large-scale software systems (2021)
https://doi.org/10.1016/j.jss.2021.110944

[12] Tosun, A., Muzamil, A., Turhan, B., Juristo, N., On the effectiveness of unit tests in test-driven development (2018)
https://dl.acm.org/doi/10.1145/3202710.3203153

[13] Turtschi, A., Werry, J., Hack, G., Albahari, J., Nandu, S., Meng Lee, W., Network Programming: Using TCP and UDP Protocols (2002)
https://doi.org/10.1016/B978-192899450-3/50010-X