Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

T E S I S

“Implementación de un Prototipo de Red P2P: Un Enfoque Didáctico en Cómputo y Programación Distribuida”

Presenta: Domínguez Moran Guillermo Andrés

Para obtener el grado de: Licenciatura en Ingeniería en Ciencias de la Computación

Director: Dr. Mariano Laríos Gómez

Puebla, Pue, Agosto 2025

Dedicatoria:

Para mi mama quien es mi mejor amiga y consejera ,

por siempre ser mi fuerza silenciosa,

por siempre estar a mi lado en los momentos de mayor dificultad,

por creer en mí incluso cuando yo dudaba de mi mismo,

y por no dejarme rendirme aun cuando yo sentía que ya no podía más.

Este logro es tanto tuyo como mío.

Gracias por siempre estar conmigo e impulsarme a llegar asta el final.

Gracias

Agradecimientos:

Agradezco profundamente a mi hermano y a mi padre, por su constante apoyo, consejos y por siempre estar presentes cuando más los necesité.

Su compañía y palabras me guiaron en momentos de incertidumbre y me recordaron que nunca estoy solo.

A mi profesor y asesor de tesis, Dr. Mariano Laríos Gómez, por acompañarme a lo largo de este proceso, por su orientación, sus observaciones precisas y por creer en este proyecto desde el inicio. Su guía fue fundamental para que esta tesis tomara forma y se consolidara.

A todos ellos, gracias por formar parte de este camino.

Índice General

Índice de figuras

Índice de tablas

Capitulo 1. Introducción

1.1 Resumen

El objetivo de desarrollar una red Peer to Peer (P2P) que aplique el paradigma del cómputo distribuido es facilitar la comprensión de dicho paradigma, debido a que las redes P2P son un componente esencial en la arquitectura de sistemas distribuidos. Estas redes permiten la comunicación directa entre nodos sin la necesidad de un servidor central, lo que mejora tanto la escalabilidad como la resiliencia de las aplicaciones. Una característica común del desarrollo de aplicaciones P2P es el necesario y constante intercambio de información entre cada nodo que puede formar parte del sistema en cuestión (Lavastida-López & Almeida-Cruz, 2009). Esta característica es especialmente relevante en entornos donde la centralización puede representar un cuello de botella o un punto único de falla (Larios G.,2020, p. 20).

El presente proyecto implementó y desarrolló un prototipo de red P2P que utilizó algoritmos distribuidos, con base en el protocolo de comunicación Socket UDP. Este enfoque permitió explorar aspectos clave del cómputo distribuido, como la comunicación asíncrona entre nodos y la tolerancia a fallos, los cuales son fundamentales para cualquier sistema distribuido moderno. A diferencia del modelo cliente-servidor tradicional, las redes P2P distribuyen la carga y la responsabilidad de procesamiento entre todos los nodos de la red, lo que otorga mayor flexibilidad y robustez.

Desde un punto de vista metodológico, se adoptó un enfoque de Desarrollo Dirigido por Pruebas (TDD) ya que esto permitió mantener una mayor calidad de código, haciéndolo más robusto. Además, facilito la implementación de las diferentes funcionalidades esperadas en la aplicación. También, para ayudarnos a mantener la consistencia de los datos distribuidos entre los diferentes nodos, se utilizo el patrón de diseño Replica. Tanto la metodología como el patrón de diseño se abordarán y explicaran más adelante, para comprender las razones por las cuales se consideraron como los mas adecuados para el desarrollo de este proyecto.

Una de las herramientas de software más útiles durante las pruebas de campo fue **Docker,** ya que permitió simular múltiples nodos en un entorno controlado, posibilitando de esta forma

probar la red en diferentes escenarios sin la necesidad de una infraestructura física a gran escala.

Otro propósito de este proyecto es que busca que sirva como apoyo para comprender, mediante un prototipo de red P2P funcional los temas relacionados con el computo distribuido por lo que a lo largo de este documento tratare de ser lo mas explicativo posible sin caer en tantos tecnicismos que en ocasiones obstaculizan la compresión de este tema (lo se por que yo también lo viví y se de que hablo). Espero que todo lo abordado en este proyecto te sea de utilidad o por lo menos te sea una buena referencia en la cual basarte cuando te encuentres con este tema que es bastante complejo de entender totalmente.

1.2 Antecedentes

La motivación para desarrollar este proyecto surgió de mi experiencia personal como estudiante de la Licenciatura en Ingeniería en Ciencias de la Computación. Durante los cursos que abordaban temas relacionados con el cómputo distribuido, observe que muchos de mis compañeros enfrentaban dificultades para entender este paradigma. Los conceptos como la distribución de tareas entre nodos, la sincronización de procesos y la tolerancia a fallos resultaban complejos de asimilar para gran parte del grupo. Estas dificultades a menudo se traducían en una mayor inversión de tiempo para completar ejercicios y proyectos relacionados, lo cual generaba frustraciones y abandono de la materia.

Mi objetivo por lo tanto, fue abordar esta problemática mediante el desarrollo de una red Peer to Peer (P2P), que permita ejemplificar de manera más práctica y comprensible los principios fundamentales del cómputo distribuidos. Una de las principales razones por las que me decante por una red P2P es por su naturaleza descentraliza, lo cual es un componente esencial en la arquitectura de sistemas distribuidos, y que por lo tanto para realizarla te veraz obligado a salir de esquema de cliente-servidor.

Teniendo todo esto en cuenta, este proyecto busca no solo desarrollar una solución funcional, sino también proporcionar una herramienta educativa que simplifique la enseñanza del cómputo distribuido y que ayude a los estudiantes a experimentar de primera mano este paradigma.

1.3 Objetivos del Proyecto

En el ámbito del cómputo distribuido, las redes Peer-to-Peer (P2P) han demostrado ser una arquitectura eficiente para la descentralización y escalabilidad de sistemas informáticos. Estas redes permiten la comunicación directa entre nodos sin depender de una autoridad central, lo que las haces adecuadas para aplicaciones que requieran alta disponibilidad y tolerancia a fallos (Tanenbaum & Van Steen, 2017).

Este proyecto tiene como finalidad el diseño de una aplicación de red P2P que sirva como un ejemplo y guía para facilitar la comprensión del paradigma del cómputo distribuido y sus aplicaciones en la programación.

1.3.1 Objetivo General

Desarrollar una aplicación de red Peer-to-Peer que facilite la comprensión y aplicación del cómputo distribuido y la programación distribuida, garantizando una arquitectura eficiente y escalable basada en principios fundamentales de la computación distribuida.

1.3.2 Objetivos Específicos

1. Analizar y comparar los principales algoritmos de enrutamiento utilizados en redes P2P, como Chord, Pastry y Kademlia, para seleccionar el más adecuado para la implementación en el proyecto. La selección se basará en criterios de eficiencia, robustez y escalabilidad, siguiendo estudios sobre optimización de algoritmos P2P (Maymounkov & Mazieres, 2002; Stoica et al., 2001)
2. Diseñar e implementar un prototipo funcional de red P2P utilizando el algoritmo seleccionado, siguiendo un enfoque modular que permita la flexibilidad y adaptabilidad en futuras mejoras y actualizaciones, conforme a las mejores prácticas de desarrollo en sistemas distribuidos (Liu et al., 2022).
3. Evaluar el desempeño del Prototipo en términos de escalabilidad, latencia y eficiencia, mediante la simulación de escenarios reales de computo distribuido. La evaluación se desarrollo usando métricas estándar para sistemas distribuidos, como el tiempo de respuesta y el ancho de banda, según recomendaciones en estudios de performance P2P.

4. Desarrollo material didáctico basado en el prototipo de la red P2P, con el fin de facilitar la enseñanza de los conceptos clave de cómputo y programación distribuida. El material incluye guías y ejemplos prácticos que promueven un aprendizaje activo en entornos académicos y profesionales, siguiendo los lineamientos actuales de educación en tecnologías de la información.

1.4 Metodología de desarrollo

El desarrollo de este proyecto se fundamenta en metodologías y patrones de diseño que garantizan la calidad del software y la estabilidad de la arquitectura distribuida. Para ello, se ha adoptado **Test Driven Development (TDD)** como metodología de desarrollo, con el objetivo de asegurar la funcionalidad del sistema mediante pruebas automatizadas desde las primeras etapas del proyecto. Asimismo, se ha empleado el **Patrón de Diseño Réplica,** ampliamente utilizado en sistemas distribuidos para mejorar la disponibilidad y tolerancia a fallos en entornos descentralizados.

1.4.1 Test Driven Development (TDD)

El **Desarrollo Dirigido por Pruebas (TDD, por sus siglas en inglés)** es una metodología ágil de desarrollo de software propuesta por **Kent Beck (2002),** cuyo enfoque principal es garantizar que cada parte del código cumpla con los requisitos antes de ser implementado.

El proceso de TDD sigue un ciclo estructurado conocido como **“Red-Green-Refactor”,** que se compone de tres fases principales (como se expone en la Figura 1):

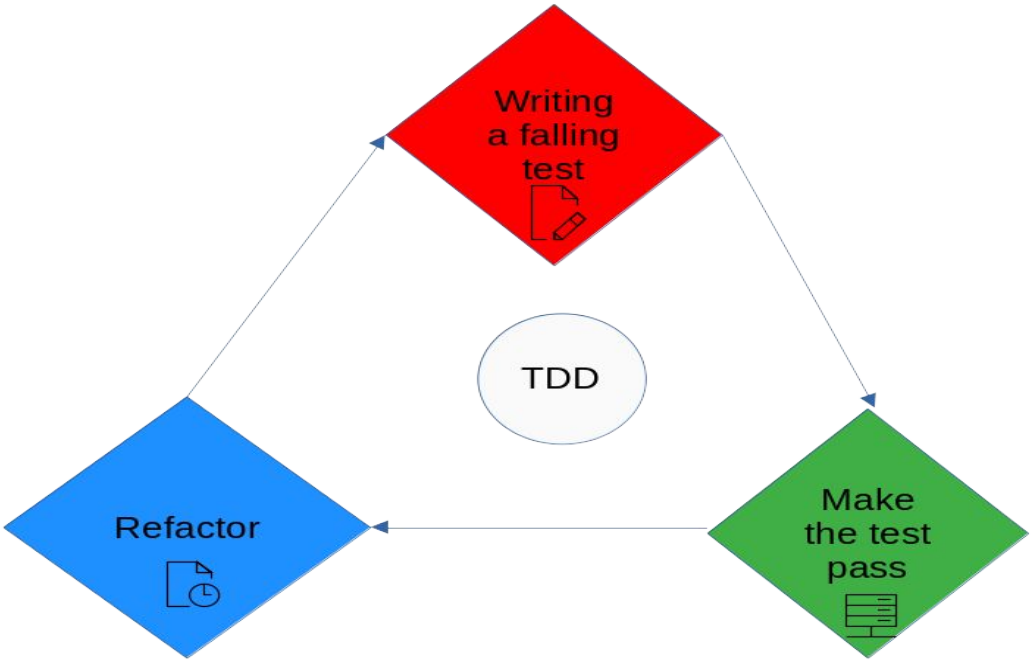

Figura 1: Diagrama del ciclo de vida de la metodología TDD

1. **Red (Rojo) –** Escribir una Prueba que falle:

    1.1. Antes de escribir cualquier funcionalidad, se crea una prueba automatizada que describe el comportamiento esperado de una función o módulo.

    1.2. En este punto, la prueba fallará porque la funcionalidad aún no ha sido implementada

2. **Green (Verde) –** Escribir el código mínimo necesario para pasar la prueba:

    2.1. Se implementa el código más básico que permite que la prueba pase satisfactoriamente.

    2.2. En esta etapa, el objetivo no es escribir el código más eficiente o elegante, sino hacer que la prueba pase lo antes posible.

3. **Refactor (Refactorizar) –** Mejorar el código manteniendo su funcionalidad:

    3.1. Se optimiza el código eliminando redundancia, mejorando su estructura y asegurando que mantenga una buena calidad sin cambiar su comportamiento.

3.2. Se puede realizar una reestructuración del código para mejorar la mantenibilidad y escalabilidad.

Como puede observarse en la *Figura 1* esta metodología es cíclica por lo tanto permite permite que en caso de encontrar un fallo este pueda ser tratado de inmediato y repetir el proceso con lo ya aprendido durante la prueba anterior, de esta manera se pretende que el desarrollo llegue a ser más rápido, ya que el enfoque del TDD pretende eliminar la duplicación del código y dejar que los desarrolladores sólo escriban código nuevo, en caso de que una prueba fallará. Por lo que a continuación presentare todos los beneficios de esta metodología de trabajo:

- **Mayor calidad del código:** Al escribir pruebas antes del desarrollo, se garantiza que el código cumpla con los requisitos desde el principio.
- **Menor número de errores:** Se detectan los problemas en una etapa temprana, lo que garantiza una reducción de correcciones en etapas futuras del proyecto.
- **Mejor diseño y modularidad:** Se promueve la escritura del código desacoplado y fácil de mantener, ya que cada componente debe poder ser probado de manera independiente.
- **Desarrollo interactivo y seguro:** Con cada nueva funcionalidad añadida, se asegura que las anteriores siguen funcionando correctamente mediante la ejecución de pruebas automatizadas.

Aplicación de TDD en el Proyecto

En el desarrollo de la red Peer-to-Peer se empleo el TDD para asegurar que cada módulo funcional pase por pruebas automatizadas antes de su implementación definitiva. Por ejemplo, se desarrollaron pruebas unitarias para verificar la correcta comunicación entre nodos, la propagación de mensajes en la red y la actualización de réplicas en distintos escenarios.

1.4.2 Patrón de diseño Réplica

En sistemas distribuidos, la replicación es una técnica fundamental que consiste en mantener copias de datos o servicios en múltiples nodos dentro de la red. Este enfoque tiene como objetivo principal mejorar la disponibilidad, tolerancia a fallos y el rendimiento del sistema. En este proyecto, su implementación permite que múltiples nodos en la red P2P mantengan

copias de ciertos datos, asegurando que la información siga disponible incluso si algunos nodos fallan o se desconectan.

Las características principales de este patrón son:

- **Redundancia de datos:** Cada nodo almacena copias de datos críticos para prevenir pérdidas de información.
- **Sincronización entre réplicas:** Se emplean mecanismos para garantizar que todas las réplicas de un mismo dato permanezcan actualizadas y consistentes.
- **Mejora en la escalabilidad:** Al distribuir la carga entre múltiples nodos con datos replicados, se reduce la congestión y se mejora el rendimiento del sistema.

Este patrón ha sido ampliamente adoptado en arquitecturas de almacenamiento distribuido, bases de datos y sistemas de alta disponibilidad, proporcionando un marco sólido para la implementación de este proyecto (Tanenbaum & Van Steen, 2017).

Principales Ventajas de la Replicación:

- **Disponibilidad Mejorada:** Al tener múltiples copias de los datos o servicios, el sistemas puede continuar operando incluso si uno o varios nodos fallan, ya que las solicitudes pueden ser atendidas por las réplicas disponibles.
- **Tolerancia a Fallos:** La replicación permite que el sistema sea resiliente ante fallos de hardware o software, ya que las réplicas pueden asumir la carga de trabajo de los nodos que experimentan problemas.
- **Optimización del Rendimiento:** Al distribuir las solicitudes entre varias réplicas, se puede reducir la carga en nodos individuales y disminuir la latencia de acceso, especialmente si las réplicas están ubicadas geográficamente cerca de los usuarios.

Desafíos Asociados a la Replicación:

- **Consistencia de Datos:** Mantener la coherencia entre múltiples réplicas es un desafío, especialmente en sistemas de gran escala. Es crucial definir políticas de consistencia que equilibren las necesidades de rendimiento y la precisión de los datos.
- **Gestión de Conflictos:** En escenarios de replicación asíncrona, pueden surgir conflictos cuando diferentes réplicas reciben actualizaciones concurrentes. Es

necesario implementar estrategias para detectar y resolver estos conflictos de manera efectiva.

- **Sobrecarga de Actualización:** La replicación síncrona puede introducir una sobrecarga significativa debido a la necesidad de coordinar y confirmar las actualizaciones en todas las réplicas, lo que puede afectar el rendimiento del sistema.

Tipos de Replicación

1. **Replicación Síncrona:** En este enfoque, todas las réplicas se actualizan simultáneamente con cada cambio en los datos. Esto asegura una consistencia fuerte, ya que todas las réplicas reflejan el mismo estado en todo momento. Sin embargo, puede introducir latencia adicional debido a la necesidad de confirmar las actualizaciones en todas las réplicas antes de completar una operación.

   a) **Alta consistencia:** Todas las réplicas tienen el mismo estado en todo momento.

   b) **Mayor latencia:** Como la actualización debe propagarse a todas las copias antes de confirmar la operación, el sistema puede volverse más lento.

   c) **Menor riesgo de pérdida de datos:** Si un nodo falla, los datos ya han sido replicados en otros nodos.

   d) **Mayor carga en la red:** Se requiere más comunicación entre los nodos para coordinar las actualización.

2. **Replicación Asíncrona:** Las actualizaciones se propagan a las réplicas de manera diferida, lo que puede resultar en inconsistencias temporales entre las réplicas. Este método ofrece mejor rendimiento y menor latencia, pero requiere mecanismos para resolver posibles conflictos y asegurar la eventual consistencia de los datos.

   a) **Mayor velocidad y rendimiento:** Las operaciones no se bloquean esperando que todas las réplicas se actualicen.

   b) **Menor carga en la red:** Se reduce la cantidad de comunicación inmediata entre nodos.

   c) **Posibles inconsistencias temporales:** Durante un tiempo, algunas réplicas pueden estar desactualizadas hasta que reciben las nuevas actualizaciones.

d) **Mayor disponibilidad:** Si un nodo cae, las operaciones pueden continuar en los otros nodos, aunque con datos ligeramente desactualizadas.

Aplicación en el Proyecto

En el contexto de este proyecto, la implementación del Patrón de Diseño Réplica en el red Peer-to-Peer (P2P) es esencial para garantizar que los datos estén disponibles incluso en caso de fallos de nodos individuales. Al replicar datos clave en múltiples nodos, se asegura que la información permanezca accesible y consistente, mejorando la resiliencia y confiabilidad del sistema.

Además, la replicación contribuye a la escalabilidad del sistema, ya que permite distribuir la carga de trabajo entre varios nodos, optimizando el uso de recursos y mejorando la capacidad de respuesta ante un número creciente de usuarios o solicitudes.

1.5 Infraestructura

Dado que este proyecto se centra en el desarrollo de un prototipo de red P2P para la comprensión del cómputo distribuido, la infraestructura utilizada se enfocó principalmente en el software. Sin embargo, también se especifica el hardware empleado para la implementación y prueba del sistema.

1.5.1 Hardware

Para el desarrollo y simulación del prototipo, se utilizó un equipo con las siguientes características:

- **Modelo:** HP Notebook 15-15ay008la
- **Memoria RAM:** 8gb
- **Procesador:** Intel Pentium N3710 (4 núcleos)
- **Almacenamiento:** SSD de 250 GB
- **Sistem Operativo:** Debian GNU/Linux 12

1.5.2 Software

El software empleado en el desarrollo del proyecto incluyó herramientas para la programación, pruebas, simulación de nodos, control de versiones y gestión del proyecto.

- **Entorno de Desarrollo**

  → **IntelliJ IDEA:** Entorno de desarrollo integrado (IDE) para Java que ofrece herramientas avanzadas de refactorización, depuración y desarrollo ágil.

  → **Java:** Lenguaje de programación utilizado para la implementación del prototipo, conocido por su portabilidad y su uso extendido en aplicaciones distribuidas.

- **Simulación y Contenedores**

  → **Docker:** Herramienta utilizada para emular múltiples nodos dentro de una misma máquina. Cada nodo se ejecutó como un contenedor Docker, lo que permitió probar la red P2P en un entorno controlado.

- **Control de Versiones y Colaboración**

  → **GitHub:** Plataforma utilizada para el control de versiones y almacenamiento del código fuente del proyecto, facilitando la trazabilidad y colaboración.

- **Documentación y Productividad**

  → **LibreOffice:** Suite ofimática utilizada para la redacción y edición de documentos del proyecto.

  → **Trello:** Herramienta de gestión de proyectos basada en tableros y tarjetas, utilizada para organizar tareas y realizar el seguimiento del progreso.

Capitulo 2. Estado del Arte

El desarrollo de una red Peer-to-Peer (P2P) basada en los principios del cómputo distribuido requiere una comprensión profunda de los fundamentos teóricos que sustentan este paradigma. En este capítulo, se presentarán los conceptos claves explicados de la forma mas clara y sencilla para que de esta forma se facilite su rápida comprensión para esto mismo se adjuntan varias imágenes que ayudarán a una mejor visualización de estos conceptos.

En primer lugar, se aborda el cómputo distribuido, su importancia en la actualidad y como las arquitecturas descentralizadas han evolucionado para mejorar la escalabilidad y tolerancia a fallos en los sistemas. Posteriormente, se estudian las redes P2P, sus tipos y características, destacando su papel en la comunicación eficiente entre nodos sin la necesidad de un servidor centralizado.

Además, se describen algunos algoritmos distribuidos tales como Bully y Anillo, describiendo sus principales ventajas y desventajas en términos de eficiencia y escalabilidad. También se aborda el protocolo UDP el cual fue empleado a la hora del envió de mensajes, archivos, etc.

Finalmente abordaremos uno de los temas mas interesantes con respecto a las posibilidades de las redes P2P ya que en este proyecto se incluyo la integración de un algoritmo de consenso el cual es este caso fue **POW (Proof of Work)** este algoritmo y todas sus implicaciones las abordaremos mas adelante.

Este capítulo servirá como base teórica para la implementación del prototipo, proporcionando los conocimientos necesarios para justificar las decisiones de diseño y desarrollo tomadas en el proyecto.

2.1 Paradigma del Computo Distribuido

Antes de entrar en detalle sobre el paradigma del computo distribuido primero me gustaría explicar que es en sí un paradigma para la cual me gustaría citar las palabras del profesor Mariano Larios G., (2009) “*Un paradigma es un patrón o modelo que se emplea en el estudio de cualquier tema de gran complejidad en lo cual es usado para identificar patrones o modelos básicos y asi poder clasificarlos de acuerdo a ellos*”.

En términos más simples podemos decir que un paradigma es un marco de referencia (instrucciones) o una estructura conceptual que ayuda a organizar y entender la complejidad de ciertos fenómenos. Al tener un paradigma, los investigadores y académicos pueden analizar y categorizar diferentes aspectos de un tema, facilitando así su comprensión y estudio.

En el contexto del cómputo distribuido es un paradigma de procesamiento en el que varias computadoras interconectadas colaboran para resolver un problema (como se muestra en la Figura 2), en lugar de depender de una única máquina centralizada. Según Tanenbaum y Van Steen (2017), un sistema distribuido es "*Una colección de computadoras independientes que parecen un sistema único para sus usuarios".* Este enfoque permite dividir una tarea compleja en partes más pequeñas que se ejecutan simultáneamente en diferentes nodos de la red, optimizando el tiempo de procesamiento y el uso de recursos.

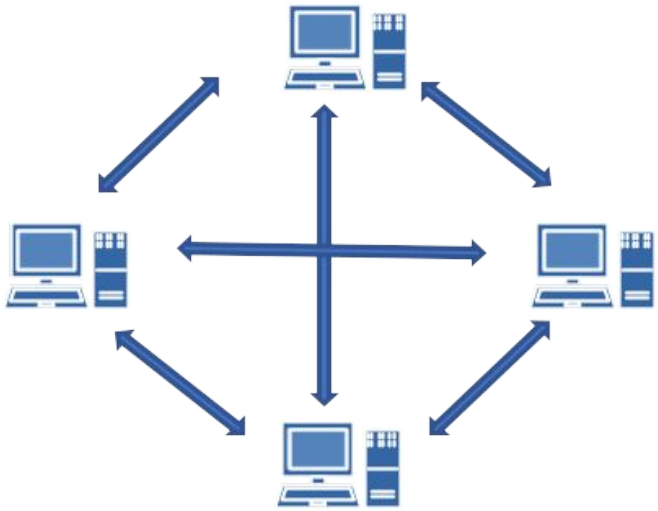

Figura 2: Sistema distribuido donde varios nodos colaboran en la ejecución de una tarea.

2.1.1 Características del Computo Distribuido

El paradigma del computo distribuido se basa en las siguientes reglas fundamentales que marcan las características que debe llevar un sistema o aplicación que se pueda considerar como "Computo Distribuido":

1. **Descentralizado:** No hay una autoridad central que controle todo el sistema.
2. **P2P (Peer to Peer):** Los nodos pueden actuar tanto como clientes como servidores.

3. **Paso de mensajes:** Los nodos se comunican intercambiando mensajes en lugar de compartir memoria.
4. **Objetos distribuidos**: Un objeto puede existir en diferentes nodos y ser accedido de manera remota.
5. **Agentes móviles:** Unidades de código que pueden desplazarse entre nodos para ejecutar tareas.
6. **Anónimo:** Los nodos pueden entrar y salir de la red dinámicamente sin afectar el sistema.
7. **Colaborativo:** Los recursos se comparten para optimizar la eficiencia.

Esto hace que el computo distribuido sea ampliamente utilizado en varios campos, incluyendo la ciencia de datos, la inteligencia artificial, investigación científica, análisis financiero, telecomunicaciones y la medicina. Su capacidad para procesar grandes volúmenes de datos de manera eficiente y su tolerancia a fallos lo convierten en una herramienta indispensable para organizaciones que buscan optimizar el rendimiento y la escalabilidad de sus sistemas.

2.1.2 Ventajas y Desafíos

Como todo en este mundo el computo distribuido ofrece muchas ventajas que lo hacen ideal para una gran cantidad de aplicaciones y entornos, pero también tiene muchos desafíos que deben abordarse cuidadosamente para garantizar un correcto funcionamiento de tu sistema o aplicación donde lo estés implementando.

A continuación en la Tabla 1 se mostraran algunas de estas ventajas y desafíos:

Tabla 1: Comparación de ventajas y desafíos del computo distribuido.

Ventajas	Desafíos
Escalabilidad: El computo distribuido permite aumentar la capacidad de procesamiento añadiendo más nodos al sistema, lo que mejora el rendimiento y permite manejar grandes volúmenes de datos y tareas.	**Complejidad:** El diseño, implementación y mantenimiento de sistemas distribuidos son más complejos que los sistemas centralizados debido a la necesidad de coordinar y sincronizar múltiples nodos.
Tolerancia a fallos: Al distribuir las tareas y	**Seguridad:** La seguridad de los datos y la

los datos entre múltiples nodos, el sistema puede seguir funcionando incluso si algunos nodos fallan. Esto aumenta la resiliencia y la disponibilidad del sistema.	comunicación entre nodos es un desafío importante, ya que los sistemas distribuidos son más susceptibles a ataques y vulnerabilidades.
Flexibilidad: Los sistemas distribuidos pueden adaptarse y escalarse fácilmente según las necesidades cambiantes, lo que los hace ideales para entornos dinámicos y de rápido crecimiento.	**Consistencia:** Garantizar la consistencia de los datos en todos los nodos puede ser difícil, especialmente en sistemas con alta latencia o en entornos con nodos que pueden desconectarse y reconectarse dinámicamente.
Reducción de Costos: En lugar de depender de un único servidor potente y costoso, se pueden utilizar múltiples servidores más pequeños y económicos, lo que reduce los costos de infraestructura.	**Coordinación y sincronización:** La coordinación y sincronización entre nodos pueden ser complicadas y requieren algoritmos y protocolos eficientes para garantizar la coherencia y el funcionamiento correcto del sistema.
Mejora del rendimiento: Al distribuir las tareas en varios nodos, se pueden realizar cálculos en paralelo, lo que reduce el tiempo de procesamiento total y mejora la eficiencia.	**Rendimiento de la red:** La comunicación entre nodos a través de la red puede introducir latencia y afectar el rendimiento del sistema, especialmente en aplicaciones sensibles al tiempo
Localización geográfica: Los sistemas distribuidos pueden estar ubicados en diferentes nodos regiones geográficas, lo que mejora el acceso y la disponibilidad de los datos para los usuarios en diferentes ubicaciones.	**Mantenimiento y gestión:** El mantenimiento y la gestión de sistemas distribuidos puede ser mas laborioso y requerir habilidades especializadas, ya que hay más componentes y nodos que monitorizar y administrar.

En resumen, la computación distribuida proporciona una serie de ventajas que la hacen ideal para manejar grandes volúmenes de datos y tareas de manera eficiente. No obstante, estos beneficios vienen acompañados de desafíos significativos que requieren una planificación y gestión cuidadosa. Esto se debe principalmente a que los entornos tecnológicos están en constante evolución por lo que requieren de sistemas que pueden evolucionar con ellos para lo cual la computación distribuida es clave, haciendo que superar los desafíos que esta presenta sea clave, maximizando de esta la capacidad de respuesta y la eficiencia operativa.

2.1.3 Arquitecturas de Computación Distribuida

El cómputo distribuido se organiza en diversas arquitecturas, según la distribución de sus nodos o recursos. Cada una de estas arquitecturas tiene sus propias ventajas y desafíos y se eligen en función de los requisitos específicos del sistema y las aplicaciones. Existes varias arquitecturas sin embargo solo voy a mencionar las 3 principales y mas conocidas.

Cliente – Servidor

En esta arquitectura, un nodo actúa como servidor central que administra y proporciona recursos, mientras que otros nodos, conocidos como clientes, solicitan estos recursos. Es una configuración común en aplicaciones web, donde el servidor aloja el sitio web y los clientes acceden a el, por medio de navegadores. A continuación en la Tabla 2 se verán sus ventajas y desventajas.

Tabla 2: Ventajas y desventajas de la arquitectura Cliente - Servidor

Ventajas	Desventajas
Centralización del control: El servidor centralizado facilita la administración, la seguridad y el mantenimiento del sistema.	**Punto único de falla:** Si el servidor central falla, todos los clientes se verán afectados.
Eficiencia en la gestión de recursos: La centralización permite una asignación de recursos más eficiente.	**Escalabilidad limitadas:** A medida que aumenta el número de clientes, el servidor puede llegar a su limite de capacidad.

P2P (Peer – to – Peer)

El la arquitectura P2P, todos los nodos tienen roles equivalentes y pueden actuar como clientes y servidores simultáneamente. Esta arquitectura se utiliza en aplicaciones como redes de intercambio de archivos (un ejemplo de esto sería BitTorrent) y algunas plataformas de comunicación.

Tabla 3: Ventajas y Desventajas de la arquitectura P2P

Ventajas	Desventajas
Descentralización: No hay un punto único de falla, lo que mejora la robustez del sistema.	**Gestión complejas:** La descentralización puede complicar la administración y coordinación de los nodos.

Escalabilidad: La red puede crecer fácilmente añadiendo más nodos sin sobrecargar un servidor central.	**Seguridad:** Es más difícil garantizar la seguridad y la integración de los datos en una red P2P.

Como se pudo observar en la Tabla 3 esta arquitectura tiene sus propias desventajas y ventajas las cuales serán abordadas a mayor profundidad mas adelante en el documento.

Sin embargo a continuación se mostrara una imagen (véase Figura 3) en la cual se compara la arquitectura Cliente – Servidor y P2P.

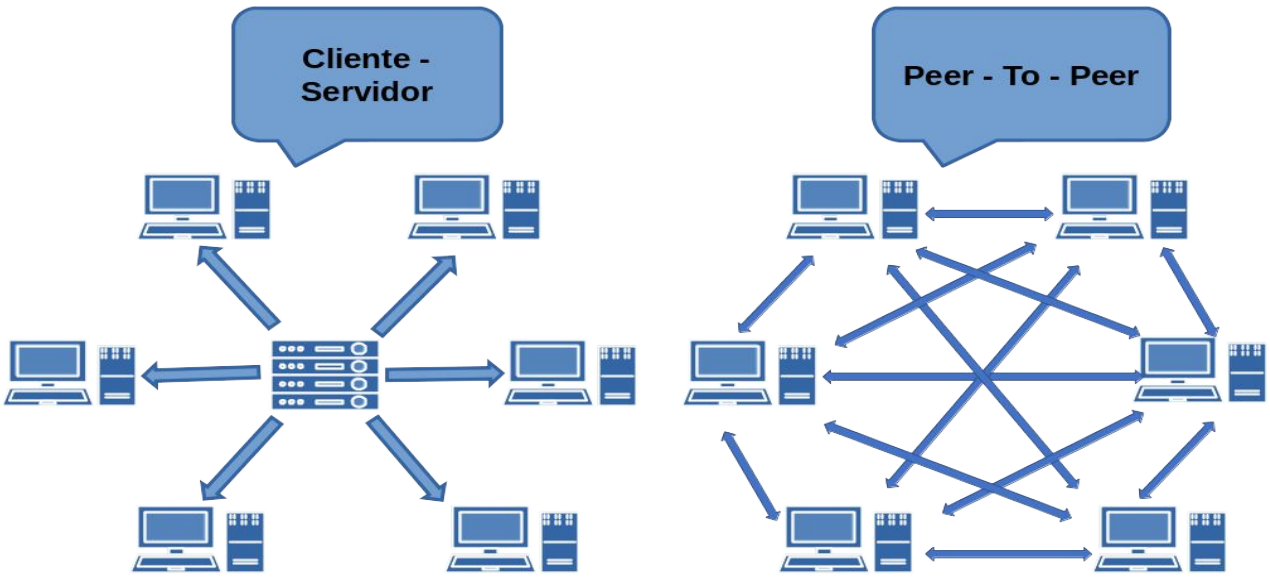

Figura 3: Comparación entre arquitectura "Cliente - Servidor" y "Peer - To – Peer"

Arquitectura híbrida

La arquitectura híbrida combina elementos de las arquitecturas cliente-servidor y P2P para optimizar la eficiencia y la escalabilidad. En esta configuración, algunas funciones son manejadas por un servidor central, mientras que otras son distribuidas entre nodos P2P. Es común en servicios de contenido como YouTube, donde los vídeos se sirven desde servidores centrales, pero también se puede utilizar el caché de los usuarios para optimizar la distribución.

Y de igual forma abajo en la Tabla 4 se muestran sus ventajas y desventajas.

Tabla 4: Ventajas y desventajas de la arquitectura híbrida.

Ventajas	Desventajas
Equilibrio entre control y robustez: Combina la administración centralizada con la robustez de la descentralización.	**Complejidad:** La implementación y gestión de una arquitectura debido a la combinación de diferentes enfoques.
Optimización de recursos: Permite una utilización más eficiente de los recursos distribuidos.	**Costos:** Puede requerir más recursos y costos iniciales para implementar y mantener.

En última instancia, la elección de la arquitectura depende de las necesidades especificas del sistemas y de las aplicaciones en cuestión. Cada enfoque ofrece soluciones diferentes para optimizar el rendimiento, resiliencia y la administración de los recursos distribuidos.

2.2 Redes Peer – to – Peer (P2P)

Las redes Peer – to – Peer (P2P) han revolucionado la forma en que los sistemas distribuidos gestionan el intercambio de información. A diferencia de los modelos cliente-servidor tradicionales, en una red P2P todos los nodos actúan tanto como clientes como servidores, permitiendo una comunicación descentralizada y escalable (véase la Figura 4 de la siguiente pagina). Este enfoque es ampliamente utilizado en aplicaciones de compartición de archivos, redes de contenido distribuido y sistemas de computación colaborativa.

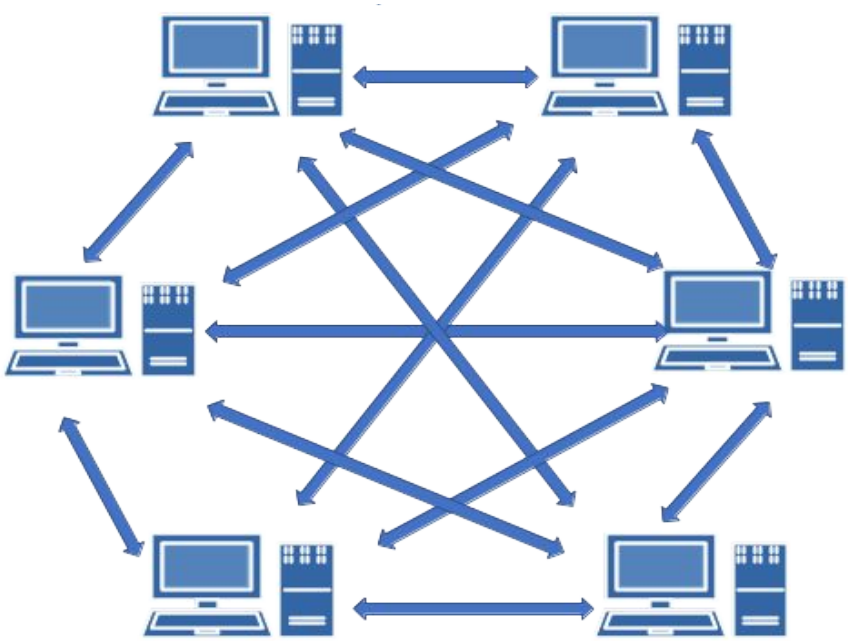

Figura 4: Diagrama de una red Peer-to-Peer

2.2.1 Características de las Redes

Las redes P2P comparten características con el computo distribuido debido a que uno depende del otro como se podrá observar a continuación:

1. **Descentralización:** En una red P2P, no existe un servidor central que controle todas las operaciones. En su lugar, cada nodo actúa tanto como cliente como servidor, compartiendo recursos directamente con otros nodos.
2. **Escalabilidad:** Las redes P2P pueden crecer fácilmente al añadir nuevos nodos ya que cada nuevo nodo contribuye sus recursos adicionales a la red.
3. **Robustez:** La ausencia de un punto único de falla hace que las redes P2P sean más resistentes a fallos y ataques.
4. **Compartición de Recursos:** Cada nodo en una red P2P puede compartir sus propios recursos, como ancho de banda, almacenamiento y datos, directamente con otros nodos.

Gracias a estas características las redes P2P son una de las opciones más atractivas para muchas aplicaciones, especialmente para aquellas que requieren alta robustez, escalabilidad y una utilización eficiente de los recursos distribuidos.

2.2.2 Ventajas y Desafíos de las Redes P2P

Tabla 5: Ventajas y Desafíos de las Redes P2P

Ventajas	Desafíos
Descentralización: La transferencia de datos se distribuye entre los nodos de la red, reduciendo la carga en un servidor central.	**Seguridad:** La falta de control centralizado puede dificultar la implementación de medidas de seguridad y aumentar el riesgo de ataques.
Escalabilidad: Las redes P2P pueden manejar un gran número de nodos sin requerir infraestructura adicional.	**Gestión de Recursos:** El uso de recursos (como ancho de banda y almacenamiento) es más difícil de gestionar debido a la naturaleza descentralizada.
Redundancia: La información suele estar replicada en múltiples nodos, lo que mejora la disponibilidad y la recuperación de datos en caso de fallos.	**Consistencia de Datos:** Asegurar que todos los nodos tienen la misma versión de los datos puede ser complicado, especialmente en redes muy grandes.

Eficiencia de Costos: Al no depender de servidores centrales, se reducen los costos de infraestructura y mantenimiento.	**Desempeño Variable:** El rendimiento puede ser inconsistente ya que depende de los recursos y la disponibilidad de los nodos participantes.
Mayor Autonomía: Los usuarios tienen más control sobre sus datos y pueden compartir recursos directamente sin intermediarios.	**Dificultad de Supervisión:** La monitorización y el mantenimiento de la red pueden ser más complejos sin una autoridad central.

Teniendo en cuenta todo lo anteriormente mencionado podemos decir que las redes P2P han demostrado ser una herramienta versátil y poderosa en el ámbito de la tecnología y las comunicación. Pero que sin embargo viene acompañado con desafíos (mírese la Tabla 5) de entre los cuales la seguridad es uno de los mas críticos ya que al carecer de un servidor central es difícil implementar medidas de seguridad efectivas; por otro lado esta misma debilidad es del mismo modo su mas grande fortaleza ya que su falta de un servidor central la hace inmune a que todo el servidor se caiga ante un punto de falla, debido a que la carga se distribuye de manera uniforme entre todos los nodos participantes.

Esta capacidad de ofrecer un equilibrio entre ventajas y desventajas , su capacidad de ofrecer soluciones descentralizadas, escalables, y rentables es la razón por la cual es la mejor opción a la hora de demostrar todas las capacidades del computo distribuido, además también de que la vuelve atractiva para implementarse en diferentes aplicaciones.

2.3 Algoritmos y Protocolos Utilizados en la Red P2P

Como se vio en el capitulo anterior, para la correcta implementación de una red P2P es necesario superar varios desafíos. Por lo que para lograr esto fue necesario emplear varios algoritmos distribuidos y protocolos que nos ayudara en esto. Dichos algoritmos y protocolos se estudiaran a continuación.

2.3.1 Protocolo de Comunicación: UDP y Sockets

Para poder trasmitir los datos entre los distintos nodos de la red P2P se implemento el protocolo **User Datagram Protocol (UDP) ,** el cual es un protocolo de comunicación que permite enviar datos a través de las redes pero sin pedir verificación si llegan correctamente; básicamente seria como enviar una carta sin pedir confirmación si dicha carta llego a su destinatario. La razón por la que se lo empleó para el desarrollo de la red es por que a

diferencia de TCP, no requiere establecer una conexión persistente lo cual reduce la latencia, permitiendo a su vez una comunicación rápida y sin sobrecarga, lo cual como ya vimos en la sección anterior es uno de los desafíos que hay que superar para el correcto funcionamiento de nuestra red P2P.

Características:

- **Baja latencia:** Como UDP no requiere establecer una conexión previa (a diferencia de TCP), los paquetes se envían y reciben más rápido.
- **Sin garantía de entrega:** No hay verificación de que un mensaje llegue correctamente, lo que puede ser un problema en ciertas aplicaciones.
- **Uso en sistemas distribuidos:** Se usa en aplicaciones donde la velocidad es más importante que la confiabilidad, como VoIP, streaming y sistemas P2P.
- **Ligero y Simple:** UDP tiene una sobrecarga mínima en comparación con otros protocolos debido a su simplicidad.
- **Multiplexación:** Utiliza puertos para dirigir paquetes de datos a aplicaciones especificas en un dispositivo, permitiendo que múltiples procesos se comuniquen simultáneamente.

Uso de Sockets en UDP

En la programación de redes, un socket UDP es un tipo de socket que utiliza este mismo protocolo para trasmisión de datos. Como ya se dijo anteriormente una de las diferencias principales con TCP es que los de UDP son de una naturaleza no orientada a conexión como se puede observar en la Figura 5. Esta característica lo hace adecuado para aplicaciones que requieran comunicaciones rápidas y eficientes, aunque a costa de no garantizar la entrega de los datos.

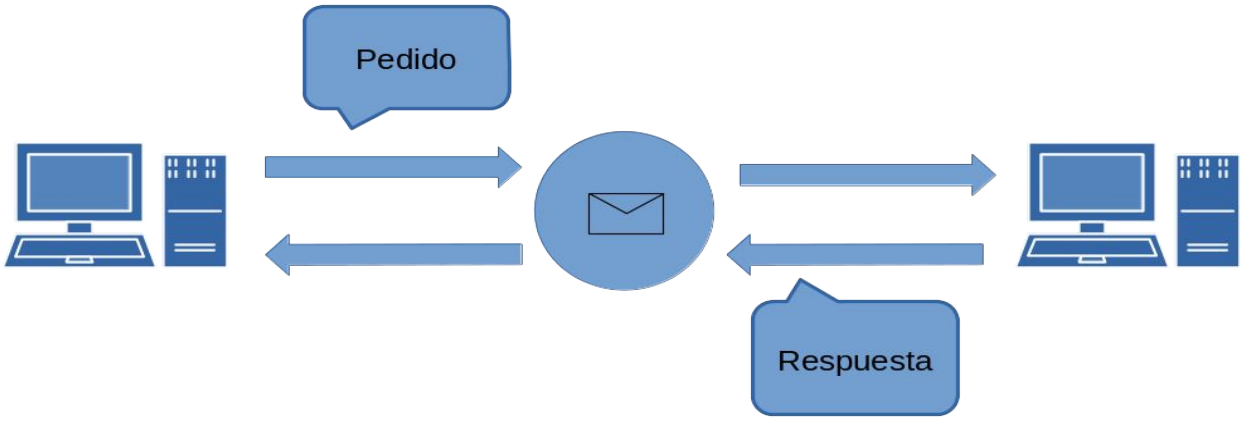

Figura 5: Diagrama de Protocolo UDP entre dos nodos

2.3.2 Algoritmos Distribuidos: Bully y Anillo

Para coordinar ciertas operaciones dentro de la red, se emplearon dos algoritmos de elección de líder entre los nodos de la red. Este coordinador puede ser responsable de tareas específicas, como la gestión de recursos o la coordinación de actividades en la red y en caso de que dicho coordinar se desconecte los algoritmos se activarían y seleccionarían uno nuevo, basándose en ciertos parámetros.

Algoritmo Bully

Es un proceso de elección de líder en el que el nodo con el identificador mas alto se convierte en el coordinador. A continuación explicare como funciona y los pasos que toma:

1. **Inicio de la Elección:** Cuando un nodo detecta que el coordinador actual no está respondiendo, inicia una elección.
2. **Mensaje de Elección:** El nodo que inicia la elección envía un mensaje de “Elección” a todos los nodos con un identificador mayor que el suyo.
3. **Respuesta de Candidatos:** Los nodos con identificadores mayores responden y a su vez inician sus propias elecciones.
4. **Declaración de Ganador:** El nodo con el identificador más alto eventualmente se declara a sí mismo como coordinador y notifica a todos los demás nodos.
5. **Aceptación del Coordinador:** Todos los nodos aceptan el nuevo coordinador.

Este algoritmo garantiza que el nodo con mayor identificador siempre sea el líder, lo que facilita la toma de decisiones en la red.

Algoritmo de Anillo

Siendo nuestra segunda opción a la hora de seleccionar un líder, funciona de una manera diferente ya que en lugar de buscar al nodo con el identificador mas alto; organiza los nodos en un anillo lógico y utiliza mensajes de “Elección” que circular alrededor del anillo para seleccionar un líder como podrán ver a continuación en sus pasos y en la Figura 6.

1. **Formación del Anillo:** Los nodos forman un anillo lógico en el que cada nodo conoce a su sucesor.
2. **Inicio de la Elección:** Cuando un nodo detecta que el coordinador no está respondiendo, inicia una elección enviando un mensaje de “Elección” a su sucesor.
3. **Paso de Mensajes:** Cada nodo añade su identificador al mensaje de elección y la pasa a su sucesor.
4. **Elección del Ganador:** Eventualmente, el mensaje de elección regresa al nodo iniciador, que selecciona al nodo con el identificador más alto como el nuevo coordinador.
5. **Anuncio del Coordinador:** El nodo iniciador envía un mensaje de “Coordinador” alrededor del anillo para informar a todos los nodos del nuevo líder.

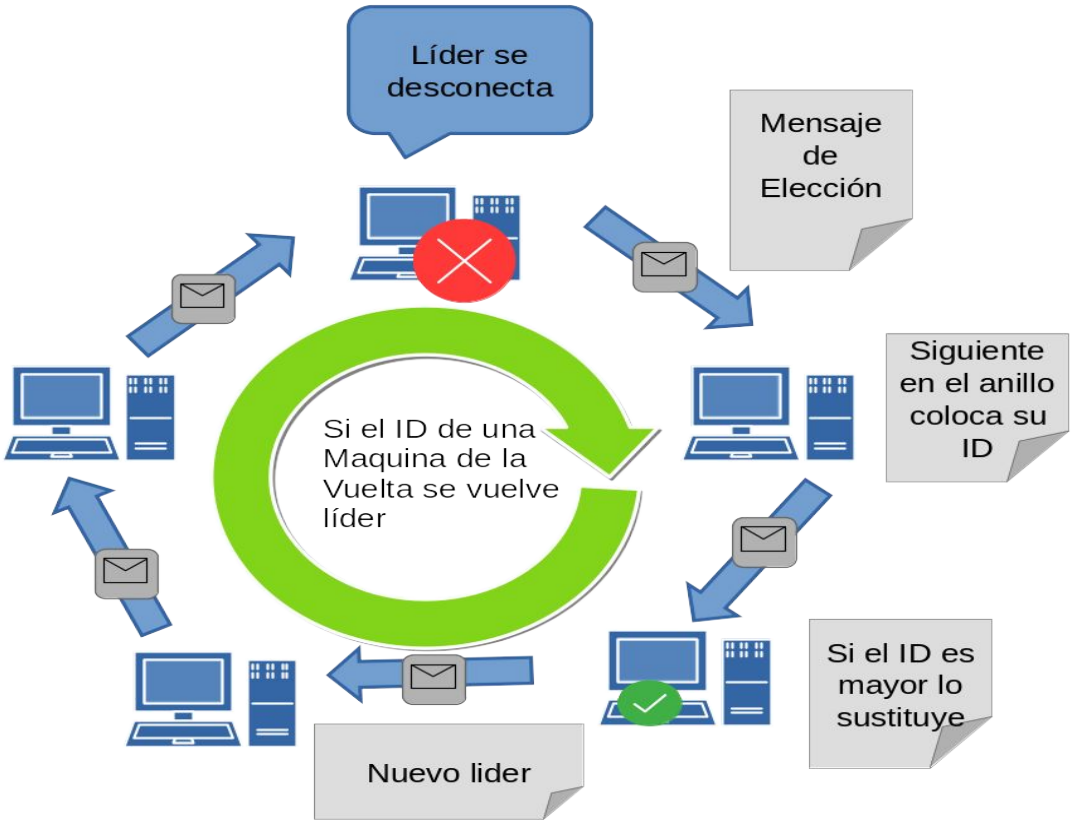

Figura 6: Representación del Flujo de Vida del Algoritmo de Anillo

Este algoritmo resulto ser mas útil cuando la cantidad de nodos era mas grande que en pruebas anteriores ya que asegura que todos los nodos participen en la elección de un nuevo líder.

2.3.3 Algoritmo de Consenso: Proof of Work (PoW)

Para garantizar la validación de ciertas operaciones en la red P2P, se implementó el algoritmo Proof of Work (PoW). Este mecanismo de consenso es fundamental para asegurar la integridad y la seguridad de las transacciones en redes descentralizas.

¿Qué es PoW?

Proof of Work (PoW) es un mecanismo de consenso utilizado en redes blockchain, como Bitcoin, que tiene como objetivo garantizar la seguridad y la integridad de las transacciones. En un sistema PoW, los participantes de la red, conocidos como mineros, compiten para resolver complejos problemas matemáticos utilizando poder computacional. Este proceso de resolución de problemas matemáticos se llama “minería”.

Una vez que un minero resuelve el problema, la solución es verificada por otros mineros en la red. Si la solución es correcta el bloque de transacciones correspondientes se agregan a la

blockchain y se registra de manera inmutable. El primer minero en resolver el problema recibe una recompensa en criptomonedas como incentivo por su esfuerzo y por proporcionar seguridad a la red.

PoW se caracteriza por su alta demanda de recursos computacionales y energía, lo que hace que sea difícil y costos para cualquier individuo o grupo atacar la red.

Pasos del Algoritmo Proof of Work (PoW)

1. Inicialización:
    - Un nodo en la red quiere agregar un nuevo bloque de información (puede ser una transacción, un conjunto de datos, etc.).
    - Se define un problema criptográfico que debe resolverse antes de aceptar el bloque.
2. Definición del Desafío:
    - Se genera un **hash objetivo,** que es un número que debe cumplir ciertas condiciones.
    - El nodo minero debe encontrar un **nonce** (un número aleatorio) que al ser combinado con los datos del bloque y pasado por una función hash (en nuestro caso se uso SHA-256), genere un resultado menor o igual al hash objetivo.
3. Prueba y Error (Minado):
    - Los nodos participantes comienzan a probar diferentes valores de **nonce** hasta encontrar uno que cumpla con la condición.
    - Esto requiere una gran cantidad de cálculos.
4. Verificación:
    - Cuando un nodo encuentra un **nonce** válido, difunde la solución a la red P2P.
    - Los demás nodos verifican la solución aplicando la función hash y comprobando si el resultado es correcto.
5. Aprobación y Adición al Registro:
    - Si la solución es válida, el bloque se agrega a la cadena y todos los nodos lo aceptan como parte del historial de transacciones o datos de la red.

6. Reinicio del Proceso:
    - Se genera un nuevo bloque y el proceso comienza nuevamente como se puede observar en el diagrama siguiente de la Figura 7.

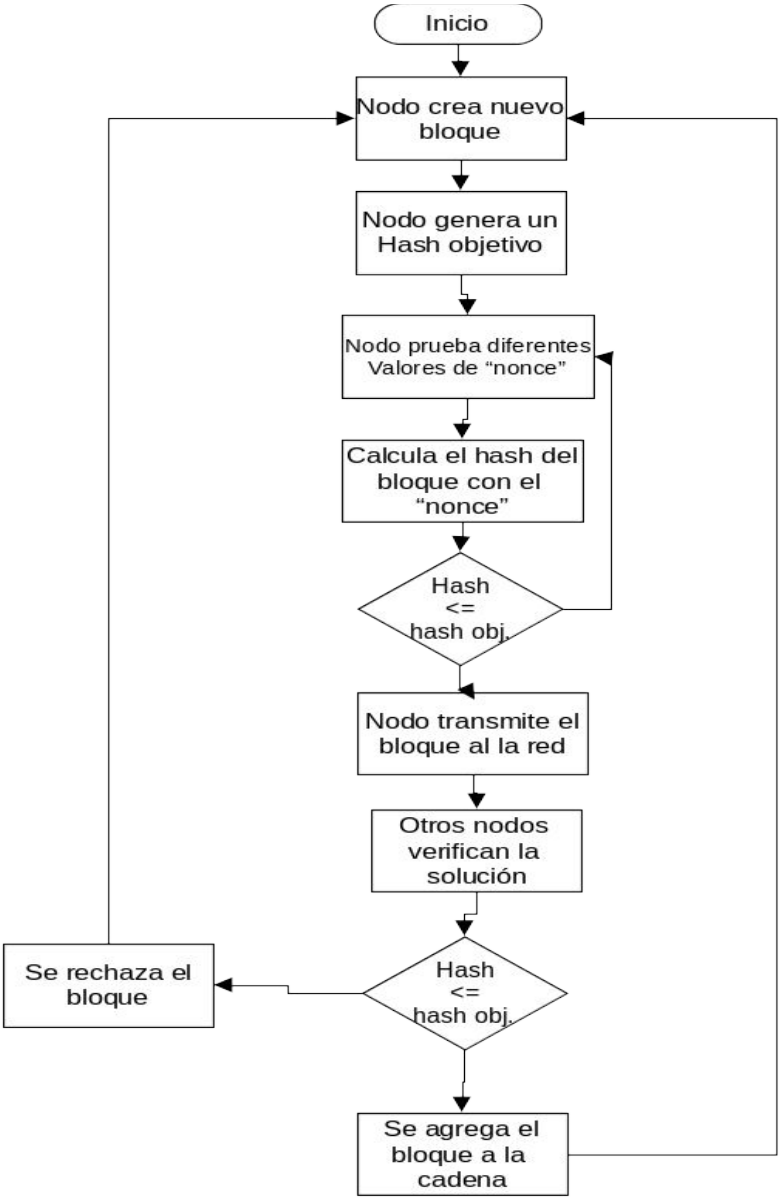

Figura 7: Diagrama de Flujo del Funcionamiento del algoritmo de consenso Proof of Work (PoW)

Aunque este algoritmo es mayormente usado en el blockchain, en una red P2P puede ser un excelente método de verificación de datos; y con esto tener una medida de seguridad para evitar ataques ya que obliga a los nodos a realizar trabajo computacional antes de aceptar transacciones o cambios, reduciendo el riesgo de ataques maliciosos. Aunque como en

todo, este algoritmo aporto ciertas ventajas como desventaja las cuales expondré a continuación en la Tabla 6.

Tabla 6: Tabla de ventajas y desventajas del algoritmo de PoW

Criterio	Ventajas	Desventaja
Seguridad	Dificulta ataques de Sybil y spam al requerir potencia computacional para participar.	No protege contra ataques del 51% (si un atacante controla la mayoría del poder de cómputo).
Descentralización	No depende de una autoridad central para la validación.	Puede favorecer a nodos con mayor poder de cómputo, creando desigualdad en la red.
Consumo de recurso	No requiere almacenamiento intensivo, solo cálculos matemáticos.	Alto consumo de energía y potencia computacional.
Tiempo de validación	Introduce un mecanismo confiable de verificación antes de aceptar bloques.	La validación de bloques puede ser lenta, afectando el rendimiento del sistema.
Resistencia a manipulación	Una vez validado, un bloque es difícil de modificar, asegurando la integridad de la red.	No impide completamente los intentos de manipulación, solo los hace computacionalmente costosos.
Facilidad de implementación	Algoritmos de hash como SHA-256 o Scrypt son fáciles de implementar.	Requiere ajustes en la dificultad para balancear seguridad y eficiencia.

Por lo anteriormente mencionado es la razón por la que este algoritmo fue el escogido como medida de seguridad para red, ya que aunque posee ciertas desventajas como el consumo de recurso, también resuelve enormemente el desafío de la seguridad dentro de nuestra red P2P.

2.3.4 Conclusión de la elección de estas Tecnologías

Gracias al empleo de estos protocolos y algoritmos fue posible resolver cada uno de los desafíos que presenta la creación de una red P2P además de que cada una de estas tecnologías se hicieron mediante la programación distribuida o están emparentadas con ella.

Por ultimo y como cierre a esta parte diré los desafíos que nos ayudo a superar cada uno de estos protocolos y algoritmos:

- **UDP** mejora la velocidad y reduce la sobrecarga de comunicación
- **Bully y Anillo** permite la autogestión de la red sin un servidor central.
- **PoW** protege la red contra ataques y garantiza que los datos sean validados correctamente.

Gracias a la combinación de este conjunto se logro hacer que la red P2P funcione de manera eficiente, descentraliza y segura cumpliendo a su vez las reglas del paradigma del computo distribuido.

2.4 Tópicos avanzados del cómputo distribuido

Antes de concluir veremos el estado del arte o campo en lo que respecta al computo distribuido para este observaremos algunas aplicaciones que se le a dado, junto con algunos ejemplos.

El computo distribuido ha avanzado considerablemente desde sus inicios, con aplicaciones que van desde sistemas de procesamiento en paralelo hasta infraestructuras de servicio en la nube. La capacidad de distribuir tareas entre múltiples nodos independientes ha permitido mejorar el rendimiento, la disponibilidad y adaptabilidad de los sistemas informáticos. Unos ejemplos de sus múltiples aplicaciones seria:

- En el sector de la *sanidad y ciencias biológicas* donde se usa la computación distribuida para modelar y simular datos complejos de las ciencias biológicas. El análisis de imágenes, la investigación de medicamento y el análisis de la estructura de los genes se vuelve más rápido con los sistemas distribuidos.
- Los *ingenieros* pueden simular conceptos complejos de física y mecánica en sistemas distribuidos. Usan esta investigación para mejorar el diseño de los productos, crear estructuras complejas y diseñar vehículos más rápidos.
- Las empresas de *servicios financieros* usan sistemas distribuidos para realizar simulaciones económicas de alta velocidad que evalúa los riesgos de las carteras, predicen los movimientos del mercado y respalda la toma de decisiones financieras.

- En la *educación y la investigación,* el computo distribuido es un área clave para el desarrollo de habilidades en el diseño y la implementación de sistemas complejos que puedan funcionar eficientemente en entornos distribuidos.

La principal ventaja de los sistemas distribuidos es que permiten la replicación de los recursos o servicios, reduciendo así la interrupción del servicio debido a fallas (M. Larios G.. (2020, p. 25); haciéndolo idóneos para diferentes usos como se mostró anteriormente.

Entre algunos de los tópicos mas destacables se encuentran el aprendizaje federado, la computación cuántica distribuida y la integración de tecnologías emergentes como el edge computing y las redes definidas por software (SDN).

Aprendizaje Federado y Computación en el Borde (Edge Computing)

Esta tecnología emergió como una solución eficaz para entrenar modelos de inteligencia artificial de manera distribuida, manteniendo la privacidad de los datos al no requerir de su trasferencia a un servidor central. Dicho enfoque es sobresaliente en entornos de edge computing, ya que los dispositivos terminales poseen capacidades de procesamiento y almacenamiento limitados. Wu et al. (2023) realizo una revisión exhaustiva sobre las topologías de red en el aprendizaje federado dentro del edge computing, destacando la importancia de considerar la estructura de la red para optimizar el rendimiento y la eficiencia del sistema.

Computación Cuántica Distribuida

Siendo un paradigma emergente la computación cuántica distribuida busca combinar múltiples procesadores cuánticos para resolver problemas que superan las capacidades de un solo dispositivo. Caleffi et al. (2022) proporciono una visión general de los principales desafíos y problemas que se presentan en este campo, enfatizando la necesidad de desarrollar nuevas arquitecturas y protocolos de comunicación para que sea mas sencilla la cooperación entre procesadores cuánticos distribuidos.

Integración de SDN, loT y Blockchain

La convergencia de las redes definidas por software (SDN), el Internet de las cosas (loT) y la tecnología blockchain están revolucionando el panorama del cómputo distribuido. Shafiq et al. (2024) analizaron cómo la integración de estas pueden mejorar la gestión de recursos, seguridad y la escalabilidad en sistemas distribuidos, poniendo énfasis en aspectos tales como la movilidad, privacidad y la eficiencia energética.

Computación Serverless y Sostenibilidad

Aunque reciente la computación serveless ha ganado un considerable popularidad debido a su abstracción en la gestión de la infraestructura lo cual permite a los desarrolladores centrarse en el código y la funcionalidad. Las principales ventajas que ofrece este modelo son la escalabilidad y la eficiencia en costos al ejecutar aplicaciones. Esto a permitido que los desarrolladores se enfoquen en diseñar sistemas que minimicen el consumo energético y reduzcan la huella de carbono, manteniendo un alto rendimiento.

Peer-to-Peer

En el caso de las redes Peer-to-Peer (P2P) son un ejemplo crucial de cómputo distribuido. En una red P2P, cada nodo actúa como cliente y servidor al mismo tiempo, lo que permite una distribución equitativa de recursos y tareas. Este enfoque es fundamental para el desarrollo de sistemas que no dependan de un único punto de falla y que pueda escalar según la necesidad (M. Larios G., 2020, p.21).

Por esta razón es que las redes P2P han encontrado aplicaciones en diversas áreas, desde compartir archivos (como en BitTorrent) hasta tecnologías emergentes como las criptomonedas y blockchain, donde la descentralización es esencial.

Un ejemplo de la combinación del computo distribuido y las redes P2P fue el que se empleo en el trabajo de la “Clustering distributed data stream in peer-to-peer environments” en el que los autores agruparon un grupo de datos totalmente distribuidos en un entorno P2P como redes de sensores, basándose en los principios del algoritmo de K-Means. Las redes de sensores se comunican de manera peer-to-peer (P2P) que permiten sólo la comunicación local entre los nodos de sensores vecinos. Esto requiere que los algoritmos de análisis de datos también se comuniquen en modo P2P y funcionen de manera asincrónica. Hasta ahora no se ha informado de tal algoritmo de agrupación en la literatura. (Bandyopadhyay et al., 2006).

Capitulo 3. Análisis y Diseño de “NeoAres”

En este capitulo se presentara el análisis y diseño que se llevo a cabo para la realización del sistema “NeoAres”, el cual es una red P2P desarrollada como parte del presente trabajo. Un dato a mencionar es que el nombre de este sistema fue elegido para homenajear a Ares, el cual fue un software ampliamente utilizado en décadas pasadas para el intercambio de archivos entre pares.

En este capitulo se describen los requisitos esenciales para el desarrollo de dicho sistemas, tales como la comunicación entre nodos, los mecanismo de enrutamiento, la seguridad en la transmisión de los datos, además de la forma en que se implementaran los algoritmos y protocolos.

Asimismo se incluyen varios diagramas que ilustran la interacción entre los distintos componentes del sistema, con el objetivo de facilitar su comprensión y facilitar sus futuras replicaciones o mejoras del proyecto.

3.1 Requisitos del Sistema

El establecimiento de los requisitos fue uno de los pasos fundamentales para el diseño de “NeoAres”, ya que sentó las bases para construir una aplicación distribuida robusta; ya que al tratarse de una red P2P es esencial que los requerimientos cumplan con los principios del computo distribuido, tales como una infraestructura descentralizada, escalable y tolerante a fallos.

3.1.1 Requisitos Funcionales

Los requisitos funcionales describen las capacidades específicas que debe cumplir el sistema para su correcto funcionamiento:

1. **Descentralización:** La red opera sin un servidor central, permitiendo que todos los nodos actúen como clientes y servidores simultáneamente. Gracias a esto cada nodo es autónomo, evitando de esta manera puntos de falla únicos.

2. **Paso de mensajes:** La comunicación entre nodos se realiza mediante el intercambio de mensajes a través de sockets utilizando el protocolo **UDP** asegurando rapidez y eficiencia en la transmisión de datos.
3. **Mantenimiento y consistencia:** Los nodos se sincronizan para mantener la coherencia de la información distribuida en la red, asegurando que todos los participantes tengan una vista actualizada del estado del sistema.
4. **Seguridad y consenso:** Para validar la autenticidad de los datos y evitar manipulaciones maliciosas, se empleo un mecanismo de consenso basado en **Proo of Work (PoW)** ya que este garantiza que solo los bloques válidos sean aceptados en la red.
5. **Autodescubrimiento de nodos:** Los nuevos nodos que se unen a la red son capaces de descubrir y conectarse con otros sin intervención manual, facilitando la escalabilidad y la autonomía del sistema.

3.1.2 Requisitos No Funcionales

Los requisitos no funcionales garantizan la eficiencia, estabilidad y escalabilidad del sistema:

1. **Escalabilidad:** La arquitectura debe permitir la incorporación de nuevos nodos son que esto degrade el rendimiento general de la red.
2. **Tolerancia a fallos:** La red de continua operando incluso si algunos nodos fallan o se desconectan inesperadamente, asegurando la disponibilidad del sistema.
3. **Baja Latencia:** La comunicación entre nodos garantizo tiempos de respuesta rápidos, minimizando la latencia en la transmisión de datos.
4. **Eficiencia en el uso de recursos:** El consumo de ancho de banda y CPU fue óptimo ya que evito la sobrecargar los nodos participantes y garantizo un rendimiento estable.

Bajo estos requisitos fue que se desarrollo la basa para el diseño e implementación de la red P2P, asegurando que el sistema sea robusto, eficiente y alineado con los principios del computo distribuido. Y a partir de estos mismo se desarrollo la arquitectura de la red y los mecanismos de interacción entre los nodos participantes.

3.2 Arquitectura del Sistema

La arquitectura de la red P2P propuesta se baso en el modelo descentralizado donde todos los nodos tienen la misma jerarquía funcional y se comunican de forma directa entre ellos, lo que favorece la escalabilidad, autonomía y resiliencia del sistema.

3.2.1 Topología de la Red

Se adopto una **arquitectura descentralizada,** donde todos los nodos tienen la misma función y pueden comunicarse directamente entre sí. Esta decisión aporto las siguientes ventajas:

- **Mayor resiliencia:** No existe un punto único de fallo.
- **Escalabilidad:** La red puede crecer sin necesidad de modificar su estructura.
- **Distribución equitativa de la carga:** Cada nodo contribuye de manera similar a la red como se puede ver en el diagrama que se muestra a continuación en la Figura 8.

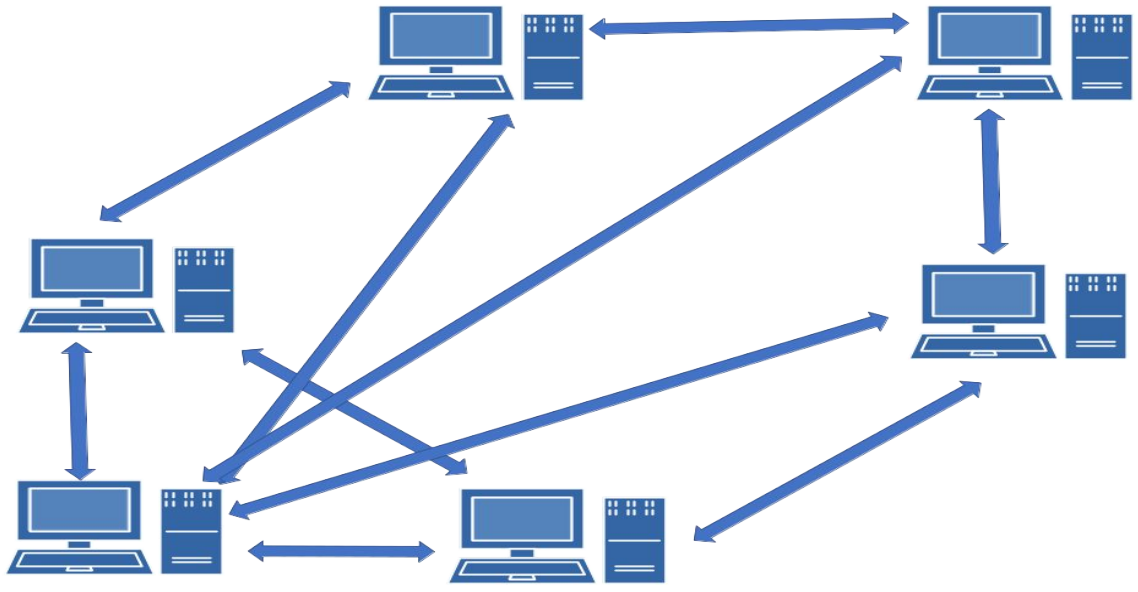

Figura 8: Ejemplo de arquitectura descentralizada

.

Dicha topologia fue implementada a través del uso de *sockets multicast,* el cual permite que todos los nodos puedan suscribirse aun grupo de comunicación común y transmitir mensaje entre sí. Este enfoque permite eliminar la necesidad de un servidor central con lo cual cualquier nodo puede iniciar una conversación o recibir mensajes y archivos (“*Este*

mecanismo se detalla en el Capitulo 4, sección ----, donde se muestra la implementación del método ***send_message*** *en la clase* ***Chat***”).

El sistema también tiene mecanismos complementarios como los algoritmos Bully y Anillo que permiten la coordinación temporal entre los nodos cuando es necesario, por supuesto sin comprometer la igualdad de funciones entre los nodos. Esta combinación mantiene la red descentralizada mientras permite algunas operaciones organizadas de forma dinámica.

Además de esta descripción conceptual en la siguiente imagen se mostrara un diagrama de clases que representa la estructura general del sistema y su topología desde el punto de vista del software.

A continuación, se presenta en la Figura 9 el diagrama general del sistema:

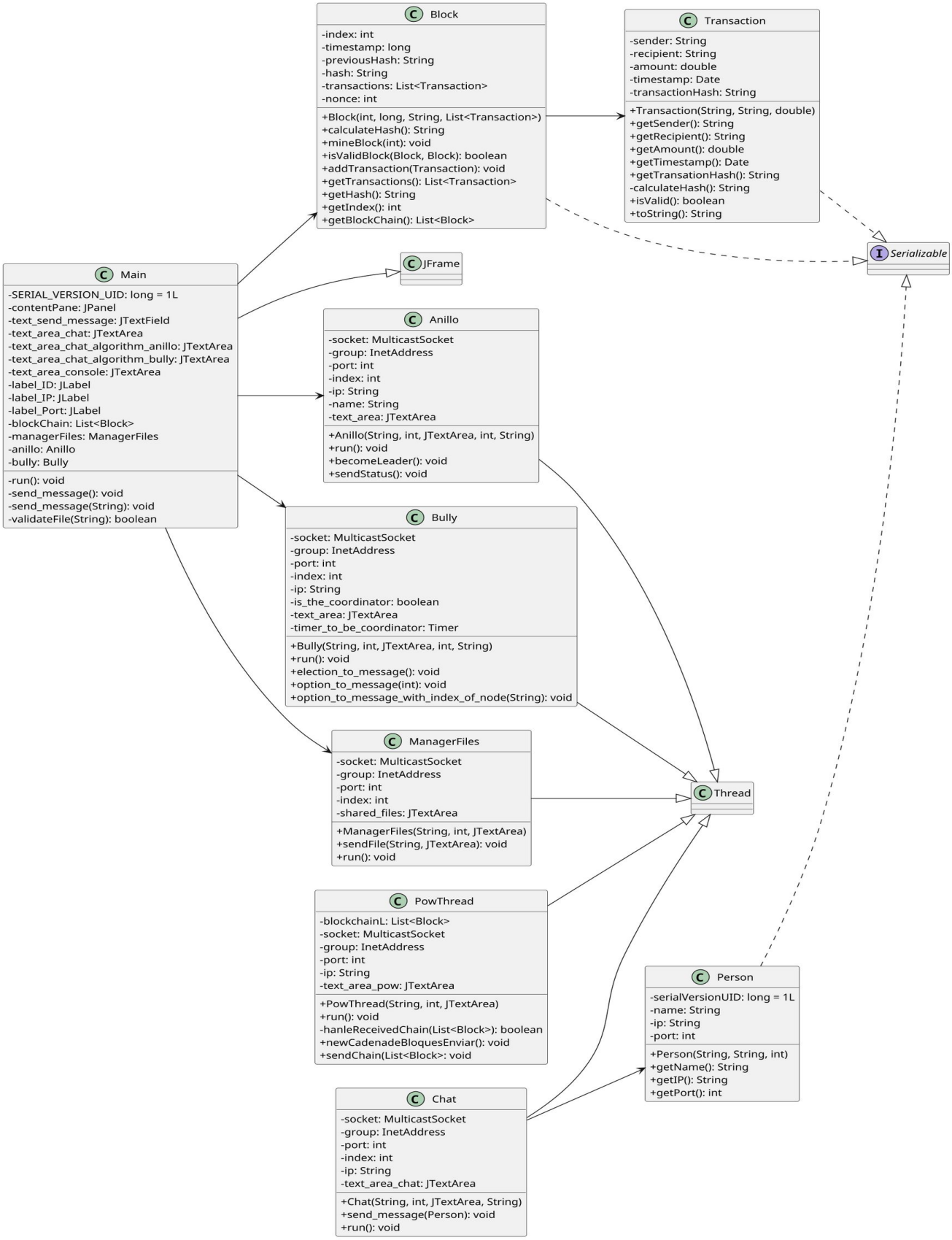

Figura 9: Diagrama de Clases

3.2.2 Protocolo de Comunicación

Como ya se había esclarecido en el "*Capitulo 2, sección 2.3.1"* el protocolo que se eligió para la comunicación entre los nodos fue **UDP** el cual se implemento a través de **Java MulticastSockets** ya que es altamente eficiente y de baja sobrecarga. A diferencian de **TCP** este protocolo no requiere de una conexión persistente, lo cual lo hace ideal para redes dinámicas como las P2P.

En la practica, al enviarse un mensaje este se llevara a través de UDP mediante un objeto serializado, y todos los nodos que esté suscritos al mismo grupo lo recibirán y si el mensaje proviene de otro nodo, se le procesa y luego se muestra en su interfaz gráfica. Este comportamiento refleja el principio de igualdad entre los nodos y su independencia de operaciones como se ve en la Figura 10.

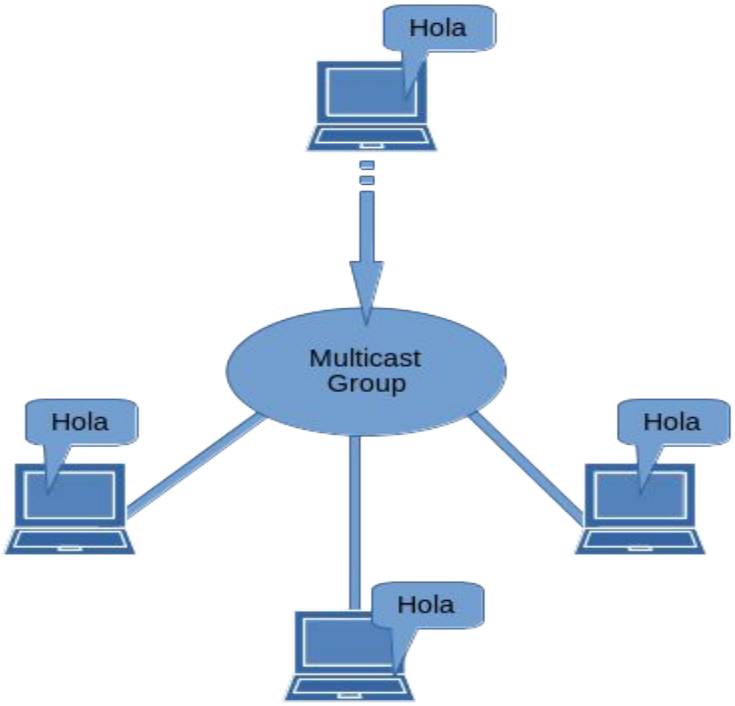

Figura 10: Representación del funcionamiento de UDP mediante MulticastGroup

3.2.3 Módulos del Sistemas

En el diseño de un sistema es importante saber como dividirlo en partes mas pequeñas para de esta forma facilitar su diseño, en este caso el proyecto de “NeoAres” fue dividido en 4 partes principales los cuales fueron **Chat, Algoritmos de Coordinación, Seguridad con PoW, Sistema de Archivos Compartidos.** Estos 4 módulos se irán describiendo a continuación.

3.2.3.1 Chat

Aunque su concepto es simple su propósito es fundamental en el funcionamiento del sistema ya que permite las comunicaciones entre los nodos en tiempo real.

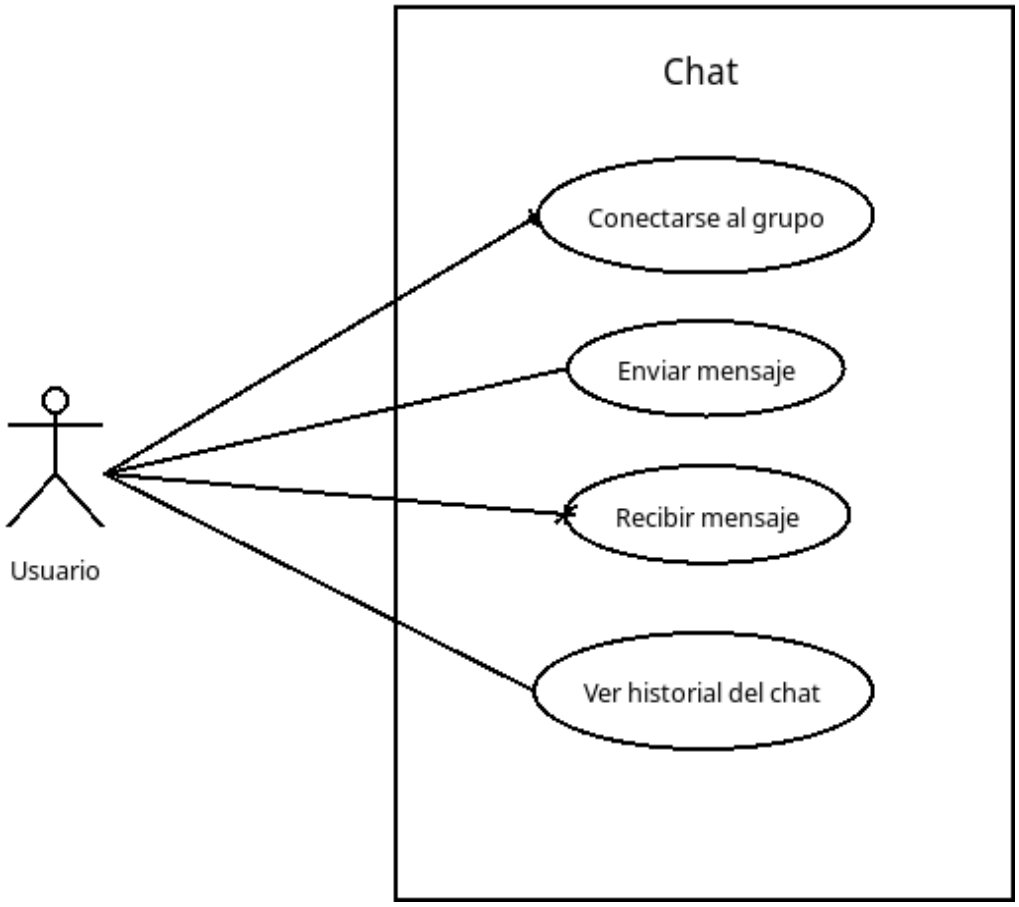

Figura 11: Casos de uso del chat

Como puede observarse en la Figura 11 la interacciónes que tiene el usuario con el sistema de mensajería son principalmente las acciones de enviar o recibir mensajes, las cuales se realizan mediante Multicast UDP.

Para manejar el proceso de recepción de mensaje de nuestro sistema, se implementaron una serie de pasos a seguir, los cuales para facilitar su entendimiento se mostraran mediante el siguiente diagrama de secuencia que se ve en la Figura 12.

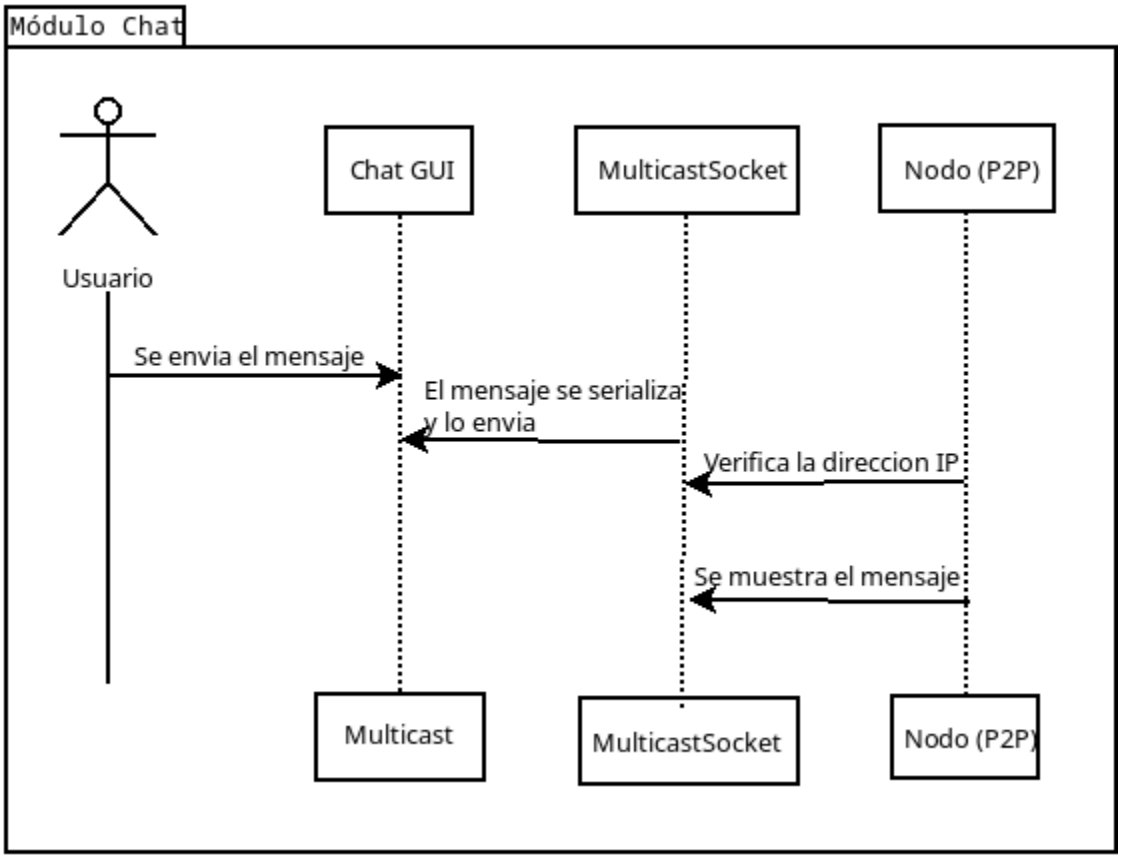

Figura 12: Diagrama de Secuencia del Chat

3.2.3.2 Algoritmos de Coordinación

Pese a que la red descrita a lo largo de este documente es de tipo descentralizada y que por los consiguiente cada nodo es independiente entre si; algunos procesos y algoritmos requieren la elección o existencia de un nodo coordinador para optimizar ciertas tareas o mejorar el rendimiento general de la red. Unos ejemplos de las funciones del coordinador serian:

- **Gestión de tareas:** El coordinador puede asumir ciertas responsabilidades especificas, como la asignación de recursos, coordinación de nodos, resolución de conflictos; evitando de esta manera que cada nodo tenga que negociar constantemente con todos los demás.

- **Distribución de la carga:** Algunas operaciones se gestionan de manera mas eficiente si un nodo actúa como líder temporal (como por ejemplo la sincronización de archivos).
- **Consenso y toma de decisiones:** Algunos algoritmos P2P necesitan un nodo lider para implementar algoritmos de consenso como por ejemplo el que se aplico en el siguiente modulo (PoW).

En esencia, aunque nuestro sistema es descentralizo, tener un coordinador mediante algoritmos es una practica que mejora la escalabilidad, eficiencia y la confiabilidad de esta misma; para esta tarea se eligieron los algoritmos: **Anillo y Bully**.

.

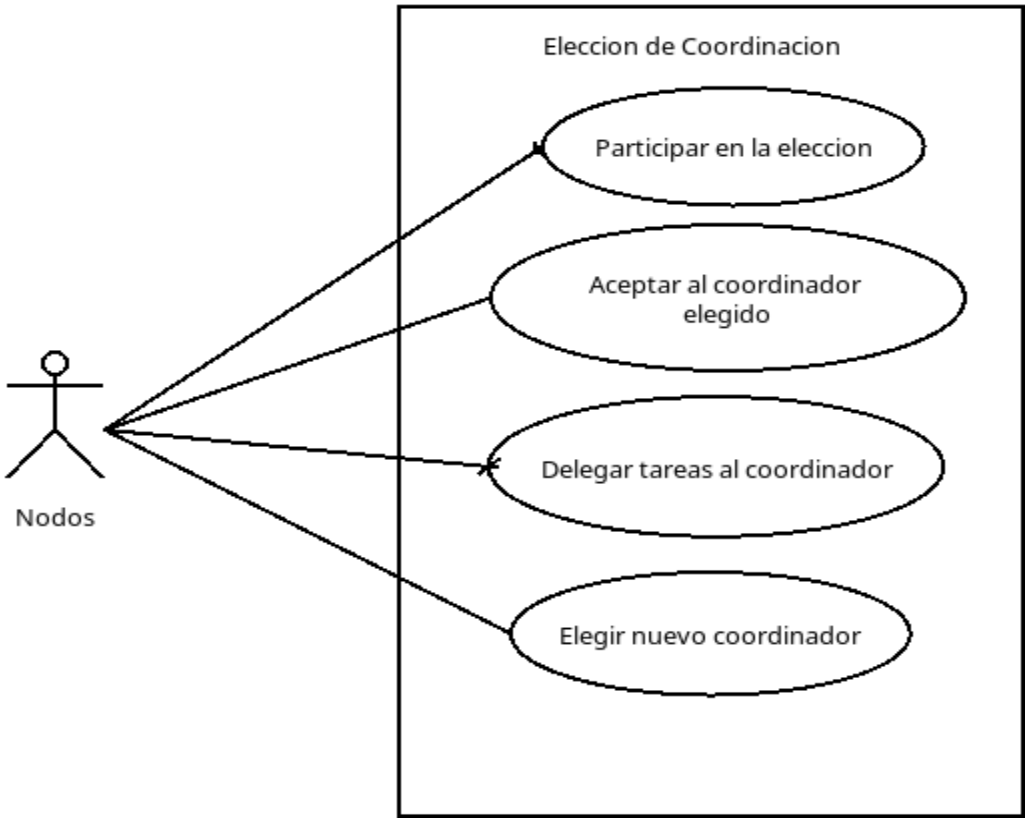

Figura 13: Casos de uso para la elección de un coordinador

¿Por que dos algoritmos de coordinación o elección?

La principal razón por la que se eligieron dos algoritmos de coordinación fue por que ambos permiten abordar el mismo problema (la eleccion de un coordinador en una red distribuida) pero desde dos enfoques distintos mejorando de esta manera la tolerancia a fallos del sistema además claro que le da una mejor adaptabilidad a diferentes entornos:

- El algoritmo de **Anillo** utiliza una topología circular donde cada nodo conoce únicamente a su compañero de al lado, siendo de esta manera más eficiente en cuanto a uso de mensajes garantizando que todos los nodos participen en la elección.
- Por otro lado el algoritmo de **Bully** busca los nodos con el identificador (ID) mas grande , siendo de esta manera mas rápido en redes pequeñas pero con un mayor consumo de mensajes en redes grandes.

Otro punto a tener en cuenta sobre la razón por la que se escogieron dos algoritmos de coordinación fue que nuestro sistema tiene también un enfoque educativo y por lo tanto no solo busca ofrecer funcionalidad si no que también busca mostrar el **"el paradigma del cómputo distribuido"** mediante el uso de diferentes estrategias para reflejar como estas puedan aplicarse en un mismo problema pero con puntos de vista alternativos.

Algoritmo de Anillo

Como en el capitulo anterior ya se abordo el algoritmo de anillo (Capitulo 2, Sección 2.3, Sub-sección 2.3.2), ahora únicamente mostraremos un diagrama de flujo del análisis del funcionamiento del algoritmo dentro de la red, en caso de que no exista un coordinador o se aya detectado su ausencia, para el cual previamente los nodos ya se abran colocada en forma de anillo lógico dentro del sistema de nuestra red.

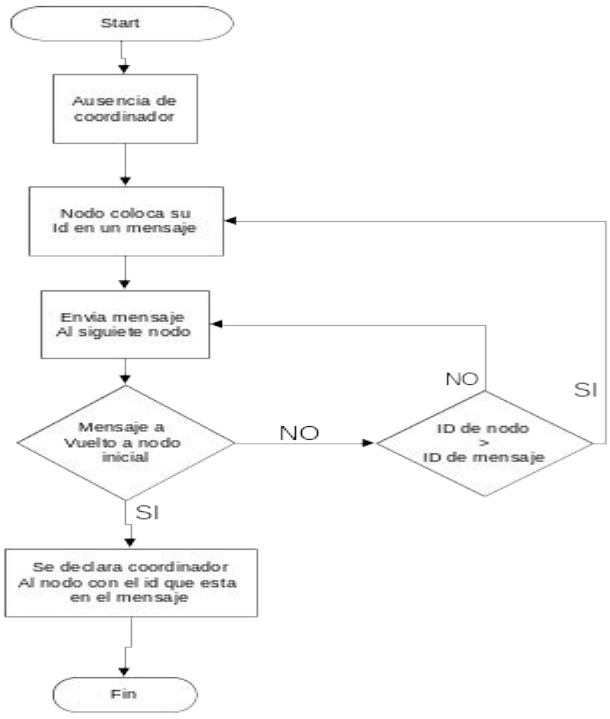

Figura 14: Diagrama de flujo del algoritmo de Anillo

Como puede observarse en el diagrama de la figura 14 una vez que se detecta la ausencia del coordinador se envia un mensaje de eleccion de coordinador con la Id del nodo que la envió y al pasar al siguiente si se detecta que la id del nodo actual es superior a la del nodo anterior se sustituye la id del mensaje con la id mayor y pasa al siguiente, asta que se complete la vuelta y por lo tanto la ID que quede en el mensaje sera la mas grande y por lo tanto el nodo dueño de dicha ID sera nombrado coordinador.

Algoritmo de Bully

Como en el caso anterior únicamente se mostrara un diagrama que refleje el funcionamiento del algoritmo para evitar la redundancia con el capitulo anterior, aunque en este caso en lugar de ser un diagrama de flujo se empleara un diagrama de secuencia ya que en el caso de este algoritmo dicho esquema es mas útil para representar la temporalidad (quien envía a quien y en que orden) lo cual es necesario para entender este algoritmo.

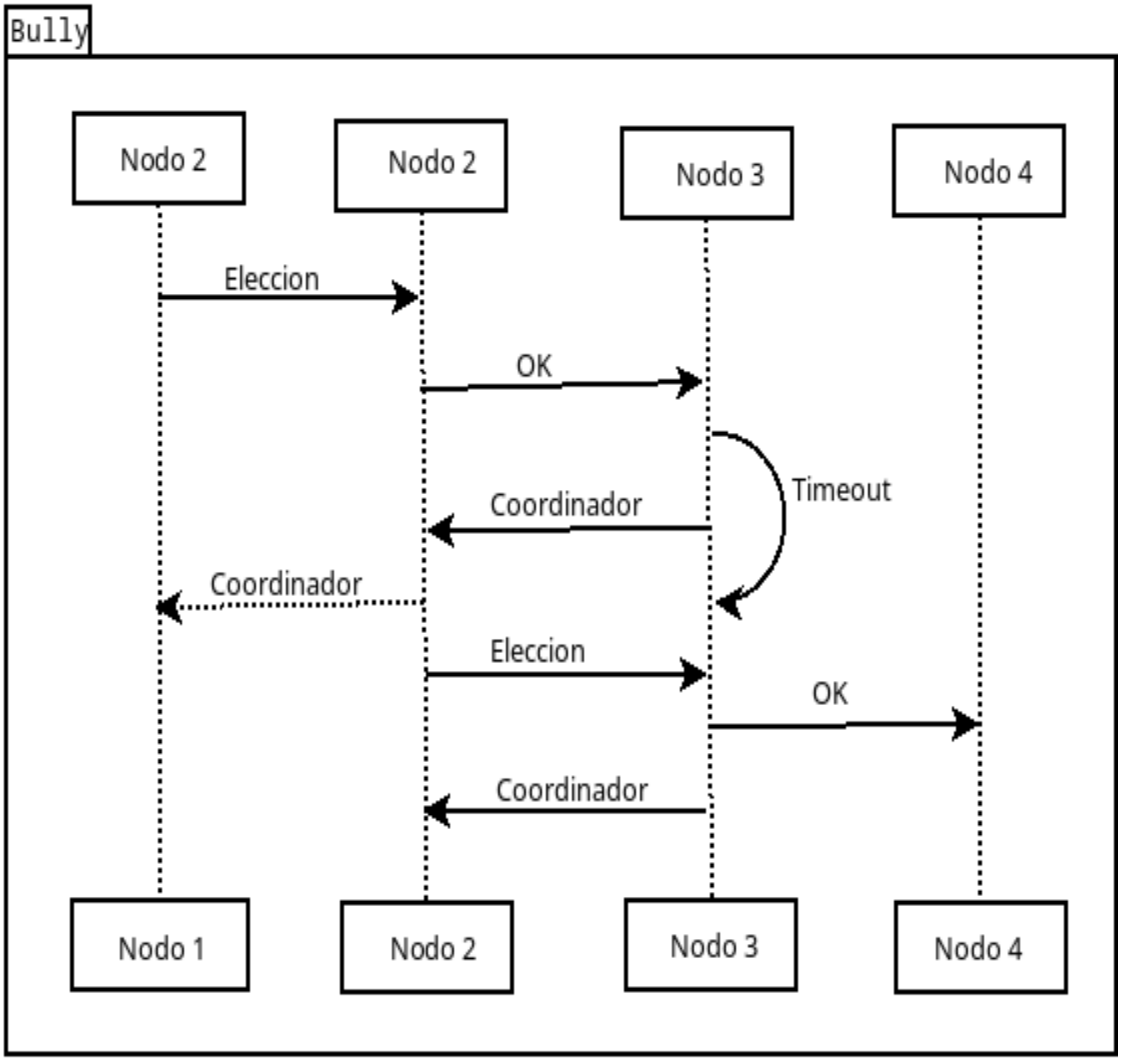

Figura 15: Diagrama de secuencia del algoritmo de Bully

Como puede observarse en el diagrama de la Figura 15 que representa como un nodo al detectar que no hay un coordinador activo, inicia el proceso de elección y con eso desata todo el programa:

1. Detección de ausencia de coordinador
    - Nodo 1 detecta que el coordinador ya no responde o no existe
    - Envía un mensaje de **Elección** al nodo 2 el cual tiene un ID mas grande
2. Nodo 2 recibe el mensaje del nodo 1
    - Le responde con **OK,** como el nodo 2 tiene un ID mas grande asume el cargo de la elección
    - Nodo 2 empieza su propia elección y envía **Elección** al nodo 3
3. Nodo 3 recibe el mensaje de Nodo 2
    - Le responde que **OK** y al tener un ID superior asume el cargo de la elección
    - Nodo 3 inicia una elección con Nodo 4 el cual tiene un ID superior
4. Nodo 4 no responde (Timeout)
    - Esto indica que Nodo 4 se encuentra ausente o no se a desconectado de la redacción
    - Entonces Nodo 3 se autoproclama coordinador y envía un mensaje **Coordinador** a los demás nodos para notificarles

Con esta información podemos ver que lo que busca el algoritmo de **Bully** es que el nodo con el ID mas grande sea el nuevo coordinador y para esto hace que durante la búsqueda los nodos con ID superiores se hagan cargo de la elección para que de esta manera busquen otros nodos con ID mas grande que el que posen y de esta forma delegarles las tarea y que el proceso se repita de nuevo asta que no aya un nodo mas grande.

3.2.3.3 Seguridad con PoW (Proof of Work)

Como se abordo previamente en el capitulo anterior (Capitulo 2, Sección 2.3, Sub-sección 2.3.3) , el algoritmo **Proo of Work (PoW),** constituye como un medio esencial para garantizar la validación y seguridad de las operaciones dentro de las redes P2P. En el contexto de nuestro sistema “NeoAres”, este algoritmo no solo reforzó la protección frente a ataques

externos como la suplantación de nodos o el envío de archivos maliciosos, sino que también cumple la función de controlar el acceso a la red y la información distribuida en ella.

Aplicación en NeoAres

Aunque este algoritmo es normalmente usado en las criptomonedas como el Bitcoin, en NeoAres se empleo para validar los mensajes de comunicación o las acciones criticas entre nodos, además de mantener segura la red de ataques.

Para ello se establecieron los siguientes objetivos que debía cumplir su implementación.

- Evitar el spam o bombardeo de mensajes por parte de nodos maliciosos.
- Aumentar la confianza entre pares, al exigir prueba de participación real.
- Fortalecer el sistema de coordinadores, de forma que las solicitudes de elección pasen por una verificación computacional previa.

De esta forma el algoritmo debería activarse cada vez que se que den ciertas operaciones sensibles como las que se ilustraran a continuación:

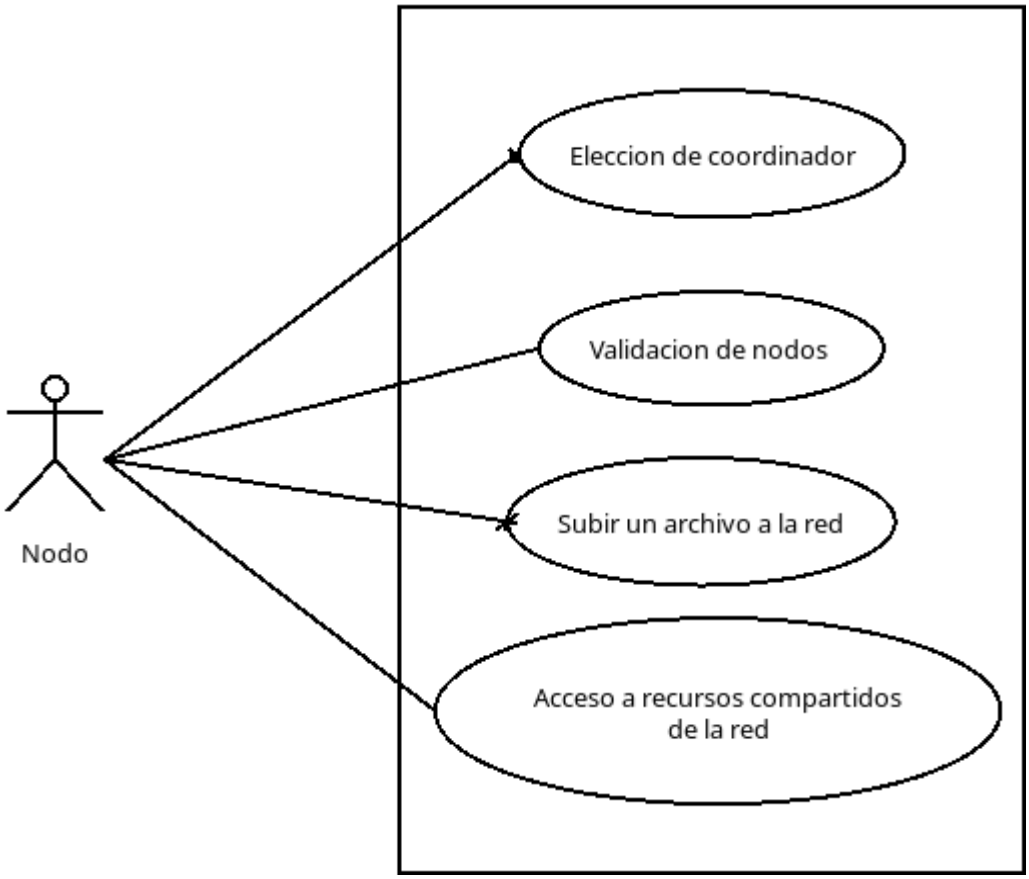

Figura 16: Casos de uso del algoritmo de conceso PoW

- **Solicitud de elección de coordinador:** Cuando un nodo detecta la ausencia de un coordinador puede iniciar una elección, pero antes debe presentar una prueba de trabajo válido para demostrar que no es un nodo malicioso.
- **Solicitud de unión a la red o acceso a compartidos de la red:** Antes de que cualquier nodo pueda acceder a la red junto a sus recurso deberá pasar una prueba de trabajo como condición previa.
- **Ingresar archivos a la red:** Antes de que un nodo pueda subir un archivo en la red primera deberá resolver un desafío PoW para validar su autenticidad. Con esto se pretende reducir la probabilidad de ataques de spam o intentos de propagación de archivos maliciosos.

Para un mayor entendimiento se presentara a continuación un diagrama de secuencia en el cual se ilustra de manera clara el proceso que siguen los nodos en NeoAres para validar una acción crítica mediante PoW. Se puede observar como un nodo que desea ejecutar una acción dentro de la red (como iniciar una elección de coordinador o subir un archivo a la red) necesita primero resolver un problema computacional.

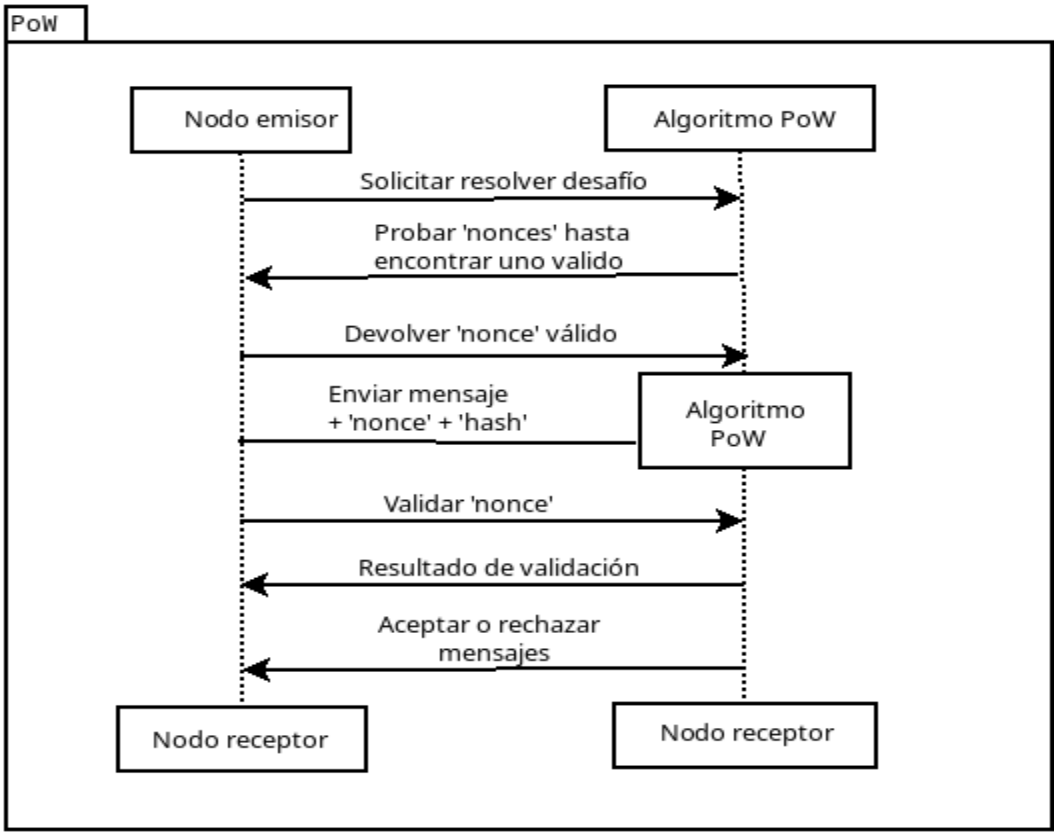

Figura 17: Diagrama de secuencia de PoW

.

Este mecanismo, aunque es mas liviano que el usado en sistemas como Bitcoin, cumple con su objetivo de prevenir abusos, reducir riesgos de ataques DoS y de reforzar la validación antes de cada interacción dentro de la red.

3.2.3.4 Sistema de Archivos Compartidos

Uno de los módulos mas fundamentales y complejos de NeoAres fue el **sistemas de archivos compartidos,** esto se debe principalmente a que es el encargado de otorgar al sistema la capacidad de cumplir con su propósito principal de intercambiar archivos entre nodos de manera descentralizada. Este subsistema aunque en la teoría puede verse como sencillo, implico una serie de desafíos técnicos significativos durante su diseño, principalmente por la necesidad de operar sin un servidor central, garantizar cierta robustez ,evitar la saturación y la perdida de información durante la transmisión de archivos de gran tamaño.

Desde un punto de vista arquitectónico, el sistema fue pensado como una especie de cliente-servidor entre pares de forma que cada nodo pueda actuar tanto como emisor como receptor de archivos, dependiendo del momento. Para que esto fuera posible, la comunicación multicast **UDP** fue clave ya que permite difundir los archivos a todos los nodos suscritos al grupo multicast, sin requerir una conexión punto a punto con cada uno de ellos.

Aun con esto los retos a superar fueron varios y complejos de superar:

1. **Multicast sin control de flujo:**
    - **UDP** aunque es una tecnología liviana no garantiza la entrega, orden o el control del flujo de los paquetes. Lo cual nos obligo a tomar decisiones a nivel de aplicación para garantizar cierta integridad y coordinación en la transferencia.
2. **Fragmentación y ensamblado de archivos**
    - Ya que UDP tiene un limite teórico de tamaño para los paquetes (~65 KB) fue necesario implementar una lógica de fragmentación para asi poder dividir los archivos grandes en múltiples datagramas lo cual a su ves obligo a que se implementara una lógica para reconstruir los archivos una vez llegaran a su destino.
3. **Gestión de rutas y directorios**

- Para que lo archivos pudieran guardarse correctamente en el sistema receptor, fue necesario implementar validaciones automáticas del sistema de archivos, creando carpetas si estas no existían y asegurando rutas relativas válidas.

4. **Evitar colisiones y auto-recepción**
    - Cuando un nodo envía un archivo también puede recibirlo de vuelta al pertenecer al mismo grupo multicast. Para que esto no ocurra se comparan las direcciones IP del emisor y receptor.
5. **Separación de metadatos y contenido**
    - Se estableció un protocolo donde primero se envía el nombre del archivo y luego los fragmentos de datos binarios.

Con estos retos y las posibles soluciones que se idearon se diseño el siguiente flujo que debería seguirse para que los archivos llegaran a sus destinos de manera óptima y con la menor perdida de información posible.

1. **Inicialización**
    - El nodo se une a la red
    - Genera un entorno local para el almacenamiento
2. **Recepción de archivos**
    - Se detecta un paquete con **metadatos** y se genera un archivo con su nombre
    - Recibe los bloques de archivos enviados en paquetes
3. **Construcción del archivo**
    - los datos recibidos se almacenan y reconstruyen el archivo
    - Al ya no recibir mas paquetes se da por terminada la tarea

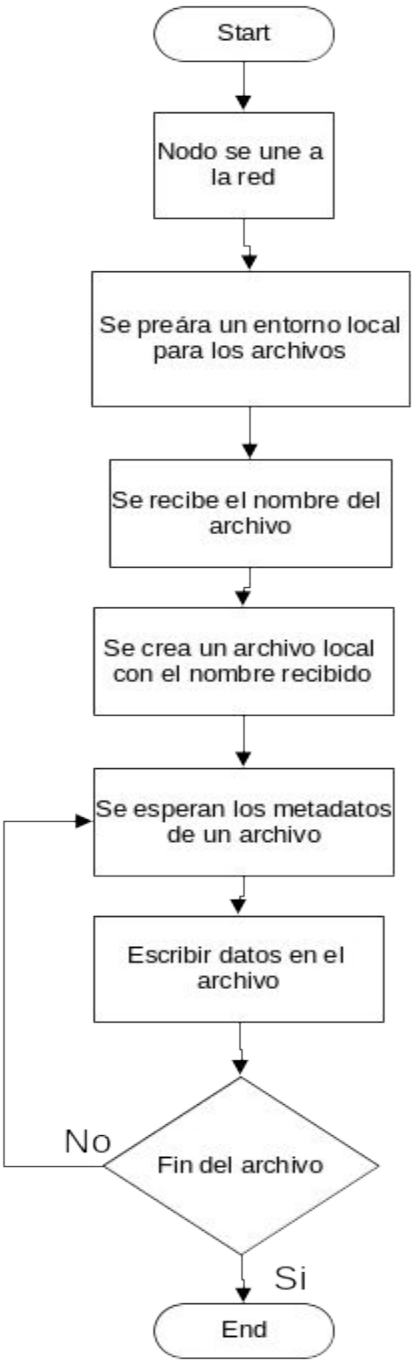

Figura 18: Diagrama de flujo de la recepción de un archivo

Aunque existen múltiples soluciones modernas para el intercambio de archivos en red, se opto por un enfoque de bajo nivel para garantizar el aprendizaje, la flexibilidad y la adaptación a una arquitectura P2P sin intermediarios. Es por esto que este modulo fue uno de los mas complicados de plantear en el sistema de NeoAres y representa fielmente los principios del cómputo distribuido que se buscaban explorar desde el planteamiento inicial del proyecto.

Capitulo 4. Desarrollo de “*NeoAres*”

En este capitulo se explicara todo el proceso de desarrollo e implementación de *NeoAres,* a diferencia del capitulo anterior que fue mas teórico aquí se mostrara nuestra aplicación desde un punto mas técnico y orientado a la programación que se tuvo que realizar para que esta red funcionara de manera óptima; aunque claro siguiendo los lineamientos definidos en el capitulo anterior (Análisis y diseño), con lo cual se procedió a realizar la construcción de cada uno de los módulos que integran el sistema, abarcando desde la comunicación entre los nodos hasta la validación de operaciones mediante mecanismos de consenso.

A lo largo de este capitulo se irán detallado las decisiones de programación que se tomaron, el como se organizo el código fuente, los retos técnicos a los que se enfrento el desarrollo durante la implementación junto a las estrategias que se tomaron para superarlos. Para esto se incluyen varios fragmentos clave del código fuente, acompañados de sus respectivas explicaciones de su funcionamiento y propósito dentro del sistema.

Al final de este capitulo se incluirán unas descripciones de las pruebas de validación que se le aplicaron al prototipo, evaluando aspectos clave como la comunicación entre nodos, coordinación distribuida, la transferencia de archivos y la resistencia ante ataques o fallos, en lo cual se demuestra la viabilidad del proyecto y su alineación con los principios del computo distribuido.

4.1 Implementación de los Módulos

Con base en la metodología planteada en el *Capitulo 1, Sección 1.4, Subsección 1.4.1;* el desarrollo de *NeoAres* se abordó siguiendo el enfoque de **Desarrollo Dirigido por Pruebas.** Como se vio en dicho capitulo esta estrategia consiste en diseñar primero pruebas específicas para la funcionalidades deseadas para que a continuación se escriba el código necesario para superar la prueba. Gracias a esto se puede mantener un control constante sobre el cumplimiento de los objetivos deseados y se redujo el riesgo de acumular errores durante la evolución del sistema.

Como complemento a esto y como una buena practica de ingeniería de software, se dividió el proyecto en módulos independientes esto se hizo con el propósito de:

- **Facilito el desarrollo incremental:** Permitiendo construir, probar y corregir cada componente de forma aislada antes de integrarlo con el resto del sistema.
- **Mejorar la mantenibilidad del código:** Ya que cada modulo posee responsabilidades claramente delimitadas es posible modificarlo o mejorar componentes especifico dentro de el sin comprometer el resto del sistema.
- **Permitió una integración más controlada:** Al validarse los modulos de manera independiente, fue posible identificar de forma mas precisa los problemas surgidos al integrarlos posteriormente al resto del sistema.

Con lo cual y como se observo en el capitulo anterior los modulos quedaron de la siguiente manera: Comunicación entre nodos (Chat), Coordinación distribuida (Bully y Anillo), seguridad mediante consenso (PoW) y trasferencia de archivos (Manager_files).

Este enfoque modular combinado con la metodología TDD, no solo permitió construir *NeoAres* de forma mas organizada y robusta, sino que también dio el plus de contribuir a uno de los objetivos principales del proyecto: facilitar el entendimiento práctico del paradigma del cómputo distribuido.

Por lo cual a continuación se mostraran los módulos desarrollados, sus estructuras internas, fragmentos mas relevantes o representativos del código y los retos que dio su implementación.

4.1.1 Módulo Chat

Al igual que se vio en el capitulo anterior el primer modulo en plantearse y desarrollarse fue el **sistema de chat** entre nodos ya que esta funcionalidad fue pensada como la base de comunicación directa en la red P2P.

El desarrollo del módulo se centró en implementar la transmisión de mensajes mediante **sockets multicas udp**, claramente respetando los principios de autonomía, igualdad y descentralización entre nodos. Los principal que se buco fue que cada mensaje enviado sea recibido por todos los nodos participantes conectados en el mismo grupo multicast, eliminando la necesidad de un servidor central.

Además de que este modulo sentó las bases para la interacción entre usuarios como para la comunicación futura entre algoritmos distribuidos, tales como el algoritmo de elección Bully, Anillo logico y el algoritmo de Prueba de Trabajo (Proof of Work).

El modulo esta compuesto unicamente por una clase la cual es **Chat** que extiende de **Thread**, para permitir la ejecución en segundo plano de otros procesos que puedan estar escuchando en caso de nuevos mensajes. Como se observa en la **Figura 19**, contiene los atributos necesarios para establecer y mantener conexión multicast, asi como métodos esenciales para enviar y recibir mensajes.

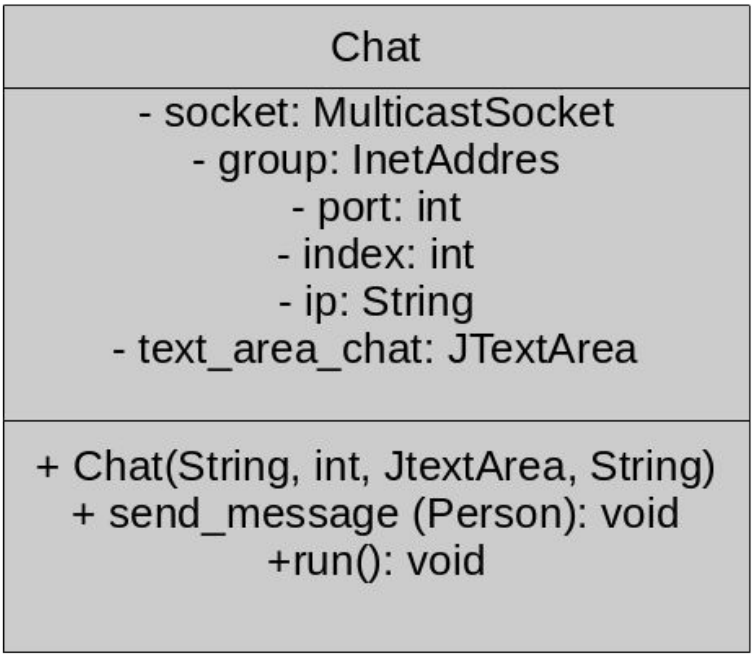

Figura 19: Atributos y metodos de la clase Chat

A continuación se detallan los elementos más importantes de su implementación:

1. Inicialización y unión al grupo multicast

```
this.socket = new MulticastSocket(port);
this.group = InetAddress.getByname(host);
this.socket.joinGroup(group);
```

Bloque de código 1: Inicialización y unión multicast

Como se puede observar en el bloque de Código 1 se crea un socket multicast asociado a un puerto específico. Para que después se obtenga la dirección IP del grupo al que el nodo se quiere unir y finalmente se llama a joinGroup() para que el socket comience a recibir mensajes que lleguen a dicho grupo.

Este paso es muy importante, debido a que permite que todos los nodos se unan a este grupo compartido y puedan recibir los mensajes de cualquier nodo en el mismo canal. Para esto no se requiere de un servidor central ya que el protocolo **UDP** multicast se encarga de entregarlos a todos los participantes.

2. Método send_message: serialización y envío de datos

```
ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream( );
ObjectOutputStream objectOutputStream = new ObjectOutputStream(byteArrayOutputStream);
objectOutputStream.writeObject(person);
byte[ ] data = byteArrayOutputStream.toByteArray( );
this.packet = new DatagramPacket(data, data.length, group, port);
this.socket.send(packet);
this.text_area_chat.append(person.getNickname( ) + ": " + person.getMessage;
```

Bloque de código 2: Método send_message

Como se puede observar en el bloque de Código 2 anterior este método se encarga de tomar un objeto "Person", que contenga el nombre de usuario (nickname) y el mensaje, para de esta forma convertirlo a una secuencia de bytes para su envío por la red. Para esto mismo emplea las clases "ObjectOutputStream" y "ByteArrayOutputStream".

Posteriormente, se crea un "DatagramPacket" que seria el contenedor de datos utilizado por los sockets UDP. El paquete se envía al grupo multicast usando el método "send( )" del socket.

Finalmente, para mostrar el mensaje al usuario se emplea una interfaz gráfica, mediante un "JtextArea", estos es con el fin de que el usuario vea su propio mensaje inmediatamente, sin esperar a que regrese por la red.

3. Método "run": recepción y deserialización de mensajes

```
DatagramPacket packet = new DatagramPacket (new byte[1024], 1024);
this.socket.receive (packet);
String addres = String.valueof (packet.getAddress( ).getHostAddress( ));
if (!address.equals (this.IP)) {
    ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream (packet.getData( ));
    ObjectInpuntStream objectInputStream = new ObjectInputStream(byteArrayInputStream);
    person = (Person) objectInputStream.readObject( );
    message_recived = person.getNickname( ) + ": " + person.getMessage( ) + "\n";
    text_area_chat.append(message_recived);
}
```

Bloque de código 3: Metodo run

Como puede observarse el método “run( )” del Fragmento de Código 3 contiene un bucle infinito que escucha continuamente nuevos paquetes UDP enviados al grupo. Para evitar bloquear la interfaz gráfica principal, esta clase hereda de “Thread” y se ejecuta en segundo plano.

De esta manera cada paquete recibido es procesado de la siguiente manera:

- Se lee la dirección IP del emisor. Si coincide con la IP del propio Nodo, se ignora (esto tiene el propósito de evitar duplicar los mensajes enviados por uno mismo).
- El contenido del mensaje se deserializa usando “ObjectInputStream”, recuperando así el objeto “Person” original.
- Finalmente, el mensaje se muestra en el área de texto del receptor.

Gracias a que se lo realizo de esta manera es que es posible que múltiples usuarios compartan mensajes de manera descentralizada y sin servidor.

4. Manejo de errores

En varios puntos del codigo fue necesario encapsular los posibles errores que se podrian dar , para este y como en cualquier aplicación se emplearon los bloques “try-catch”, como por ejemplo se puede observar en el bloque de código 4 a continuación:

```
catch (Exception e) {
        e.printStackTrace( );
}
```

Bloque de Código 4: Bloque Try - catch

Aunque es una técnica para prototipos y pruebas, en aplicaciones mas grandes o robustas es mas recomendable usar sistemas de manejo de errores más estructurados, tales como inclusión de logs informativos y alertas al usuario en caso de errores de red o deserialización.

4.1.2 Módulo de Coordinación

Como ya se abordo en el capitulo anterior uno de los desafíos más relevantes en la construcción de una red distribuida es la necesidad de establecer un mecanismo de coordinación que, sin comprometer la descentralización, permita la organización temporal de las decisiones críticas del sistema. Para es que se establecieron dos algoritmos clásicos de elección de coordinador: Bully y Anillo.

Para que ambos algoritmos puedan funcionar sin problemas y que además se respete la modularidad es que se decidió que ambos se implementarían en su propia clase (“Bully” y “Anillo”, respectivamente), y ambos extienden la clase “Thread” lo cual les permite ejecutarse en segundo plano de forma autónoma y paralela al resto del sistema. A continuación se analizaran sus componentes y funcionamiento.

Estructura del Módulo

Ambas clases (“Bully” y “Anillo”) tienen una estructura similar:

- Crean un socket multicast y se unen a un grupo de direcciones IP compartidas
- Utilizan “Timer” para programar tareas como el tiempo de espera (“timeout”) para la elecciones o promociones automáticas a coordinador.
- Cada clase mantiene una referencia al área de texto en la GUI (“JtextArea”) donde se reflejan los eventos de elección.
- Manejan internamente un conjunto de condiciones booleanas (“coordinador”, “status”, “is_here_coordinador”, etc.) que permiten tomar decisiones en tiempo de ejecución.

4.1.2.1 Algoritmo de Bully

A continuación se mostraran las partes mas importantes de la clase “Bully” junto a un diagrama UML que muestra sus atributos y métodos.

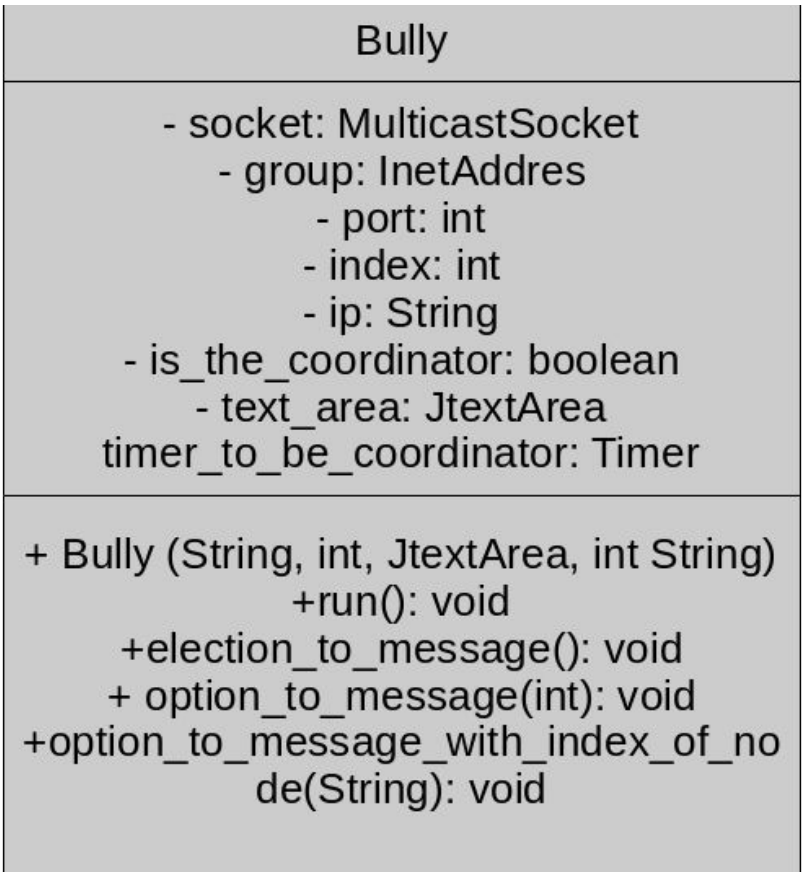

Figura 20: Atributos y métodos de la clase Bully

1. Envío y recepción de mensajes

Cada nodo, al detectar la ausencia de un coordinador, puede enviar un mensaje “Elección” a los nodos con identificadores mayores como se puede apreciar a continuación en el bloque de código 5.

```
message = (“Eleccion” + “ : “ + this.ID).getBytes( );
```

Bloque de Código 5: Asignación a message

En caso de que algún nodo responda, se espera a ver si se declara como coordinador. En caso contrario, el nodo que inició la elección se proclama coordinador.

2. Temporizadores para esperar coordinadores

```
timer_to_convert_in_coordinador.schedule(new TimerTask( ) {
      public void run( ){
            If(!status && !coordinador){
                  // Convertirse en coordinador
            }
      }
}, 1500);
```

Bloque de Código 6: Temporizador

En el temporizador del Bloque de codigo 6 se puede observar que actúa como mecanismo de “timeout”. Si no se recibe respuesta, el nodo se declara líder.

3. Verificación del estado de otros nodos

Cada nodo que recibe un mensaje “Elección” y tiene un ID mayor, responde y se postulo como candidato, mostrando así un ejemplo de competencia distribuida.

4.1.2.2 Algoritmo de Anillo

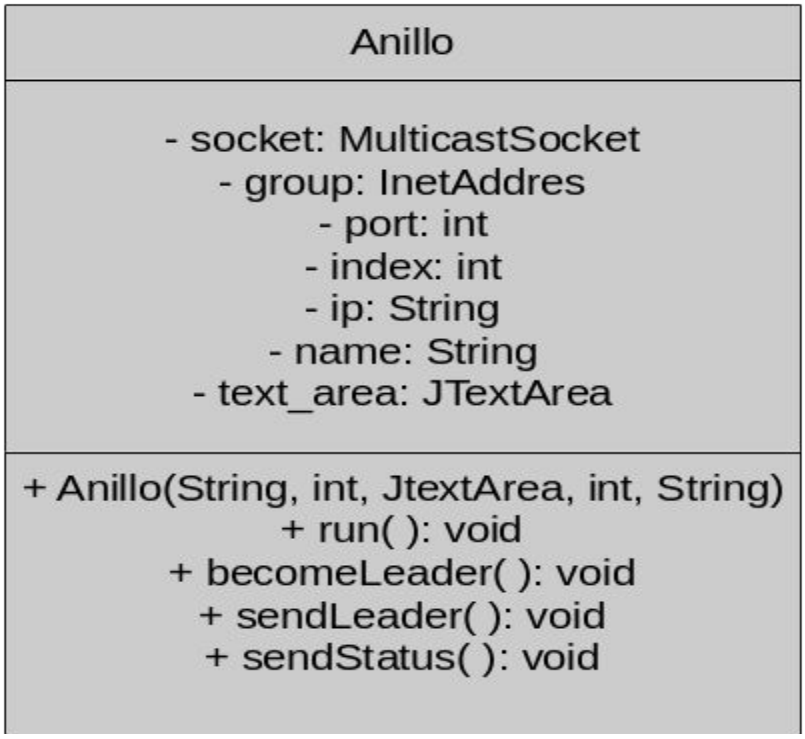

Figura 21: Diagrama de atributos y metodos de la clase Anillo

1. Propagación circular del mensaje

La principal diferencia que tiene en comparación con el algoritmo de bully es que el algoritmo de anillo recorre una lista lógica circular de nodos. De forma que al recibir un nodo el

mensaje de “elección” este lo reenvía al siguiente en la lista como se puede ver a continuación en el Bloque de código 7:

```
DatagramPacket new_packet = new DatagramPacket(…,
InetAddress.getByName(next_nodo), this.port);

Socket.send(new_packet);
```

Bloque de Código 7: Reenvío del mensaje de elección

2. Determinación del líder

Si el mensaje regresa al nodo que tiene el ID más alto, este se declara coordinador y lo anuncia como se puede ver en el Bloque de código 8 inferior:

```
This.text_area_chat_algorithm_anillo.append(“Soy ahora el coordinador:” + ID + “\n”);
options_of_messagess_to_send(3);
```

Bloque de Código 8: Declaración de coordinador

3. Elección automática

Un temporizador (“timer_to_be_coordinador”) se usa para iniciar automáticamente una nueva ronda si no se detecta coordinador:

```
timer_to_be_coordinador.schedule(new TimerTask( ) {
    Public void run( ) {
        If (!is_there_coordinador && !coordinador){
            options_of_messagess_to_send(3);
        }
    }
}, 1200);
```

Bloque de Código 9: Temporizador de nueva ronda

Dicho comportamiento se capturo en el fragmento del Bloque de código 9 presentado.

4.1.3 Módulo de Seguridad con PoW

Como ya se había explicado en la sección anterior, este modulo junto con el que se encuentra a continuación representan los mas difíciles y complejos de implementar dentro de la arquitectura de “*NeoAres”.* Y la razón principal de esto es que su proposito es validar acciones críticas dentro de la red P2P, como la transmisión de datos, la verificación de integridad y la coordinación entre nodos; esto se hace con el proposito de garantizar la autenticidad, fortalecer la resistencia a ataques y el consenso distribuido.

Para esto este módulo fue desarrollado con el objetivo de implementar un mecanismo de confianza sin depender de nodos centrales o jerárquicos.

Debido a su complejidad y a las múltiples acciones que debe ejecutar, se decidió dividir sus responsabilidades en tres clases principales, cuya función se describe a continuación:

Tabla 7: Responsabilidades de las clases del modulo de Seguridad con PoW

Clase	Responsabilidad principal
PowThread	Escuchar, validar y propagar cadenas de bloques en la red multicas
Block	Representar un bloque en la cadena, incluir transacciones, realizar PoW y validar bloques.
Transaction	Modelar las transacciones individuales y validar su integridad y autenticidad.

Diseño y funcionamiento general

La clase “PowThread” extiende de “Thread” y su propósito es manejar la lógica principal de la blockchain local de cada nodo. Da inicio generando transacciones ficticias que luego serán incluidas en una serie de bloques. A partir del bloque génesis, de forma que cada bloque se mina de forma local mediante PoW antes de agregarse a la cadena.

Una vez generada, la cadena se propaga al resto de la red mediante un objeto serializado enviado por UDP multicast. Mientras que a su vez el mismo hilo escucha constantemente por nuevas cadenas enviadas por otros nodos. Y al recibir una, ejecuta una validación rigurosa para asegurar que los bloques sean válidos y respeten el consenso; como se muestra a continuación:

```
Private boolean handleReceivedChain(List <Block> chain) {
    For (int i = 1; i < chain.size( ); i++) {
        Block currentBlock = chain.get(i);
        Block previousBlock = chain.get(i – 1);
        If (!currentBlock.isValidBlock(currentBlock, previousBlock, 3)) {
            Return false;
        }
    }
    BlockchainL = chain;
    Retunr true;
}
```

Bloque de Código 10: Propagación de la cadena

Proceso de minado y validación

La clase “Block” encapsula el mecanismo de Pow mediante su método “mineBlock( )”, el cual consiste en encontrar un “nonce” que al ser procesado junto a los datos del bloque mediante la función hash SHA-256, genera un resultado que comience con una cierta continuidad de ceros (según la dificultad) como se puede ver a continuación:

```
Public String mineBlock( ){
    String target = new String( new char [dificultad]).replace(‘\0’, ‘0’);
    While (!hash.substring(0, dificultad).equals(target)){
        Nonce++;
        Hash = calculateHash( );
        }
        Return hash;
}
```

Bloque de Código 11: Método mineBlock

Este procedimiento implica una alta carga de trabajo computacional, lo cual garantiza que cualquier intento de alterar la información requiera un esfuerzo considerable, dando así una mejora en la seguridad del sistema.

La clase “Transaction” por otro lado no solo se encarga de almacenar la información de origen, destino y cantidad, sino que asegura que sea valida mediante la verificación de la integridad del hash de la transacción y su integridad momentánea con lo cual para este esquema se empleo lo siguiente:

```
Public boolean isValid( ) {
      Long currentTime = new Date( ).getTime( );
      Long transactionTime = timestamp.getTime( );
      Long maxTimeDifference = 5 * 60 * 1000;
      If (currentTime – transactionTime > maxTimeDifference) {
          Return false;
       }
      Return calculateHash( ).equals(transactionHash);
}
```

Bloque de Código 12: Validación

Integración y difusión de la cadena

Una vez se completo el proceso de la minería, los bloques son almacenados y propagados mediante el método “sendChain( )”. Para la propagación se utilizo serialización de objetos Java, empaquetados y enviados por la red multicast UDP. Esta integración permite mantener sincronizada la blockchain en todos los nodos sin necesidad de respuesta directas o una infraestructura centralizada.

El método “handleRecivedBlock( )” dentro de la clase “Block” permite validar y almacenar bloques de manera individual en la cadena local, reforzando el modelo descentralizado cuya implementación se vio de la siguiente manera:

```
Public void handleRecivedBlock (Block block){
      If (getLatesBlock( )==null){
            Blockchain.add(block);
      } else if (isValidBlock (block, getLatesBlock( ), dificultad)) {
            Blockchain.add(Block);
      }
}
```

Bloque de Código 13: Almacenar Bloque

Comunión de las clases

Como este modulo se desarrollo de manera modular (al igual que toda la aplicación de “*NeoAres”)* es de vital importancia mostrar cómo se relacionan e integran las clases que lo componen. Para esto es que a continuación se mostrara una representación UML que muestra gráficamente la estructura del sistema, reflejando la relación directa entre PoWThread, Block y Transaction, así como su responsabilidad en el flujo de generación, validación y propagación de bloques dentro de la red P2P.

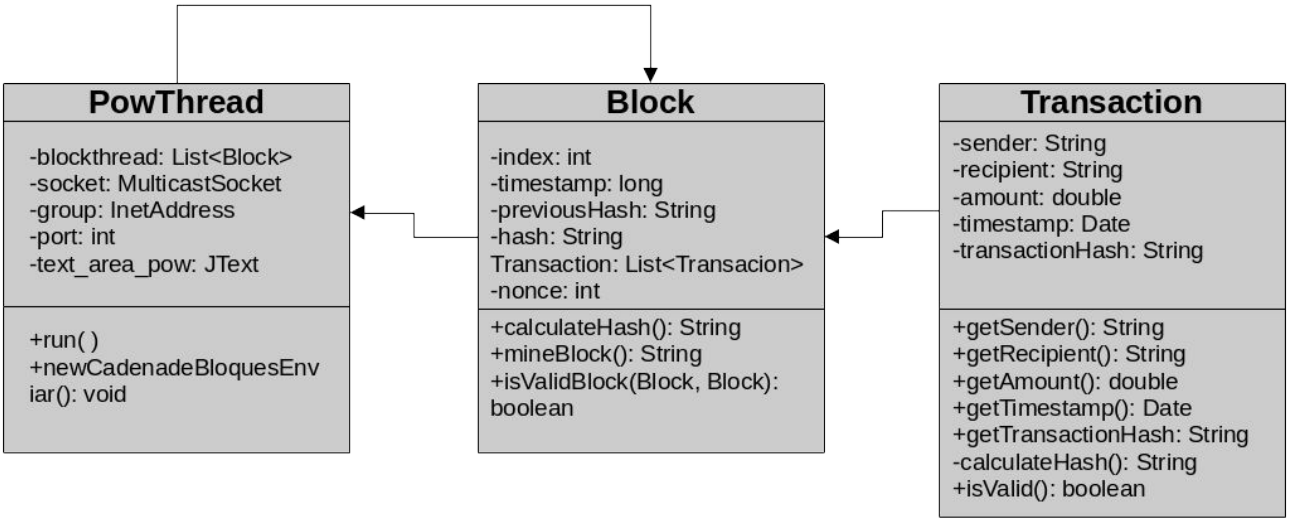

Figura 22: UML del funcionamiento del Algoritmo de PoW en NeoAres

.

4.1.4 Módulo de Archivos Compartidos

Siendo el segundo modulo mas complicado después del modulo de la seguridad con PoW, el módulo de Archivos compartidos constituye uno de los componentes más complejos y significativos de “NeoAres”, ya que permite a los nodos compartir archivos entre sí de forma distribuida, sin recurrir a servidores centrales. Este módulo fue diseñado cuidadosamente para garantizar la integridad de los archivos, la descentralización en la transmisión y la tolerancia a fallos inherentes al paradigma del cómputo distribuido.

Durante el desarrollo, se enfrentaron numerosos desafíos técnicos como el manejo de archivos grandes en fragmentos, sincronización entre nodos, control de errores y la construcción de un sistema robusto que permitiera tanto el envío como la recepción de archivos sin pérdida de datos.

A diferencia del modulo anterior que requirio de tres clases para su correcta implementación, este modulo se consolidó únicamente en una clase principal llamada “Manager_Files”, que al igual que todos los módulos ya vistos extiende a la clase “Thread” y cumple con dos funciones cruciales:

- **Emisor:** Permite a un nodo seleccionar un archivo y enviarlo a la red multicast, dividiéndolo en paquetes UDP.

- **Receptor:** Escucha constantemente en la red para recibir los paquetes de archivos, reensamblarlos y guardarlos localmente.

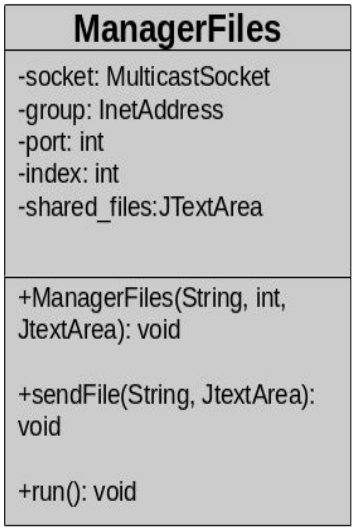

Figura 23: Diagrama UML de la clase Manager_Files

Diseño y funcionamiento general

El fragmento de código siguiente muestra al constructor de la clase “Manger_Files” esta encargado de configurar el cocket multicast, a la hora de unirse al grupo para de esta forma obtener el IP del nodo; :

```
Public Manager_Files(String addres, int port, JtextArea chat_files){
      This.socket = new MulticastSocket (port);
      This.group = InetAddress.getByName(address);
      This.port = port;
      This.socket.joinGroup(group);
      This.IP = String.valueOf (InetAddress.getLocalHost( ).getHostAddress( ));
}
```

Bloque de Código 14: Constructor de clase Manager_Files

El método “send_file(String path, String name)” gestiona la transmisión del archivo. Primero envía un paquete con el nombre del archivo y luego divide su contenido en bloques de hasta 65500 bytes, que son enviados en secuencialmente cuya implementación se muestra en el siguiente fragmento:

```
FileInputStream file_input_stream = new FileInputStream(file);
While ((bytes_readed = file_input_stream.read(buffer)) != -1){
      DatagramPacket packet_data= new DatagramPacket(buffer, bytes_readed, group, port);
      Socket.send(packet_data);
}
```

Bloque de Código 15: Transmisión de archivos

De parte del lado receptor, el método “run( )” espera recibir los metadatos del archivo (nombre), luego escucha los paquetes de datos y los reensambla en un archivo físico como como se puede observar en el siguiente fragmento:

```
While (true) {
     DatagramPacket packet = new DatagramPacket (buffer, buffer.length, group, port);
     Socket.receive(packet);
     If (packet.getLength( ) < 65500) {
          file_output_stream.write(packet.getData( ), 0, packet.getLength( ));
          Break;
      }
      file_output_stream.write (packet.getData( ), 0, packet.getLength( ));
}
```

Bloque de Código 16: Reensable de los paquetes

Un punto importante fue hacer que el receptor tuviera la capacidad de diferenciar los paquete por su IP, para que de esta manera ignorara los que provienen de sí mismo. Además, el módulo asegura que los archivos se reciban en un directorio dedicado (“files_recived”), creándolo de ser necesario.

4.2 Integración de Módulos

Después de haber desarrollo cada uno de los módulos de manera independiente (chat, coordinación, seguridad y archivos compartidos) fue necesario abordar la integración de todos ellos en una única interfaz funcional: la clase principal “Main”. Esta fase representó un punto critico en el proceso de desarrollo, debido a que se debía garantizar que cada componente funcionara de forma armónica dentro de un mismo entorno gráfico y que los recursos se compartirán sin interferencias o conflictos.

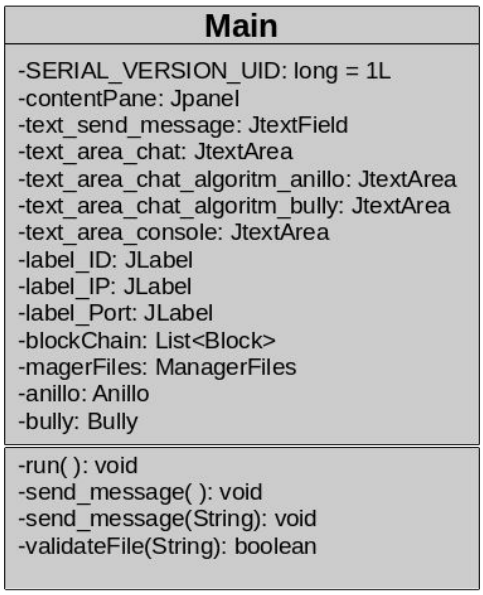

Figura 24: Diagrama UML de la clase principal Main

Desde un principio se adopto la estructura modular basada en el enfoque TDD (Test Driven Development), como se vio desde el capitulo 1. Ya que esta estrategia no solo permitió validar individualmente cada módulo a través de pruebas unitarias y pruebas empíricas en ejecución, sino que también facilito el proceso de la depuración y evolución del sistema. Dividir el sistema en módulos permitió mantener un mayor control sobre el flujo de ejecución y enfocarse en resolver problemas de manera aislada.

4.2.1 Principales problemas durante la integración

Pese a que la modularidad nos permitió desarrollar cada parte de la aplicación de manera independiente para que de esta manera su desarrollo fuera mas sencillo, también trajo consigo la dificultad de que al unir los módulos en un solo proyecto estos chocaran entre si, revelando de esta manera lo complejo que puede llagar a ser el computo distribuido en ambientes reales:

- Los **algoritmos de elección** (Anillo y Bully), aunque funcionales por separado, no se activaban correctamente cuando se inicializaban dentro de la misma aplicación. Esto se debió principalmente a la superposición de mensajes en el canal multicast, lo cual fue resuelto implementando mecanismos de aislamiento por puertos específicos.
- El módulo **chat**, algunos mensajes se repetían o llegaban en un orden inconsistente. Esto se mitigó ajustando la comparación de direcciones IP de origen y asegurando que cada nodo filtrara correctamente los mensajes provenientes de sí mismo.

- El módulo de **PoW (Proof of Work)** debido a su carga computacional, congelaba la interfaz su se ejecutaba sin restricciones. La solución fue encapsular la lógica de minería en hilos separados y aislar las actualizaciones de la cadena de bloques para evitar bloqueos de la interfaz.
- Durante el desarrollo del módulo de **archivos compartidos** se observo que los archivos enviados a través de la red llegaban corruptos. El problema estaba en la fragmentación y reconstrucción de datos, que fue solucionado controlando manualmente los tamaños de los paquetes UDP y gestionado adecuadamente el cierre del flujo de escritura.

4.2.2 Integración

La integración de los módulos comienza con el método “start_objects( )” el cual instancia y lanza los cinco módulos principales como se puede ver en el siguiente bloque de código:

```
Peer = new Chat (host, port, text_area_chat, IP);
Maneger = new Manager_Files (host, port + 500, text_area_chat_files);
Anillo = new Anillo (host, port + 1000, text_area_chat_algoritmo_anillo, ID, IP);
Bully = new Bully (host, port + 1500, text_area_chat_algoritmo_bully_ ID, IP);
Pow = new PowThread (host, port + 2000, text_area_pow);
```

Bloque de Código 17: Fragmento del método start_objects( )

A cada uno se le asigna su propio puerto, lo que evita colisiones en la red multicast. Luego, cada hilo es lanzado con “.start( )” permitiendo su ejecución concurrente.

Interfaz unificada y manejo de eventos

El diseño gráfico agrupa visualmente los módulos en distintas secciones de la ventana principal (“Jpanel”), usando diferentes colores de fondo para diferenciarlos. Los mensajes de los módulos se muestran en su propia área (“JtextArea”) facilitando lo depuración visual y la compresión del estado del sistema.

También y para mayor comodidad para los usuarios se implementaron los botones dedicados a activar comportamientos específicos, como las elecciones en los algoritmos de coordinación o enviar archivos. Además claro un botón para enviar un bloque PoW , como se presenta en el siguiente fragmento:

```
Powthread.addActionListener (new ActionListener( ) {
      Public void actionPerformed (ActionEvent e) {
            Pow.newCadenadeBloquesEnviar( );
      }
});
```

Bloque de Código 18: Listener del botón para nuevo bloque

Manejo de eventos críticos

Durante el desarrollo del proyecto se estableció un botón “Salir” para permitir la desconexión controlada del nodo. Esto fue especialmente importante para mantener la coherencia del sistema distribuido, ya que el resto de nodos debía ser notificado de la salida del coordinador, dicha implementación se muestra en el siguiente fragmento:

```
btn_exit.addActionListener (new ActionListener( ) {
    Public void actionPerformed (ActionEvent e) {
          Bully.option_to_send_message(4);
          System.exit(0);
    }
});
```

Bloque de Código 19: Acción para salir

Resultados de la Integración

Gracias a todos estos ajustes y diseños se logro que el sistema tuviera una ejecución fluida y estable donde cada módulo conserva su autonomía, pero colabora para alcanzar un objetivo común.

Con esto además se demuestran los beneficios del diseño modular aplicado a un entorno distribuido: facilita el mantenimiento, favorece la estabilidad y mejora el proceso de prueba y error.

4.3 Validación del Sistema

Al termina la integración de los diferentes módulos del sistema *NeoAres* fue obligatorio realizar una serie de pruebas para validar que el comportamiento conjunto de todos sus componentes respondiera a los objetivos iniciales del planteamiento del proyecto. Para esto se llevaron a cabo diversas pruebas de integración, carga y tolerancia a fallos, que permitieron verificar tanto la funcionalidad como la estabilidad de la red.

4.3.1 Pruebas de Integración

Como se observo en la sección anterior la parte final del desarrollo fue la integración y por lo tanto su correcta verificación fue clave para encontrar posibles errores que se pudieran dar al intentar hacer que todos los módulos funcionaran unos con otros sin robarse recursos o causarse interferencias:

Tabla 8: Pruebas realizadas durante la integración

Módulo Evaluado	Prueba Realizada	Resultado Obtenido	Observación
Chat + Bully	Envió de mensajes mientras se ejecuta la elección.	Mensajes entregados de manera correcta	Al inicio se repetían los mensajes pero se corrigió comparando la IP.
Chat + Anillo	Envió de mensajes durante la ronde de Anillo.	Funciona correctamente	Se sincronizo el temporizador del anillo
PoW + Chat	Minar bloque mientras se conversa.	Congelamiento temporal de la interfaz.	Se aíslo el proceso de PoW en hilo independiente.
Archivos + PoW	Envío de archivo mientras se mina.	Archivos parcialmente corruptos.	Se implementó pausa en envío tras validación PoW.
Coordinación + Archivos	Elección de líder durante transferencia.	Transferencia completada sin pérdida.	

Estas pruebas permitieron identificar errores como repetición de mensajes multicast, fallos en la detección de coordinador y trabas en la interfaz gráfica durante la minería de bloques. Cada uno de estos fue corregido mediante la reestructuración de timers, validación de IP de origen y ajustes en la visualización asincrónica.

4.3.2 Comportamiento Bajo Carga

Con el sistema estable, se procedió con la siguiente prueba en la cual se simularan múltiples nodos en paralelo ejecutando los módulos de manera continua:

- Se instanciaron hasta cinco nodos ejecutando el sistema completo.
- Se enviaron archivos de distintos tamaños (desde 500KB hasta 5MB).
- Se iniciaron elecciones simultáneas con Anillo y Bully

- Se generaron bloques PoW en paralelo.

Tabla 9: Resultados de Pruebas de Carga

Número de nodos	PoW Activo	Tamaño del Archivo	Tiempo promedio de transferencia	Latencia en Chat	Observaciones
3	NO	500 KB	2.1 s	Baja	Todo funciona de forma fluida.
4	SI	1 MB	4.3 s	Media	CPU alta en nodo minado.
5	SI	5 MB	10.2 s	Alta	Chat retrasado, PoW acaparó recursos.
5	NO	5 MB	6.0 s	Baja	Mejora significativa sin PoW activo.

Como puede observarse, debido a la alta demanda de recursos que requiere PoW, este puede llegar a causar degradación en la recepción de archivos si ambos proceso se encuentran activos al mismo tiempo.

4.3.3 Comprobación de Tolerancia a Fallos

Para esta prueba se simularon desconexiones repentinas de nodos, con el propósito de observar las reacciones del sistema ante estos acontecimientos.

Tabla 10: Resultados de Tolerancia a fallos

Escenarios	Resultados	Comportamiento esperado
Nodo coordinador se desconecta (Bully)	Nuevo coordinador es elegido	Sin interrupción en modulos.
Nodo que mina se desconecta (PoW)	Resto de nodos continúa activos.	Sin bloque corrupto
Nodo que envía un archivo se desconecta	Trasferencia incompleta	Mejor validación de conexión
Nodo de chat se desconecta	Mensajes no se pierden	Los demás nodos mantiene la funcionalidad

Gracias a estos resultados se pudo comprobar que el sistema puede conservar su correcto funcionamiento pese a que algunos nodos se desconecten o interrumpan sus procesos, lo cual hace que sistema demuestre sus propiedades como parte del paradigma de computo distribuido (autonomía, descentralización y tolerancia a fallos)

4.3.4 Validación en Contenedores Docker

Finalmente para una validación más robusta y realista del entorno distribuido, se utilizo Docker para desplegar múltiples nodos en contenedores aislados. Gracias a esto se pudo simular una red P2P en un entorno controlado, replicando escenarios con múltiples Ips virtuales.

- Cada contenedor Docker contenía una instancia del sistema NeoAres.
- Se configuro una red Docker personalizada para permitir la comunicación multicast entre los distintos nodos.
- Para facilitar las cosas se programaron unos scrips que lanzaban elecciones, enviaban archivos y activaban el minado PoW.

Tabla 11: Resultados de Pruebas en Docker

Contenedor	Sistema Operativo	Módulos Activos	Resultado General	Observación Técnica
Nodo 1	Windows 10	Chat, Bully, PoW	Correcto	Requiere mas RAM para PoW
Nodo 2	Windows 10	Archivos, Chat	Correcto	Sistema estable para pruebas
Nodo 3	Debian 12	Todos	Correcto	Archivos correctamente guardados en /files_recived
Nodo 4	Linux Mint	Chat, Archivos, Anillo	Correcto	Detección correcta de directorios y permisos

Con estos resultados se pudo observar que el sistema funciona correctamente en diferentes sistemas operativos, además de ver su estabilidad en un escenario distribuido real, además de que se pudieron observar algunas posibles mejoras que se podrían implementar mas adelante.

Capitulo 5. Pruebas y Resultados

A lo largo de todo este trabajo se ha detallado el diseño, implementación y validación modular del sistema distribuido “NeoAres”. Pero para cerciorarse de que el sistema pueda ser implementado en un entorno real fue necesario realizar pruebas a una escala mucho mas grande que simule un entorno distribuido real.

Por lo tanto, en este capítulo se ilustran los resultados obtenidos de dichas pruebas, ejecutadas en múltiples nodos con diferentes sistemas operativos, donde se evaluaron los aspectos mas importantes como por ejemplo la comunicación entre nodos, la propagación de los bloques, eficiencia en la trasferencia de archivos y la robustez de los algoritmos de coordinación. De esta forma los datos obtenidos, permitieron analizar el rendimiento y la tolerancia a fallos del sistema.

Además, como cierre de este trabajo se incluyen un apartado de conclusiones generales donde se reflexiona sobre el alcance del proyecto, el cumplimiento de lo propósitos originales del proyecto (construir una aplicación que permita comprender, mediante la práctica, los fundamentos del cómputo distribuido) y los aprendizajes obtenidos durante el desarrollo del proyecto.

5.1 Pruebas a Gran escala

Pese a que en el capitulo anterior ya se realizaron pruebas de validación, estaban mas enfocadas en rectificar la correcta integración de los distintos módulos (chat, algoritmos de coordinación, PoW y sistema de archivos compartidos), por lo que en esta sección se busco simular una red compuesta por una gran cantidad de nodos distribuidos virtualmente mediante Docker. Esto tuvo el propósito de evaluar cómo se comportaría la aplicación bajo el estrés de manejar un escenario realista de escalabilidad, para esto además también se quiso ver como reaccionaria a un uso intensivo y concurrencia alta (con nodos operando de manera autónoma, comunicándose, participando en elecciones de coordinación y validando peticiones).

Objetivos de la Simulación

- Verificar la escalabilidad del sistema bajo una topología de gran tamaño.
- Evaluar el rendimiento de cada nodo con sistemas operativos distintos.

- Medir la tolerancia a fallos al desconectar nodos aleatorios durante la ejecución.
- Analizar la latencia promedio en comunicaciones y validación.

Metodología de la prueba

La simulación se llevó a cabo utilizando múltiples contenedores Docker, que representaron a los nodos, de forma que cada contenedor replicaba la ejecución completa de la aplicación, utilizando puertos configurados dinámicamente.

- **Entorno Docker:** 100 contenedores individuales.
- **Sistemas base:** Imagen Debian 12.
- **Recursos simulados por contenedores:** 1 CPU, 512 MB RAM.
- **Roles por nodo:**
  - Todos los nodos deben enviar un mensaje cada 2 minutos.
  - 30 enviaran archivos
  - El nodo que sea líder debe desconectarse al pasar 3 minutos para empezar una elección.
  - 20 envían paquetes a verificación con PoW, constantemente para comprobar su impacto en el sistema.
- **Mensajes y archivos generados:** Se simularon eventos controlados con scrips de prueba automatizada para imitar acciones de usuarios.

Para que cada parte de la aplicación fuera evaluada de mejor manera, se establecieron 5 categorías de métricas principales:

1. Latencia de mensajes
2. Tiempo de convergencia de coordinadores
3. Trasferencia e integridad de archivos
4. Rendimiento del Algoritmo PoW
5. Consumo de recursos por Nodo

Cada una fue evaluada en cinco escenarios distintos con: 10, 25, 50, 75 y 100 nodos activos.

Esto fue hecho con el propósito de representar una red en la cual se van sumando mas nodos activos con el pasar de tiempo, para ver que la aplicación fuera escalable y mantuviera una métricas aceptable pese al aumento de flujo.

5.2 Resultados de la Prueba a Gran Escala

Latencia Promedio de Mensajes

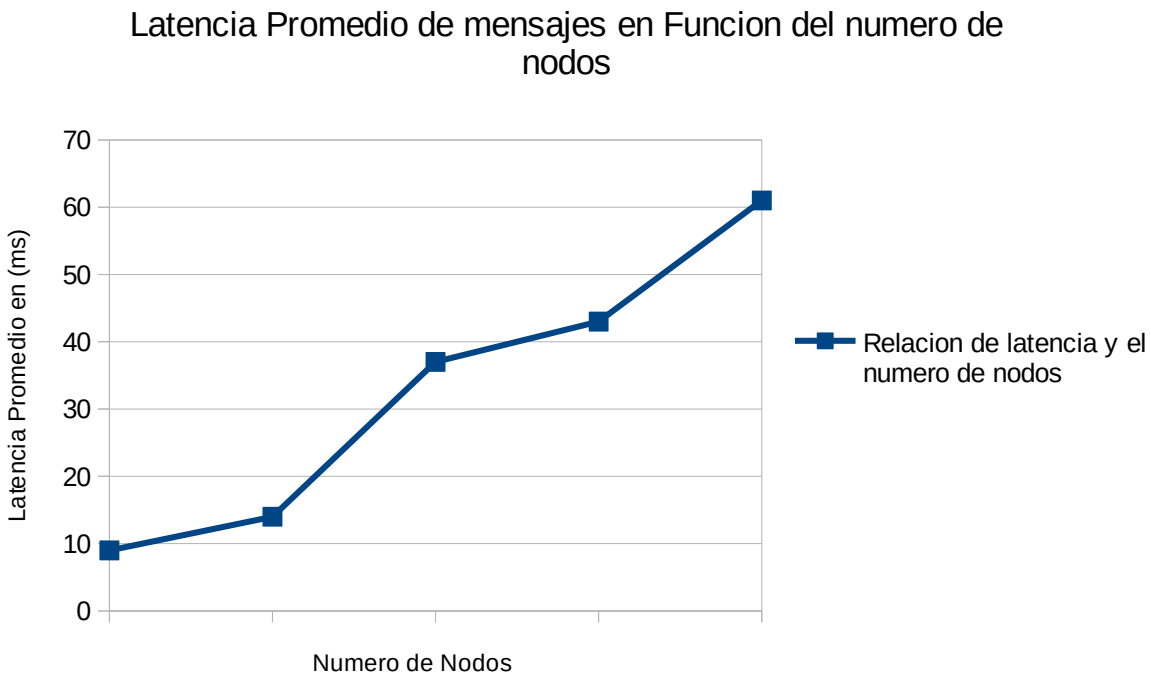

Figura 25: Gráfica de latencia promedio en el envío de los mensajes

Como puede observarse en la grafica superior, el crecimiento de la latencia fue casi lineal en relación con la cantidad de nodos que se iban agregando. Lo cual fue una señal positiva, ya que demostró que el sistema soporta un numero creciente de nodos sin deteriorarse abruptamente.

En particular, el promedio de latencia pasó de 9ms con 10 nodos a 61 ms con 100 nodos, lo cual representa un incremento gradual y predecible; con lo cual se puede inferir que el sistema es escalable y no tiene problemas con cuellos de botella significativos en la trasmisión de mensajes.

En resumen estos resultados fueron un buen inicio en nuestras evaluaciones, ya que demostró que el sistema era capaz de manejar un número creciente de equipos sin deteriorarse abruptamente, demostrando una vez mas la eficiencia de los entornos distribuidos.

Tiempo de Convergencia de los Algoritmos de Coordinación

Se analizo el tiempo de respuesta ante la desconexión simulada del nodo coordinador de ambos algoritmos.

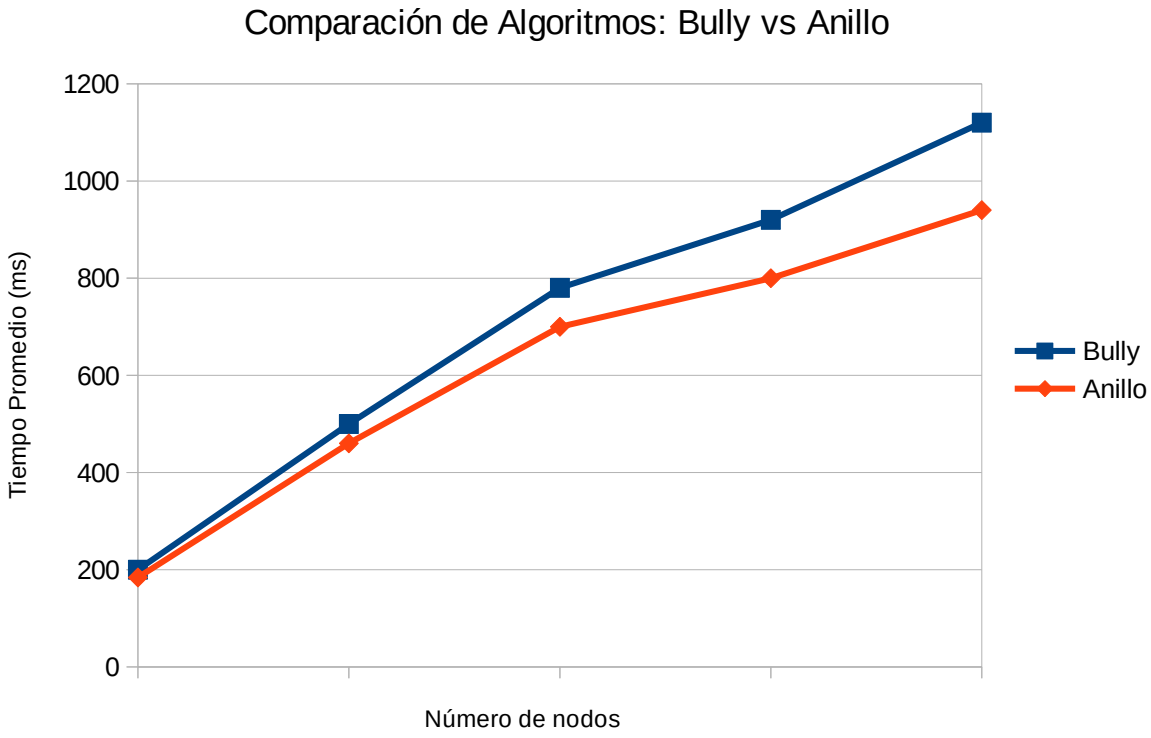

Figura 26: Gráfica de comparación de los algoritmo Bully y Anillo

Y en base a los datos que arrojo la simulación su pudo observar una notable diferencia en los comportamientos de los dos algoritmos de elección ya que conforme aumentaba el numero de nodos en la red el algoritmo de Anillo demostró tener una mayor eficiencia en términos de tiempo promedio, especialmente en los escenarios con una alta carga. Por ejemplo, en el caso cuando había 100 nodos activos y el nodo líder se desconectó, Anillo alcanzo un tiempo promedio de 940 ms, mientras que Bully requirió 1120 ms. Esta diferencia nos sugirió que

Anillo escala mejor en entornos distribuidos con muchos participantes, este se debe seguramente su estructura secuencia y con menos cantidad de mensajes redundantes.

En comparación Bully mostró una mayor precisión y rapidez en la identificación del nodo con el ID mas alto. Esta característica puede llegar a ser mas ventajosa en sistemas donde la jerarquía o el poder de procesamiento del nodo líder es critico.

Transferencia e Integridad de Archivos

En este apartado originalmente solo se analisarian con archivos de tamaño de 1MB, 5MB Y 10MB pero se considero que no seria suficiente para saber realmente si los archivos grandes tendrían a sufrir de corrupción debido al método de trasferencia que se empleo. Por esta razón y para un análisis mas completo es que se agregaron dos mas de mayor tamaño (20MB y 50MB).

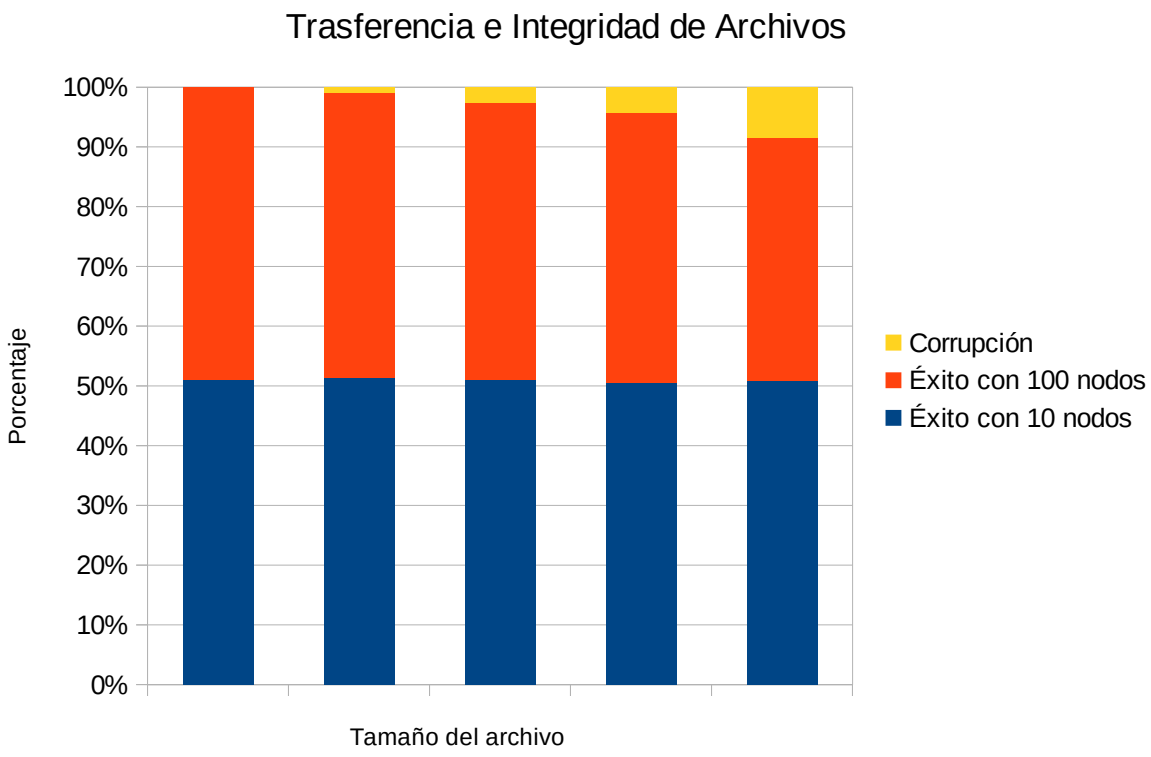

Figura 27: Gráfica de resultados de la trasferencia de archivos con diferentes volúmenes

Gracias a que se extendió la prueba a archivos de mayor tamaño se pudo observar que la tasa de corrupción aumento ligeramente. Con mas de 100 nodos, la tasa de éxito cayo al

85% para archivos de 20 MB y al 72% para los de 50MB mientras que la corrupción aumentó a 8% y 15% respectivamente, este comportamiento confirma que, a medida que el tamaño del archivo crece, el sistema enfrenta mayores desafíos relacionados con la fragmentación de paquetes, pérdida de segmentos y saturación de buffers.

Sin embargo y pese a estas limitaciones el sistema mantuvo una tasa de éxito razonable incluso con archivos de 50 MB, lo cual demuestra una capacidad de tolerancia aceptable. Sin embargo, estos resultados sugieren que para aplicaciones que requieren alta fiabilidad en la transferencia de archivos grandes, sería recomendable implementar mecanismos adicionales como verificación de integridad (hashes), retransmisión selectiva o incluso considerar protocolos alternativos como TCP o QUIC.

Rendimiento del Algoritmo PoW

La minería de bloques se realizo con una dificultad fija (dificultad = 3) y se midio principalmente el tiempo en que un nodo lograba minar un bloque válido.

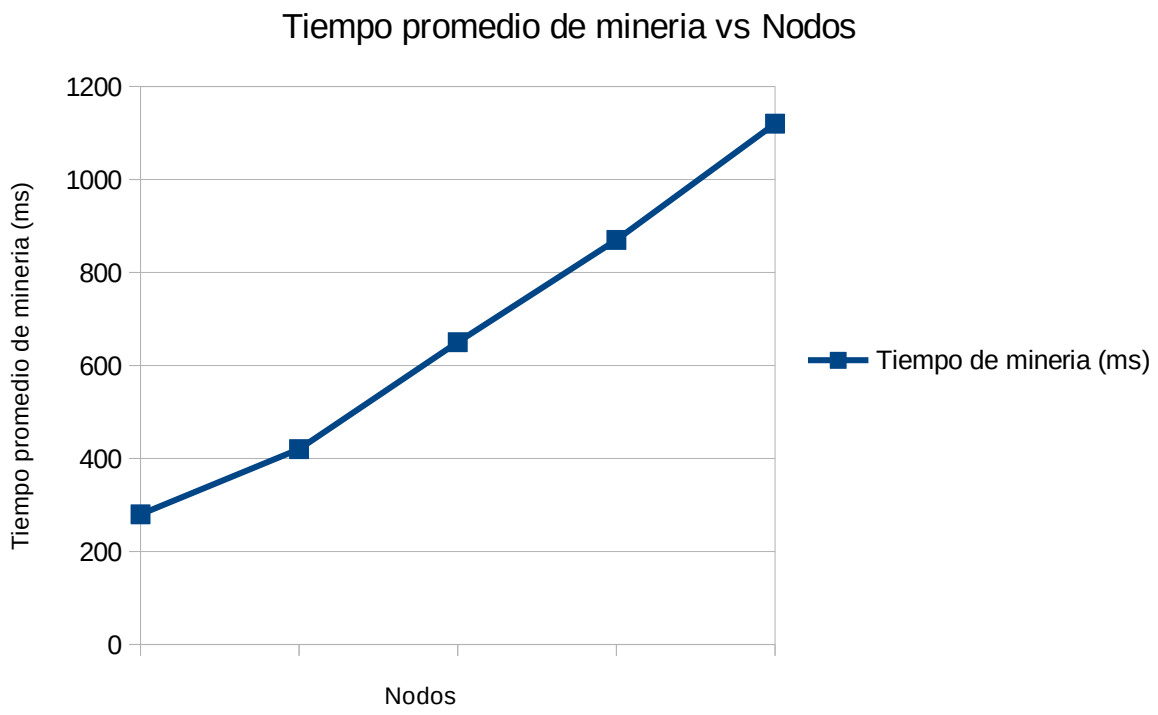

Figura 28: Gráfica de tiempo promedio de mineria

El algoritmo de prueba de trabajo (PoW) mostró un comportamiento intensivo en recursos conforme se incrementó el número de nodos en la red. A medida que más nodos participaron en la minería, el tiempo promedio para encontrar un bloque válido aumentó de forma significativa, pasando de 280 ms con 10 nodos a 1120 ms con 100 nodos. Esta tendencia refleja el aumento en la competencia por resolver el desafío criptográfico, incluso con una dificultad fija.

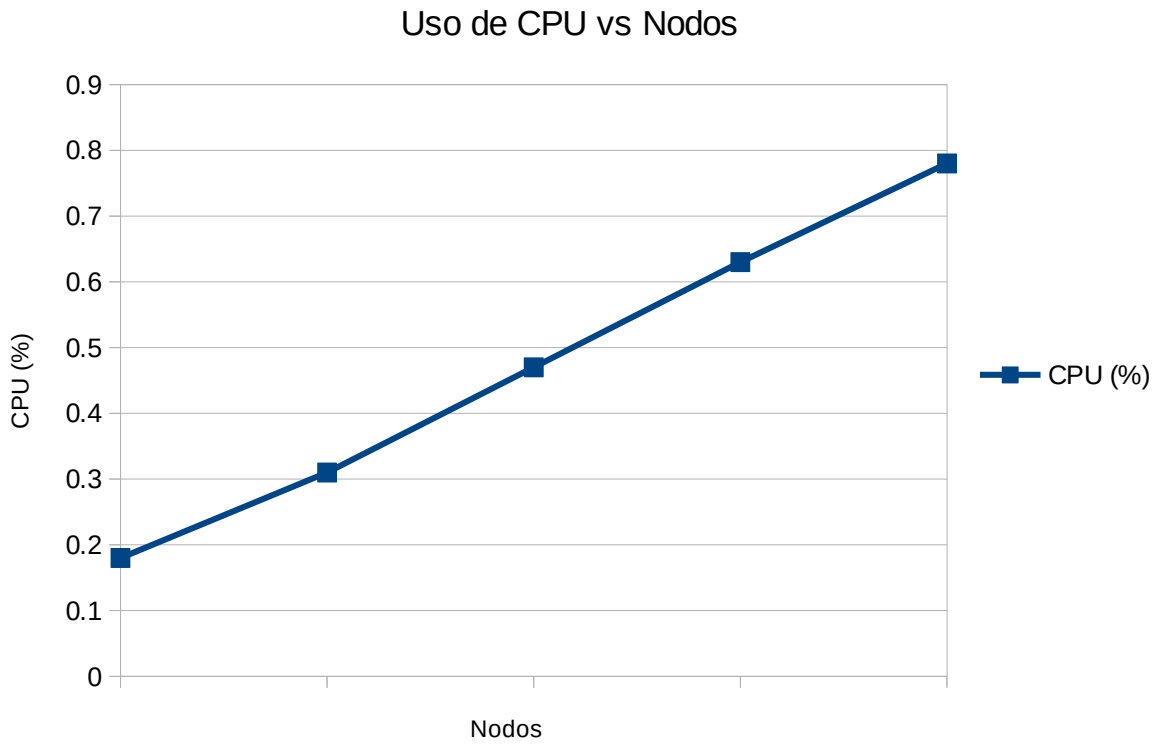

Figura 29: Grafica del rendimiento del CPU

En paralelo, el uso de CPU también se incrementó de manera proporcional, alcanzando un 78% de utilización promedio con 100 nodos. Este nivel de consumo puede generar cuellos de botella, especialmente en entornos con recursos limitados, como contenedores ligeros o dispositivos embebidos.

Mientras se realizaban la pruebas se pudo observar ciertos bloqueos temporales en algunos nodos de la simulación, los cuales debido a que tenia menor capacidad tuvieron mayores problemas en completar la prueba, lo cual comprobó que PoW es el modulo del sistema mas exigente en lo que respecta a recursos computacionales.

Recursos y Eficiencia General

Finalmente se analizo el uso general de recurso de la aplicación en general en lo cual se revelo un crecimiento progresivo en el consumo de memoria RAM y en la cantidad de hilos utilizados por nodo conforme se incrementa el tamaño de la red. Con 10 nodos, el uso promedio de RAM fue de 60 MB por nodo, mientras que con 100 nodos aumentó a 91 MB. Este incremento era esperable, ya que cada nodo debe manejar más conexiones y procesos simultáneos.

Tabla 12: Tabla comparativa de uso de recursos

Nodos	Uso promedio de RAM por nodo (MB)	Thread por nodo	Hilos bloqueados
10	60	4	0
50	75	5	1
100	91	6	2

Asimismo, el número de threads por nodo creció de 4 a 6, reflejando la necesidad de paralelizar tareas como minería, recepción de archivos y mantenimiento de la red. Sin embargo, también se detectaron hilos bloqueados en configuraciones con 50 y 100 nodos, lo cual indica problemas de sincronización o competencia por recursos compartidos.

Una observación crítica fue la aparición de deadlocks temporales, causados por la compartición de buffers entre los hilos encargados del algoritmo PoW y los de recepción de archivos. Esta situación fue resuelta al separar estos procesos en hilos independientes, lo que mejoró la estabilidad del sistema.

Todo esto nos indico que aunque ya se habían realizado mejoras con anterioridad la aplicación aun requería de ciertos ajustes a la hora de tratar con una mayor carga de nodos activos.

5.3 Conclusiones de la pruebas

Como puede observarse en la seccion anterior las pruebas realizadas evidenciaron que NeoAres cumple con los principios fundamentales del cómputo distribuido: autonomía, descentralización, tolerancia a fallos y ausencia de punto único de falla. A lo largo de las

simulaciones, el sistema demostró un comportamiento estable y predecible incluso bajo condiciones de alta carga.

- El sistema fue capaz de mantener la operatividad con hasta **100 nodos activos**, sin interrupciones críticas.
- La integración de todos los módulos (PoW, transmisión de archivos, elección de líder, etc.) no generó conflictos fatales.
- La **modularidad del diseño** permitió identificar cuellos de botella y aplicar mejoras específicas, como la separación de hilos para evitar bloqueos.
- La **latencia promedio** creció de forma casi lineal con el número de nodos, lo cual es un indicador positivo de escalabilidad.
- El sistema logró mantener una **alta tasa de éxito en la transferencia de archivos**, incluso con tamaños de hasta **50 MB**, aunque con un aumento proporcional en la corrupción de datos.

Pero también se pudo identificar que tenia limitaciones importantes que futuros trabajos que quieran usar este proyecto como base podrían abordar y mejorar.

- El algoritmo PoW pese a ser una pieza fundamental para entender la programación distribuida también fue el componente más demandante en términos de uso de CPU y tiempo de procesamiento, alcanzando hasta 78% de uso de CPU y 1120 ms de tiempo promedio de minería con 100 nodos.
- La fragmentación de archivos grandes en entornos con muchos nodos provocó pérdidas parciales de datos. Aunque el sistema mantuvo una tasa de éxito aceptable, se recomienda implementar mecanismos de verificación de integridad y retransmisión selectiva.
- El algoritmo Bully mostró eficiencia en la elección del nodo líder, pero generó duplicación de mensajes en escenarios caóticos, lo cual podría optimizarse con filtros o control de concurrencia.

Sin embargo y aun con todo esto los resultados validan la arquitectura de NeoAres como una solución funcional y adaptable para entornos distribuidos.

Además para complementar lo ya visto asta ahora se agregara a continuación una ultima tabla que muestra los beneficios de NeoAres en comparación con una arquitectura tradicional

cliente-servidor bajo un escenario de 50 nodos; con el propósito de reforzar la elección de una arquitectura descentralizada.

Tabla 13: Comparación de arquitectura cliente-servido y NeoAres

Metrica	Cliente - Servidor	NeoAres (P2P)
Punto único de fallo	Si	No
Escalabilidad	Media	Alta
Latencia con 50 nodos	35 ms	27 ms
Costo computacional del servidor	Alto	Distribuido
Tolerancia a fallos	Baja	Alta
Balance de carga	Centralizado	Distribuido
Flexibilidad de crecimiento	Limitada	Dinámica

.

Como puede verse en una arquitectura cliente-servidor, el servidor central se convierte en un cuello de botella natural: a medida que aumenta el número de clientes, también lo hace la carga sobre el servidor, lo que puede provocar latencias elevadas, caídas del servicio o incluso pérdida de datos si el servidor falla. Por el contrario NeoAres, al estar basado en una arquitectura P2P, distribuye tanto la carga como la responsabilidad entre todos los nodos. Esto elimina el punto único de fallo y permite que el sistema escale de forma más natural.

Conclusiones

A lo largo del desarrollo de este proyecto titulado “NeoAres: Aplicación P2P como guía práctica del cómputo distribuido”, se ha logrado cumplir de forma integral con los objetivos propuestos, tanto en el aspecto técnico como en su enfoque pedagógico.

Desde el inicio, se planteó el reto de diseñar e implementar una aplicación basada en el paradigma de las redes Peer-to-Peer (P2P), con el doble propósito de demostrar en la práctica los principios del cómputo distribuido y de servir como material didáctico útil para su enseñanza. Tras un proceso de análisis, diseño modular, desarrollo con enfoque TDD (Test Driven Development) y validación con pruebas reales y simuladas, se puede afirmar que ambos propósitos han sido alcanzados de manera satisfactoria.

En primer lugar, el sistema implementado cumple con los principios fundamentales del cómputo distribuido: descentralización, autonomía de los nodos, tolerancia a fallos y escalabilidad. Estos principios se vieron reflejados en cada uno de los módulos del sistema: el chat distribuido, los algoritmos de elección de coordinador (Bully y Anillo), la validación mediante Proof of Work, y el sistema de archivos compartidos. Cada uno de estos componentes se desarrolló de forma independiente y luego fueron integrados, enfrentando y resolviendo diversos desafíos técnicos, como la sincronización de procesos concurrentes, la gestión eficiente de recursos, la validación de datos distribuidos y la interoperabilidad entre nodos heterogéneos.

A nivel de validación, el sistema fue probado en múltiples escenarios, tanto de forma local como distribuida mediante Docker, simulando hasta 100 nodos con distintos sistemas operativos. Las métricas obtenidas en dichas pruebas demostraron que el sistema puede escalar eficientemente, manejar la comunicación concurrente entre pares y tolerar fallos de nodos sin comprometer el funcionamiento global. Las gráficas y tablas presentadas en el capítulo 5 respaldan empíricamente el cumplimiento de los objetivos técnicos de rendimiento, consistencia y disponibilidad.

Desde el punto de vista pedagógico, NeoAres ha logrado plasmar en código los conceptos teóricos estudiados en el estado del arte, tales como mecanismos de consenso, algoritmos de coordinación, modelos de comunicación sin servidor central y pruebas de resistencia a fallos. El sistema resultante no solo es funcional, sino que está documentado y estructurado

de tal forma que puede ser fácilmente reutilizado, extendido y comprendido por otros estudiantes o profesionales que deseen adentrarse en el mundo de los sistemas distribuidos. Se integraron diagramas UML, ejemplos detallados, fragmentos explicados de código y pruebas comentadas, consolidando así un material con alto potencial didáctico.

Asimismo, se desarrolló con una arquitectura modular que favorece la mantenibilidad del sistema y permite incorporar nuevas funcionalidades sin afectar la estructura existente. Esto no solo refleja buenas prácticas de desarrollo, sino que refuerza la comprensión del principio de separación de responsabilidades tan importante en el diseño de sistemas complejos distribuidos.

Finalmente, este trabajo no solo representa la construcción de una red funcional P2P, sino también la creación de un puente entre la teoría y la práctica en el ámbito del cómputo distribuido. Se ha demostrado que es posible desarrollar herramientas que además de cumplir una función técnica, tengan un alto valor educativo y contribuyan a la formación de nuevas generaciones de desarrolladores e ingenieros con una comprensión profunda de los sistemas descentralizados.

En conclusión, NeoAres representa un esfuerzo exitoso y completo en la materialización de los principios del cómputo distribuido y en el diseño de herramientas que favorecen su enseñanza y comprensión. Se recomienda continuar con esta línea de trabajo incorporando más algoritmos distribuidos (como Kademlia o Chord), ampliando el sistema a nivel de red pública y desarrollando interfaces más intuitivas, con el fin de seguir enriqueciendo su alcance académico y profesional.

Bibliografía

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2021). Distributed Systems: Concepts and Design (5th ed.). Pearson.

Liu, C., Zhang, F., & Yuan, W. (2022). Performance optimization techniques for P2P distributed systems. Journal of Parallel and Distributed Computing, 155, 45-59.

Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., & Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. IEEE Communications Surveys & Tutorials, 7(1), 72-93.

Kshemkalyani, A. D., & Singhal, M. (2011). Distributed Computing: Principles, Algorithms, and Systems (2nd ed.). Cambridge University Press.

Beck, K. (2003). Test-Driven Development: By Example. Addison-Wesley Professional.

Deitel, H. M., & Deitel, P. J. (2011). Java: How to Program (9th ed.). Prentice Hall.

Richards, M., & Nair, N. (2020). Learning Java: An Introduction to Real-World Programming with Java. O'Reilly Media.

Maymounkov, P., & Mazieres, D. (2002). Kademlia: A peer-to-peer information system based on the XOR metric. Proceedings of the 1st International Workshop on Peer-to-Peer Systems (pp. 53-65).

Schollmeier, R. (2001). A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. Proceedings First International Conference on Peer-to-Peer Computing, 101–102. https://doi.org/10.1109/P2P.2001.990434

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., & Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking, 11(1), 17-32.

Astels, D. (2003). Test-Driven Development: A Practical Guide. Prentice Hall.

Tanenbaum, A. S., & Van Steen, M. (2017). Distributed Systems: Principles and Paradigms (3rd ed.). Pearson.

Huhns, M. N., & Singh, M. P. (2018). Readings in Agents (3rd ed.). Morgan Kaufmann.

Bandyopadhyay, S., Giannella, C., Maulik, U., Kargupta, H., Liu, K., & Datta, S. (2006, July 22). Clustering distributed data streams in peer-to-peer environments. Information Sciences, 176(14), 1952-1985. https://www.sciencedirect.com/science/article/abs/pii/S0020025505003038

Calderon, A. d. J. R., Cañas, G. A., & Perez, S. B. (2014, Enero 1). Las simulaciones, una alternativa para el estudio de los protocolos P2P. ReCIBE. Revista electronica de Computación, Informatica Biomédica y Electronica, 1(1). https://www.redalyc.org/articulo.oa?id=512251566002

Larios G., M. (2019, enero). Sistemas Distribuidos. BUAP. https://correobuap-my.sharepoint.com/personal/mariano_larios_correo_buap_mx/Documents/Catedras/PDA/Material%20y%20herramientas%20did%C3%A1cticas/Diapositivas%20Distribuida.pdf?CT=1714332478517&OR=ItemsView

Larios G., M. (2020). Programación Distribuida: un enfoque práctico (1st ed.). Benemerita Universidad Autónoma de Puebla-México.

Androutsellis-Theotokis, S., & Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. ACM Computing Surveys (CSUR), 36(4), 335–371. https://doi.org/10.1145/1041680.1041681

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf

Lavastida-López, Z., & Almeida-Cruz, Y. (2009). Propuesta de un modelo para el intercambio automático de información en redes P2P. https://www.researchgate.net/profile/Yudivian-Almeida-Cruz/publication/266203233_Propuesta_de_un_modelo_para_el_intercambio_automatico_de_informacion_en_redes_P2P/links/5429f3c00cf29bbc12676f7c/Propuesta-de-un-modelo-para-el-intercambio-automatico-de-infor

Glosario

- **Algoritmo:** Es un conjunto finito de instrucciones o reglas preestablecidas que nos proporcionan una base sobre la cual resolver un problema o realizar una tarea especifica dentro de un sistema computacional.
- **Blockchain:** Tecnología de registro distribuido que permite almacenar datos de forma segura, inmutable y descentralizada.
- **Consenso:** Mecanismo mediante el cual los nodos de una red distribuida acuerdan un estado para el sistema.
- **Descentralizado:** Principio según el cual no existe un nodo central de control, si no que se distribuyen entre múltiples nodos.
- **Enrutamiento:** Proceso mediante el cual se determina el camino óptimo que deben seguir los datos para llegar a su nodo de origen o nodo destino en una red.
- **Latencia:** Tiempo que tarda un paquete de datos en viajar desde su origen hasta su destino.
- **Metodología:** Es un conjunto de procedimientos, técnicas y herramientas sistemáticas utilizadas en las investigaciones científicas o en el desarrollo de tecnologías.
- **Multicast:** Técnica de transmisión de datos en redes que permite enviar información desde un solo emisor a múltiples receptores simultáneamente.
- **Nodo:** Unidad básica de una red P2P que puede actuar como cliente, servidor o ambos, participando en el intercambio de datos.
- **Paradigma:** Modelo o estructura conceptual que define como se estructura y organiza el conocimiento en un campo determinado.
- **Performance:** Medida del rendimiento de un sistema, que puede incluir métricas como latencia, velocidad de procesamiento, uso de recursos y estabilidad.
- **Protocolos:** Serie de reglas y convenciones que permiten la comunicación entre distintos dispositivos en una red, asegurando que los datos se trasfieran de manera correcta.

- **Refactorización:** Proceso en el cual se reestructura el código fuente de un sistema con el objetivo de mejorar su legibilidad, mantenibilidad y eficiencia.
- **Resiliencia:** Capacidad de un sistema distribuido para continuar funcionando correctamente ante fallos y ataques externos e internos.
- **Sockets:** Interfaces de programación que permiten la comunicación entre diferentes procesos mediante una red.
- **Topología:** estructura física o lógica de una red, que define como se conectaran sus distintos nodos y como fluirá la información a través de ella.
- **Trazabilidad:** Capacidad de rastrear el historial, la ubicación y el uso de un dato o recurso dentro de un sistema, útil para auditorios y control de versiones.