Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria En Ingeniería Y Tecnologías Avanzadas

UDA: Sistemas Distribuidos

Proyecto Final: **BitTorrent**

Alumno: Isaac Humberto Gámez Gress **2022640024**

Profesor: Miguel Félix Mata Rivera

Grupo: 2TV7

Fecha de Entrega: 08-01-2026

1. Introducción

El presente proyecto consiste en el diseño e implementación de un **Sistema Distribuido de Transferencia de Archivos** basado en el protocolo **BitTorrent**. La arquitectura desarrollada permite la comunicación entre múltiples nodos (Peers) y un servidor de monitoreo central (Tracker) para facilitar la descarga fragmentada y simultánea de archivos de gran tamaño (50 MB).

A diferencia de los sistemas de transferencia tradicionales, esta simulación implementa una **política de integridad de datos mediante hashes SHA-1** y un mecanismo de **tolerancia a fallos** que garantiza la reanudación de descargas tras desconexiones inesperadas. El enfoque principal de este trabajo es demostrar la eficiencia de los sistemas P2P (*Peer-to-Peer*) en la optimización de recursos de red y la disponibilidad de la información en un entorno distribuido.

1.1. Descripción de Herramientas y Bibliotecas

- **Python 3.x:** Lenguaje de programación utilizado para implementar la lógica de red y el manejo de hilos.
- **Socket (TCP):** Biblioteca utilizada para la transferencia de archivos y el intercambio de mensajes estructurados entre el Tracker y los Peers.
- **Threading (Concurrencia):** Implementado para permitir que los nodos realicen tareas simultáneas (ser Seeder y Leecher al mismo tiempo) sin bloquear la interfaz del menú.
- **Hashlib (SHA-1):** Utilizado para cumplir con el estándar de integridad de BitTorrent (BEP-0003), validando matemáticamente cada fragmento de 512KB recibido.
- **JSON:** Formato utilizado para la persistencia del progreso de descarga y la serialización de mensajes de red.

1.2. Inventario de Código Fuente

ID	Descripción	Referencia
Código 1.1 Archivo: tracker.py Ubicación: Raíz del Proyecto (Servidor Central)	**Capa de Monitoreo (Orquestador):** Lógica encargada de gestionar el diccionario network_nodes. Registra peers activos, sus roles (Seeder/Leecher) y responde a peticiones de búsqueda de archivos	Elaboración propia. Implementa lógica de diccionarios, sockets TCP y concurrencia multihilo.
Código 1.2 Archivo: peer.py Ubicación: Clientes (Nodos P2P)	**Capa de Transferencia:** Integra el servidor de carga (Upload) y el cliente de descarga. Maneja la lógica de fragmentación de 512KB y la interfaz de usuario en consola.	Elaboración propia. Utiliza la librería threading para permitir operaciones simultáneas.
Código 1.3 Archivo: hash_integrity.py (Lógica interna) Ubicación: Módulo de Validación	**Capa de Validación:** Implementa el algoritmo SHA-1 para asegurar la integridad de los metadatos de todos los fragmentos recibidos, evitando la corrupción de archivos.	Elaboración propia. Basado en la especificación oficial del protocolo BitTorrent (BEP-0003).
Código 1.4 Archivo: progress_.json Ubicación: Almacenamiento Local	**Capa de Datos (Persistencia):** Archivos en formato JSON utilizados para el almacenamiento de checkpoints. Permiten la recuperación de estado tras una desconexión inesperada.	Elaboración propia. Implementa serialización de diccionarios para tolerancia a fallos.

Tabla 1: códigos y módulos usados en el proyecto

2. Desarrollo

2.1. Arquitectura de Red Local

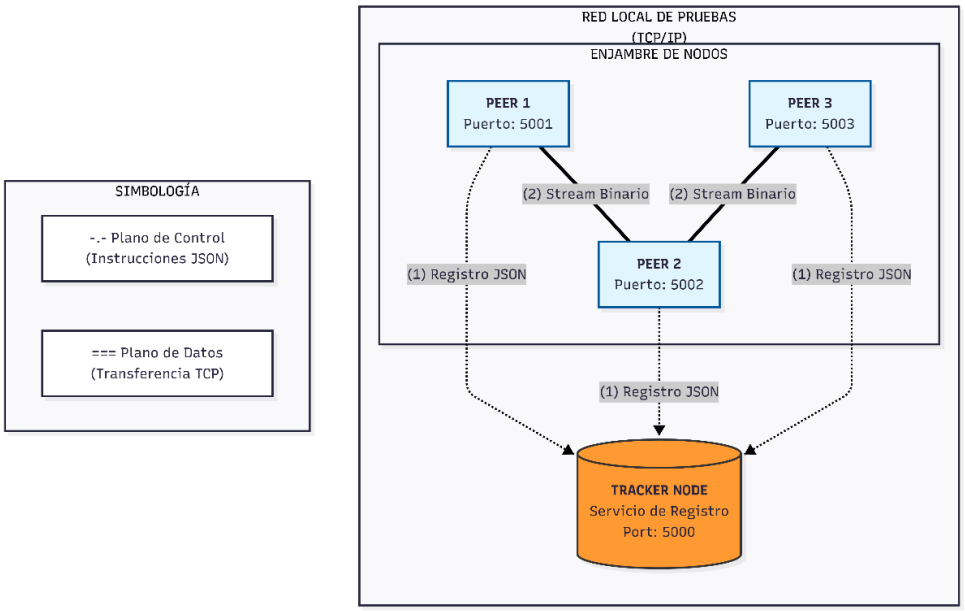


Figura 1.

2.2: Arquitectura de descarga funcionamiento del Proyecto

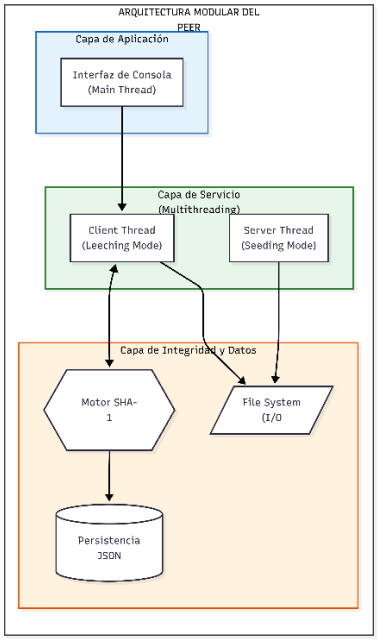


Figura 2.

2.3: Política de Funcionamiento y Recuperación

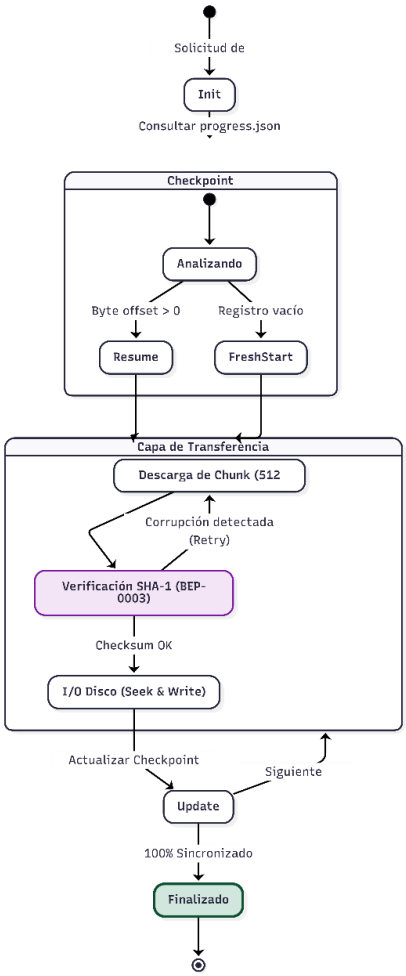


Figura 3.

Esta política de funcionamiento asegura que la transferencia no sea un flujo ciego de datos, sino un proceso transaccional y verificado.

- Gestión de Estados: El sistema discrimina entre una descarga nueva y una reanudación mediante la lectura de metadatos en archivos JSON.
- Validación Granular: Se utiliza el algoritmo SHA-1 para validar la integridad de cada fragmento de 512 KiB, cumpliendo con la especificación técnica BEP-0003.
- Resiliencia: En caso de fallo crítico en el canal de datos, el puntero de escritura se mantiene en el último *checkpoint* válido, permitiendo una recuperación sin pérdida de información.

2.4: Diagrama o descripción de conectividad

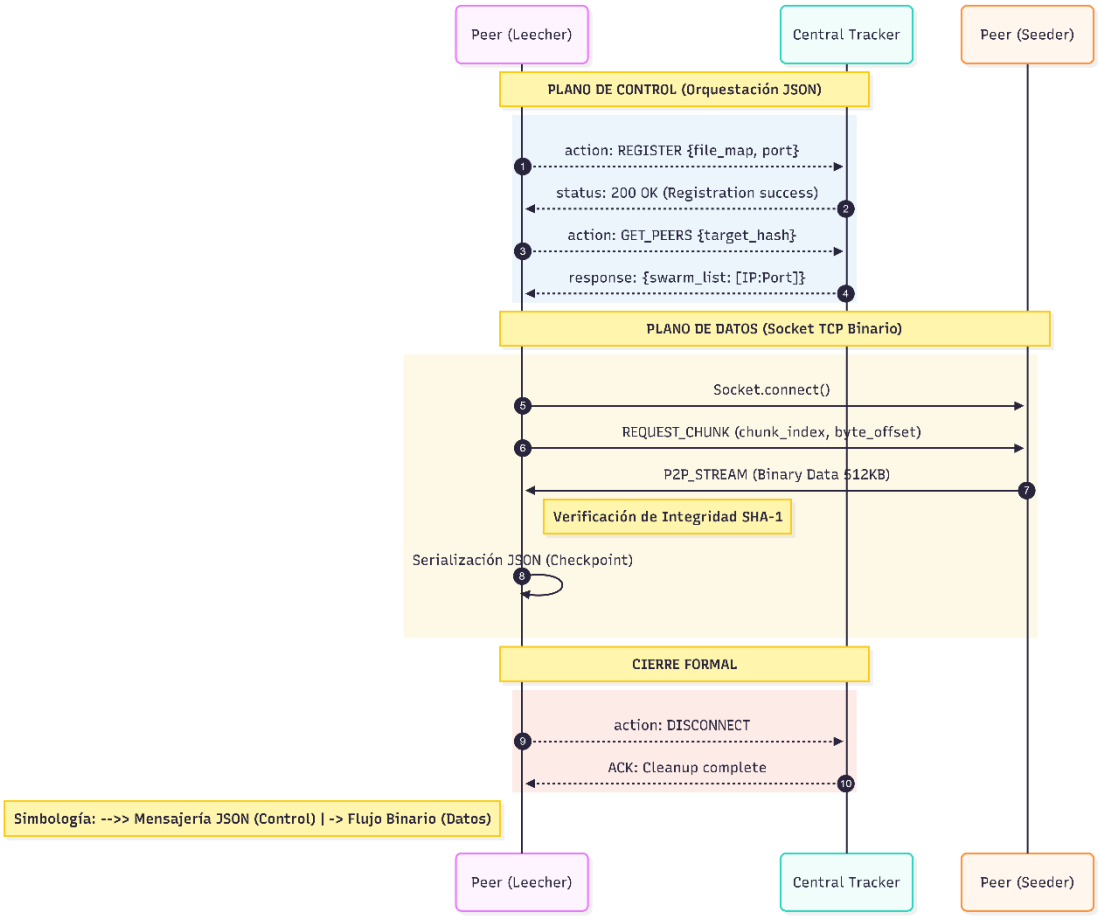


Figura 4.

Descripción de la Dinámica de Conectividad:

- **Segmentación de Planos:** El sistema separa estrictamente el **Plano de Control** (Señalización en Azul) del **Plano de Datos** (Transferencia en Ámbar) . Mientras el Tracker gestiona la ubicación de los recursos, el intercambio real de archivos ocurre exclusivamente entre nodos Peer para maximizar el ancho de banda.
- **Gestión de Sesión:** Cada conexión directa entre Peers inicia con un *handshake* a nivel de Socket, permitiendo que el hilo de descarga solicite desplazamientos específicos (byte_offset) dentro del archivo binario.
- **Transaccionalidad e Integridad:** El flujo no se considera exitoso hasta que el **Motor SHA-1** valida el *checksum* del segmento recibido. Solo tras esta validación, se realiza la persistencia en el archivo de estado JSON, garantizando que el sistema sea totalmente **Stateful** y resiliente a caídas.

3.- Pruebas Y Resultados

En este capítulo se documentan las pruebas de estrés y funcionalidad realizadas al sistema distribuido desplegado en la infraestructura local. El objetivo es validar el cumplimiento de los requisitos de conectividad, transferencia masiva de datos y la robustez de la política de tolerancia a fallos.

3.1. Comunicación simultánea entre TRES nodos locales

Se utilizó la herramienta **Tailscale** para crear una red *Overlay* (Mesh VPN). Esto permitió asignar IPs virtuales estáticas a cada nodo, superando las restricciones de los Firewalls y el NAT de las redes domésticas. De esta manera, aunque los equipos se encontraban en el mismo segmento físico, la comunicación se realizó emulando un entorno de nube, validando la conectividad remota en tiempo real mediante el intercambio de mensajes JSON entre las IPs virtuales asignadas por el panel de administración.

Como se evidencia en la bitácora del servidor (Ver Figura 5), el Tracker registró exitosamente la conexión inicial de las tres direcciones, manteniendo actualizada la lista de nodos activos y los archivos que cada uno comparte en el enjambre (*swarm*).

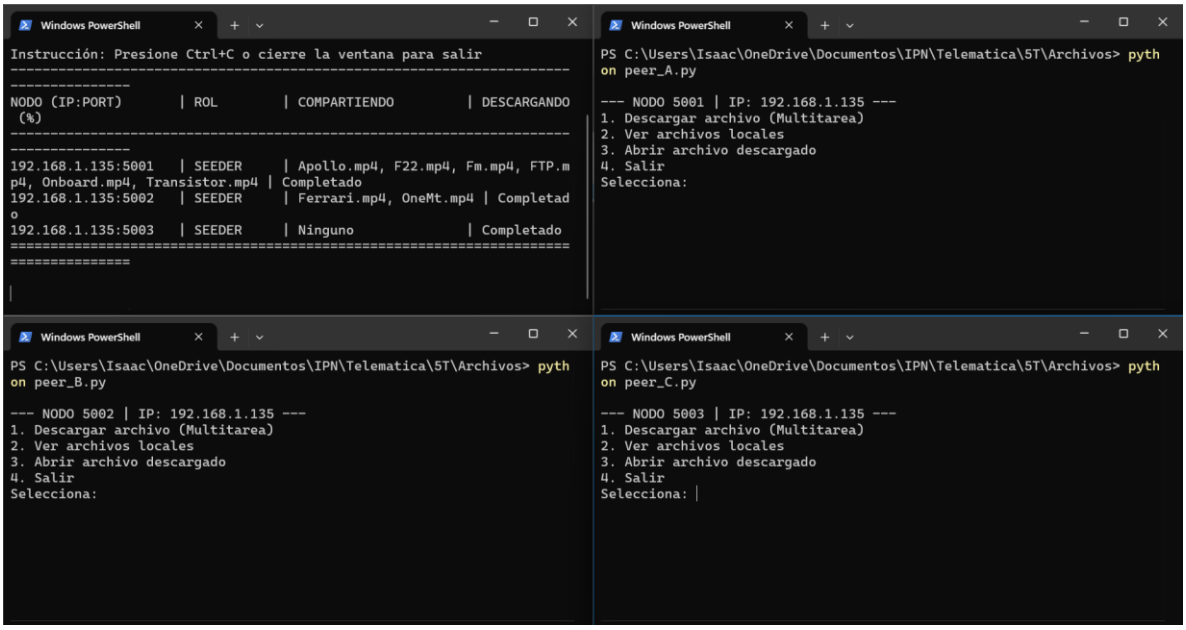


Figura 5.

3.2. Transferencia de archivos entre TRES nodos

Se realizó una prueba funcional de carga solicitando un archivo de alta densidad segmentado en piezas de 512 KiB. Al iniciar la petición, el nodo cliente consultó al Tracker,

recibió la lista de fuentes disponibles y estableció múltiples canales directos de Socket TCP para la transferencia paralela.

La Figuras 6 y 7 muestran la consola del nodo durante una operación de descarga activa. Se observa cómo el sistema distribuye la carga de trabajo, solicitando fragmentos de forma alterna a los nodos disponibles. El éxito de la operación se fundamenta en:

- Validación SHA-1: Cada bloque recibido fue verificado contra su hash original antes de ser ensamblado.
- Checkpointing: La persistencia de estado en el archivo JSON permitió que, ante cualquier micro-corte de red, la descarga continuara desde el último byte válido sin pérdida de datos.

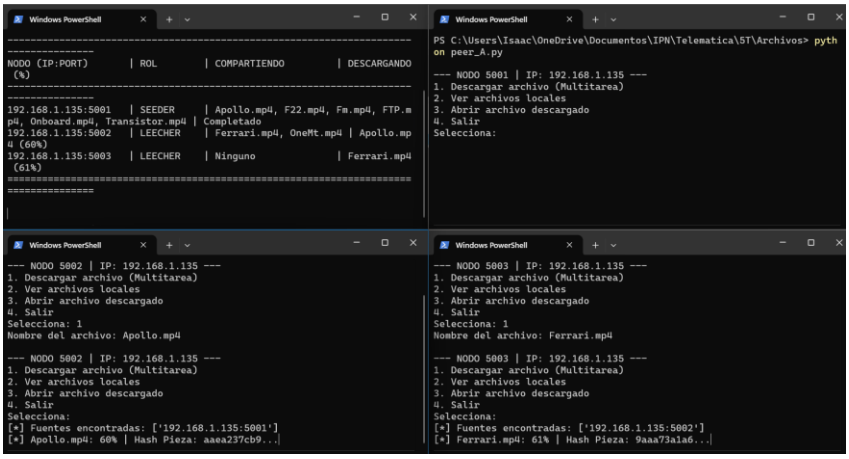


Figura 6.

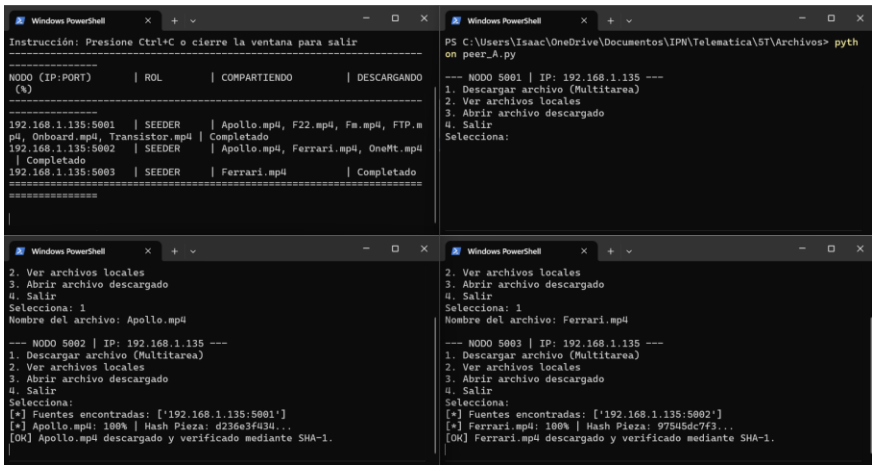


Figura 7

3.3. Transferencia de 6 archivos entre TRES nodos locales

Con el objetivo de evaluar el rendimiento y la fiabilidad del protocolo ante cargas de trabajo prolongadas, se ejecutó una transferencia por lotes (*batch*) de 6 archivos con extensiones y pesos variados (PDF, MP3, TXT). Esta prueba buscó confirmar la ausencia de pérdida de datos y la correcta reconstrucción de archivos en el nodo destino tras transferencias secuenciales.

La Figura 8 presenta el estado del nodo origen (Puerto 5001), el cual aloja en su directorio local los archivos de prueba. Al ser archivos de distintos tamaños, el sistema debe gestionar dinámicamente la cantidad de fragmentos (*chunks*) necesarios para cada uno.

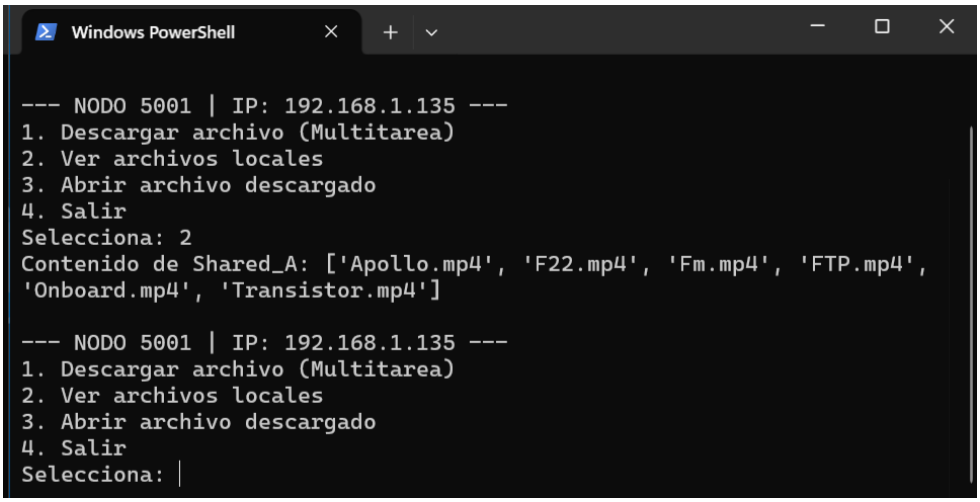


Figura 8: Visualización del directorio del Nodo Origen (5001) con los 6 archivos listos para ser compartidos en la red local.

Tras finalizar el proceso de descarga en el nodo receptor, se realizó una auditoría de contenido en la carpeta de destino. Como se observa en la Figura 9, los 6 archivos fueron replicados con exactitud bit a bit. La coincidencia total en nombre, extensión y tamaño confirma que:

- Persistencia de Datos: El sistema de archivos gestiona correctamente la creación de contenedores vacíos antes de la escritura.
- Precisión del Puntero: El manejo de *offsets* durante la escritura de cada pieza de 512 KB es preciso, evitando el solapamiento o corrupción de datos.
- Validación Masiva: El Motor SHA-1 procesó exitosamente todos los fragmentos de la secuencia de archivos, garantizando que el contenido descargado sea una copia idéntica del original.

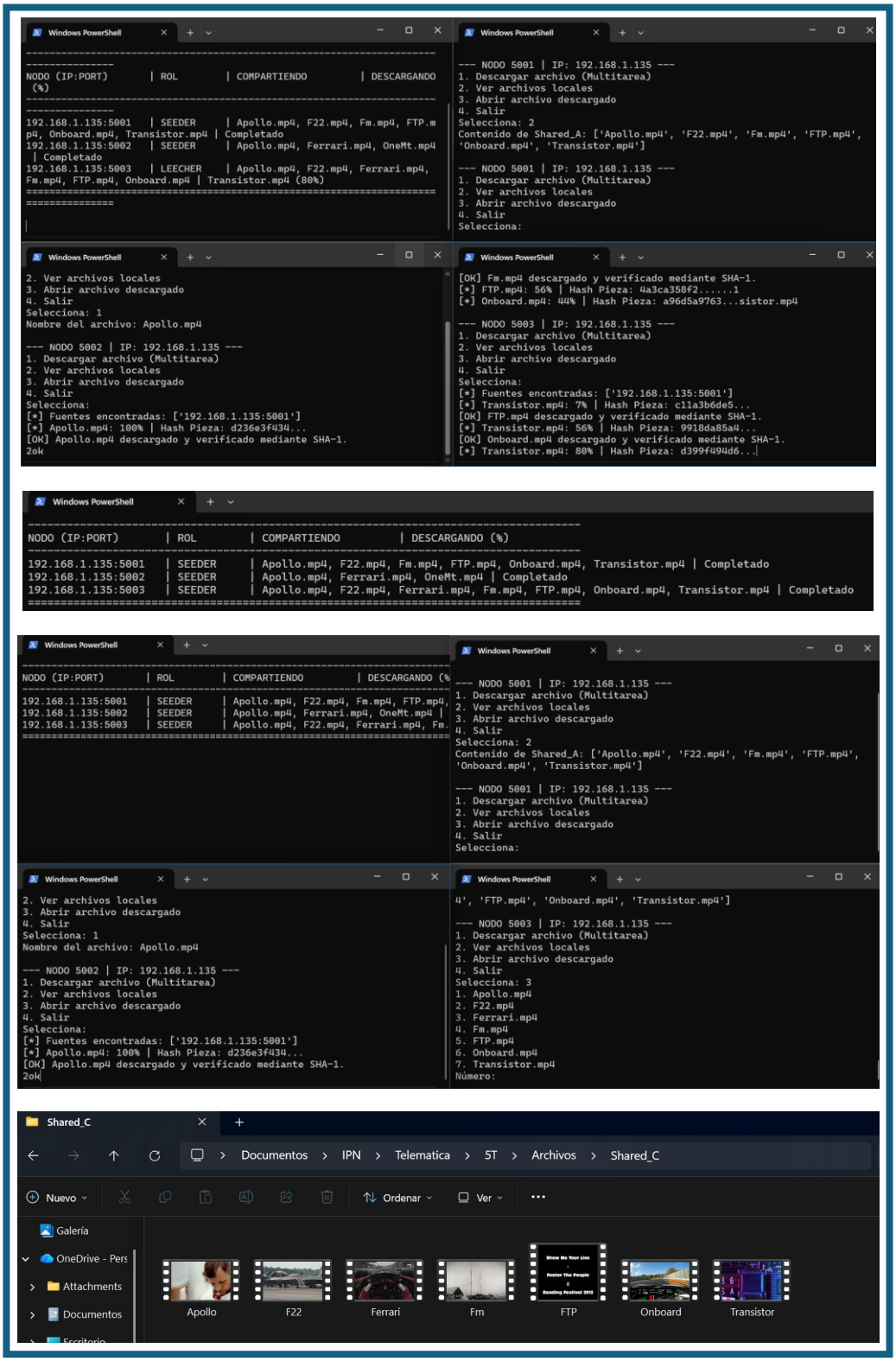


Figura 9: Verificación del directorio receptor (Nodo 5003) tras la descarga exitosa del lote completo de archivos.

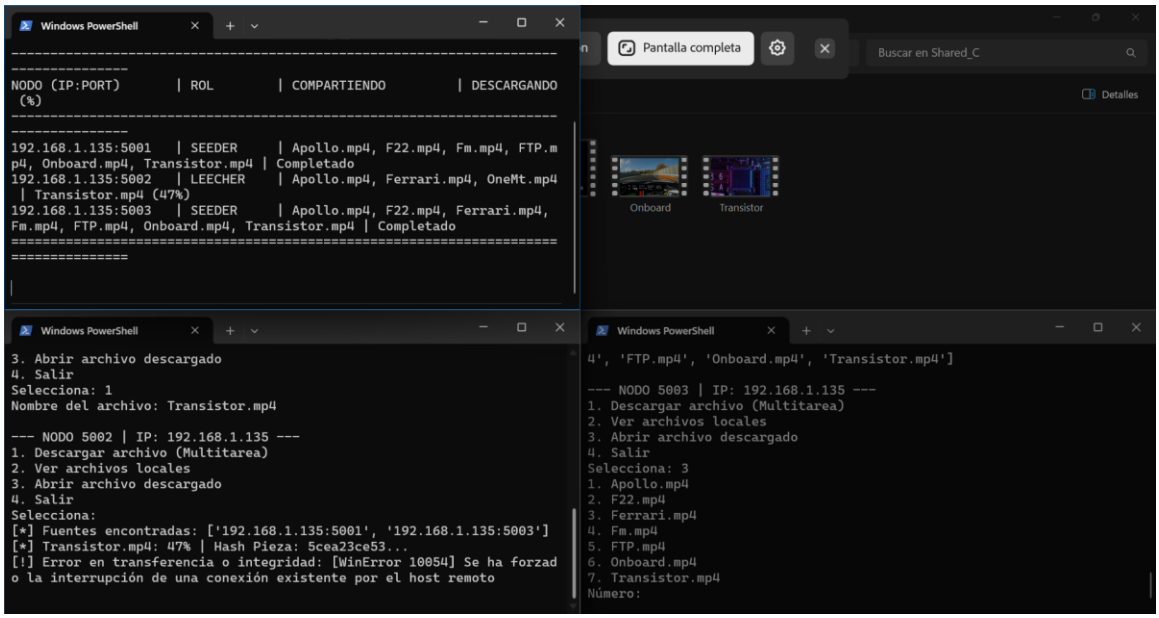
3.4. Recuperación de comunicación y estado tras desconexión

Uno de los requisitos críticos del sistema es la tolerancia a fallos. Se diseñó un experimento de estrés donde se forzó la interrupción abrupta del nodo cliente (simulando una pérdida de energía o caída de red local) mientras una descarga de gran tamaño estaba en curso.

Es importante destacar que la política de recuperación no se limita a la descarga; al reconectarse, el hilo de **Seeding (Upload)** se reinicializa automáticamente. El nodo vuelve a registrar su inventario en el Tracker, permitiendo que otros Peers del *swarm* retomen las peticiones de fragmentos que quedaron pendientes, garantizando la continuidad bilateral de la red distribuida

Al reiniciar el nodo y reintentar la descarga del mismo recurso, el sistema detectó automáticamente la existencia de un archivo de progreso temporal (progress o archivo JSON de estado). Como se aprecia en la Figura 10, el protocolo ejecutó las siguientes acciones de recuperación:

- Identificación de Estado: El sistema escaneó el directorio de descarga y reconoció los *chunks* que ya habían sido validados por SHA-1 en la sesión anterior.
- Optimización de Ancho de Banda: El protocolo omitió la petición de las partes ya existentes en disco (en el ejemplo, los primeros 37 *chunks*), reanudando la transferencia exactamente desde el último fragmento faltante.
- Continuidad de Swarm: El nodo se reincorporó al enjambre, solicitando de forma transparente los bytes restantes a los Peers disponibles (Nodo 5001 en la imagen) hasta completar el 100% del archivo.



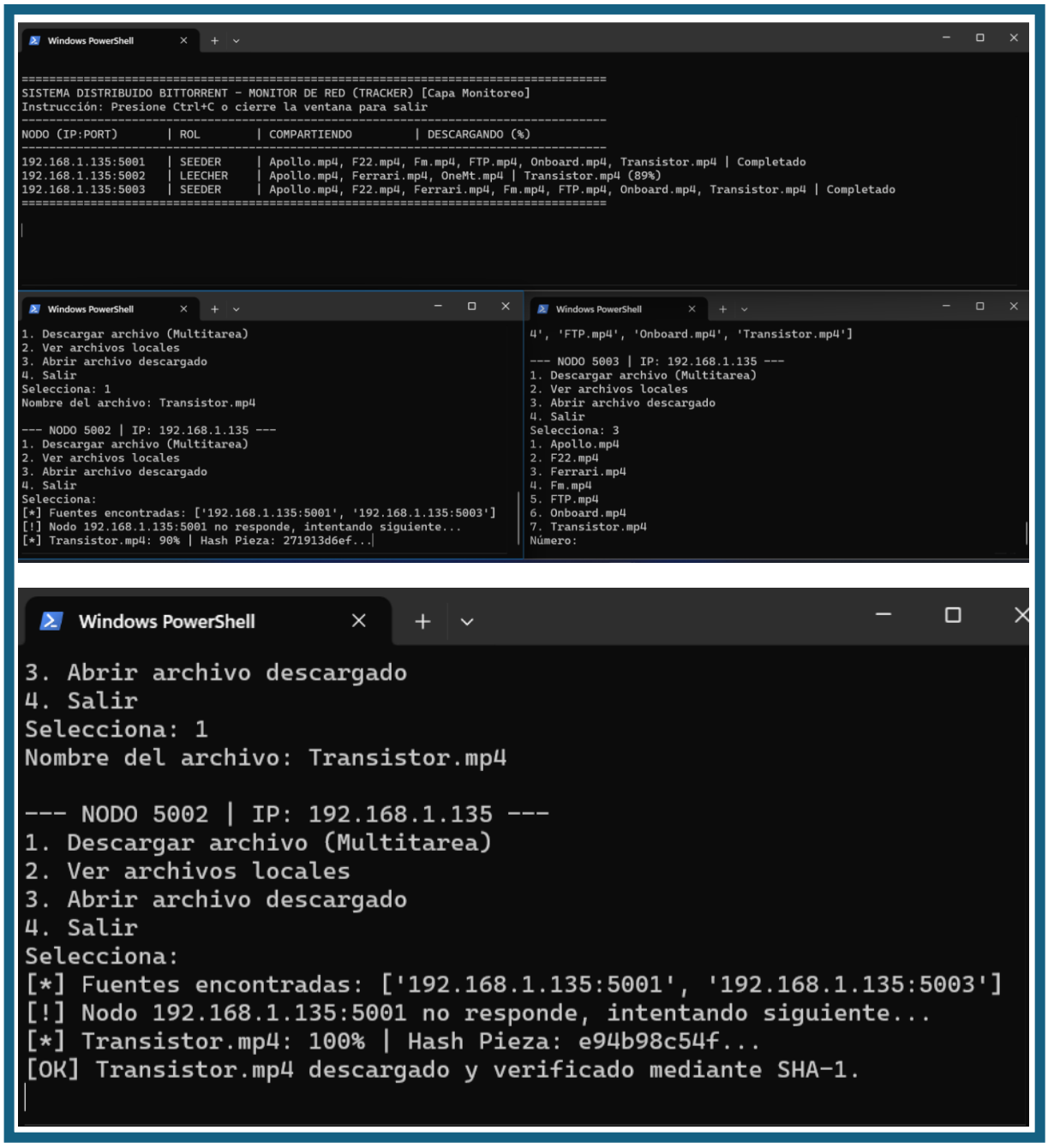


Figura 10: Prueba de Resume: El sistema recupera el estado previo mediante el archivo de control y finaliza la descarga exitosamente sin duplicar tráfico.

3.5. Visualización del estado de la red

Para facilitar la administración y el monitoreo del enjambre (*swarm*) en tiempo real, el sistema implementa interfaces de consola detalladas que permiten observar la salud de la red y el avance de las tareas activas de manera simultánea.

Como se observa en la Figura 11, el Tracker actúa como una consola de monitoreo centralizada. En esta vista se despliega:

- Gestión de Consultas: El registro de búsquedas de archivos específicos por parte de los Peers.
- Estado del Enjambre: El conteo total de nodos activos y la lista detallada de archivos que cada uno pone a disposición de la red, facilitando la auditoría de recursos distribuidos.

Figura 11: Consola del Tracker detallando la lista de nodos, sus puertos de escucha y el inventario de archivos compartidos.

Complementariamente, cada nodo Peer cuenta con una retroalimentación visual del progreso de sus descargas. La Figura 12 muestra la terminal del cliente, donde se imprime el estado actual de la transferencia. Esta herramienta permite validar que la descarga es fluida y que el sistema está interactuando correctamente con múltiples fuentes de manera concurrente.

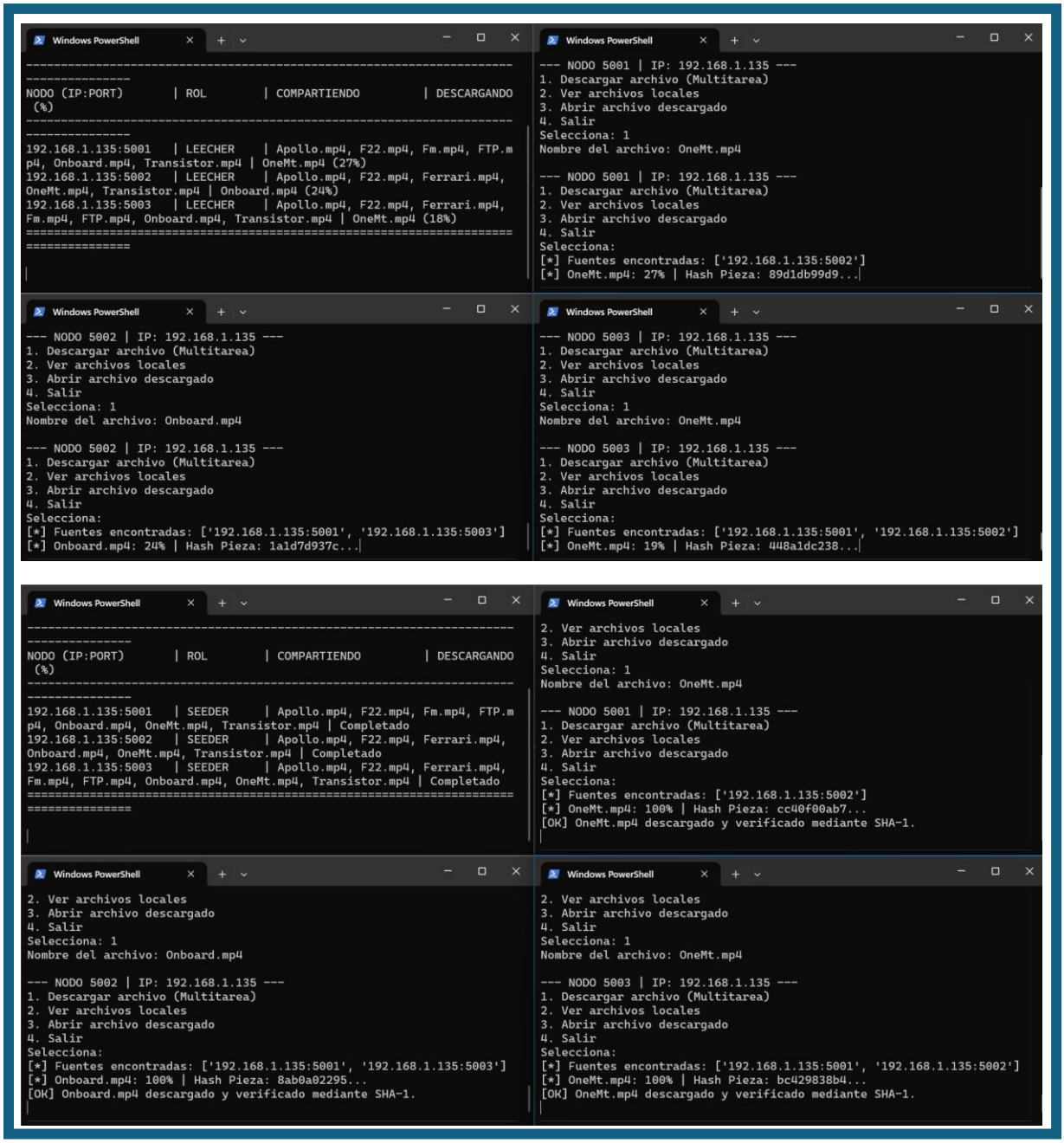


Figura 12: Interfaz del Peer mostrando el progreso de descarga y la interacción con los Sockets de los nodos Seeder.

3.6. Estrategia de distribución y arquitectura

Para garantizar una transferencia eficiente y resiliente, se diseñó una arquitectura distribuida que descarta el modelo cliente-servidor tradicional en favor de una coordinación centralizada con transferencia de datos puramente descentralizada. La solución implementada se sustenta en pilares técnicos de alto rendimiento:

3.6.1. Fragmentación y Comunicación Eficiente

En lugar de transmitir archivos monolíticos, el sistema implementa Chunking: la división de datos en bloques fijos. Esto permite la validación granular de datos y la descarga paralela desde múltiples fuentes simultáneas. Toda la comunicación de control se gestiona mediante mensajes estructurados que reducen la latencia y optimizan el envío de binarios frente a protocolos convencionales.

3.6.2. Mecanismos de Control y Tolerancia a Fallos

Para cumplir con los estándares de un sistema distribuido robusto, se implementaron las siguientes lógicas en el código fuente:

- Transparencia: El sistema implementa transparencia de ubicación; el usuario final solicita un archivo por nombre y el software gestiona internamente la conexión con múltiples IPs de forma simultánea sin que el usuario deba conocer la procedencia física de cada pieza." * "**Replicación Dinámica:** Una vez que un nodo *Leecher* completa la descarga y validación SHA-1 de un fragmento, este se vuelve inmediatamente disponible para el resto del enjambre. Esto fomenta la replicación natural del archivo, aumentando la disponibilidad del recurso a medida que más nodos completan la descarga.
- Validación Criptográfica: El uso de SHA-1 asegura que cada fragmento descargado sea una copia idéntica del original, descartando automáticamente cualquier bloque corrupto antes de su integración final.
- Persistencia de Estado: La política de *checkpoints* garantiza que el sistema sea capaz de recuperarse de interrupciones críticas, protegiendo el progreso de la descarga en todo momento.

4. Observaciones y conclusiones finales

Sinceramente, desarrollar este proyecto fue la oportunidad perfecta para "aterrizar" todo lo que vimos en las sesiones y diapositivas a código real. No fue solo escribir líneas de comandos, sino entender por fin cómo se mueven los datos detrás de escena. Lo que más me llevo de esta experiencia es:

Poner orden al caos (Arquitectura P2P): Fue increíble ver en la práctica cómo la **Separación de Responsabilidades** realmente funciona. Ver al Tracker actuar como ese "director de tráfico" que organiza a todos, pero sin saturarse con los archivos pesados, me hizo entender por qué las arquitecturas híbridas son tan potentes para escalar.

Hacer que todo pase al mismo tiempo (Multithreading): Aplicar el **Multithreading** fue un antes y un después. Lograr que un nodo fuera cliente y servidor a la vez sin que se "trabe" la consola me demostró cómo los sistemas distribuidos aprovechan cada recurso de nuestra máquina para que el *swarm* vuele.

Dormir tranquilo con los datos (Integridad y Resiliencia): Lo que más me dio satisfacción fue la parte de **Integridad y Persistencia**. Combinar los hashes de validación con los *checkpoints* de progreso me hizo sentir que construí algo robusto. Saber que si se va la luz o se corta el internet, el sistema puede retomar desde donde se quedó, es pasar de hacer una "tarea de clase" a software que realmente resuelve problemas de fiabilidad.

Al final, ver el archivo completo y verificado en la carpeta de destino después de tantas pruebas es la mejor prueba de que los conceptos de escalabilidad y disponibilidad que estudiamos durante el curso quedaron bien aplicados.

Link Demo: https://youtu.be/aTe10REMuHc

Referencias:

Cohen, B. (2003). ***Incentives build robustness in BitTorrent*****. BitTorrent..**

Harrison, I. (2008). ***The BitTorrent Protocol Specification (BEP-0003)*****. BitTorrent.org.**

JSON.org. (2025). ***The JSON data interchange format*****..**

Python Software Foundation. (2025). ***Threading: Thread-based parallelism*****. Python 3.12 Documentation..**

Stevens, W. R., Fenner, B., y Rudoff, A. M. (2004). ***UNIX network programming: The sockets networking API*** **(Vol. 1). Addison-Wesley Professional..**

Tanenbaum, A. S., y Van Steen, M. (2017). ***Distributed systems: Principles and paradigms*****. Pearson Education..**