



INSTITUTO POLITÉCTICO NACIONAL

---

---

UPIITA - Unidad Profesional  
Interdisciplinaria en Ingeniería y  
Tecnologías Avanzadas IPN

Sistemas Distribuidos

*PROYECTO FINAL EMULACIÓN DE UNA  
RED BITTORRENT LIMITADA*

Marco González Luna 2024640044

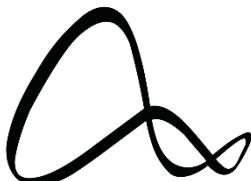
Olvera Valdivia Cristobal 2024640014

Grupo

2TV7

Profesor

Mata Rivera Miguel Felix



upiita-ipn

# **1. INDICE**

<b>1.</b>	<b>INDICE .....</b>	<b>2</b>
<b>2.</b>	<b>INTRODUCCIÓN .....</b>	<b>3</b>
A)	DESCRIPCIÓN DE HERRAMIENTAS, BIBLIOTECAS, CÓDIGOS INTEGRADOS PARA EL PROYECTO .....	3
B)	TABLA DE CÓDIGOS UTILIZADOS EN EL PROYECTO .....	4
<b>3.</b>	<b>DESARROLLO .....</b>	<b>6</b>
1.	DIAGRAMA O DESCRIPCIÓN O FIGURAS DE LA CONFIGURACIÓN DE RED O DE LA NUBE .....	6
2.	DIAGRAMA DE ARQUITECTURA DE FUNCIONAMIENTO DEL PROYECTO, ENFATIZANDO LOS ELEMENTOS DISTRIBUIDOS .....	11
3.	DISEÑO DE POLÍTICA DE FUNCIONAMIENTO DE DESCARGAS, TRANSFERENCIAS Y RECUPERACIÓN DE ERRORES	13
4.	DIAGRAMA O DESCRIPCIÓN DE CONECTIVIDAD .....	14
<b>4.</b>	<b>PRUEBAS Y RESULTADOS .....</b>	<b>16</b>
A)	LISTA O TABLA CON EXPLICACIÓN DEL PRODUCTO O FUNCIONALIDAD NO LOGRADA.....	16
B)	IMÁGENES DE LA EJECUCIÓN DEL PROYECTO CON LOS PASOS MÁS RELEVANTES(SCREENDSHOTS) Y DE LOS EXPERIMENTOS Y RESULTADOS OBTENIDOS , EXPLICANDO EN TEXTO LOS DETALLES.....	17
<b>5.</b>	<b>OBSERVACIONES Y CONCLUSIONES FINALES.....</b>	<b>25</b>
<b>6.</b>	<b>REFERENCIAS.....</b>	<b>27</b>

## 2. INTRODUCCIÓN

### a) Descripción de herramientas, bibliotecas, códigos integrados para el proyecto

Para el desarrollo de este sistema de intercambio de archivos P2P (Peer-to-Peer) emulando el protocolo BitTorrent, se utilizó el lenguaje de programación Python 3, debido a su versatilidad para el manejo de Sockets y manipulación de archivos binarios.

Las principales bibliotecas y herramientas integradas fueron:

- **Socket (Python Standard Library)**: Utilizada para establecer la comunicación TCP/IP tanto entre los Nodos (Peers) como con el servidor de rastreo (Tracker). Se implementaron sockets no bloqueantes y con timeout para gestionar desconexiones.
- **Threading**: Esencial para la concurrencia. Permite que cada nodo funcione simultáneamente como cliente (descargando) y servidor (subiendo), además de mantener hilos secundarios para heartbeats (latidos al tracker) y escucha de peticiones entrantes.
- **Hashlib (SHA-256)**: Integrado para la seguridad e integridad de los datos. Se utiliza para generar la firma digital única del archivo (filehash) y, crucialmente, para validar cada pieza (chunk) descargada individualmente, asegurando que no haya corrupción de datos.
- **JSON**: Empleado como el protocolo de serialización para el intercambio de mensajes (metadatos, listas de peers, comandos) y para el almacenamiento de archivos de progreso (.progress) y descriptores de torrent (.json).
- **AWS EC2 (Amazon Web Services)**: Se utilizó una instancia en la nube para alojar el **Tracker** principal y un **Seed** inicial, garantizando alta disponibilidad y una IP pública fija para la orquestación de la red.
- **Tailscale / VPN**: Herramienta de red implementada para superar las restricciones de NAT (CGNAT) y Firewalls locales, permitiendo la conexión directa entre pares en diferentes redes domésticas sin necesidad de configurar *Port Forwarding* manual en los routers.

## b) Tabla de códigos utilizados en el proyecto.

ID	Descripción	Referencia
Codigo 1.0  Archivo peer_node.py  Ubicado en la carpeta del Proyecto BitTorrentSD: <a href="https://github.com/MarkKriegsbergerit/BitTorrentSD">https://github.com/MarkKriegsbergerit/BitTorrentSD</a>	Es el núcleo del cliente P2P. Gestiona la lógica de descarga fragmentada, la selección de peers basada en latencia/velocidad, la verificación de integridad SHA-256 y actúa como servidor para compartir piezas con otros nodos. Incluye el "Modo Sabotaje" para pruebas de fallos.	Código de elaboración propia.  Ubicado en la carpeta raíz.
Codigo 1.1  Archivo tracker.py  Ubicado en la carpeta del Proyecto BitTorrentSD: <a href="https://github.com/MarkKriegsbergerit/BitTorrentSD">https://github.com/MarkKriegsbergerit/BitTorrentSD</a>	Servidor centralizado que mantiene el registro de qué nodos poseen qué archivos (swarm). Gestiona comandos como ANNOUNCE, LIST_FILES y limpia peers inactivos (pruning).	Código de elaboración propia.  Ubicado en la instancia AWS / Carpeta BitTorrentSD.
Codigo 1.2  Archivo file_manager.py  Ubicado en la carpeta del Proyecto BitTorrentSD: <a href="https://github.com/MarkKriegsbergerit/BitTorrentSD">https://github.com/MarkKriegsbergerit/BitTorrentSD</a>	Módulo encargado de la persistencia en disco. Maneja la lectura y escritura aleatoria de piezas (chunks) en el archivo físico, así como el guardado y recuperación del estado de descarga (resume capability).	Código de elaboración propia.  Ubicado en la carpeta raíz.
Codigo 1.3  Archivo crear_torrent.py  Ubicado en la carpeta del Proyecto BitTorrentSD: <a href="https://github.com/MarkKriegsbergerit/BitTorrentSD">https://github.com/MarkKriegsbergerit/BitTorrentSD</a>	Script de utilidad para generar los metadatos .json. Calcula el hash global del archivo y, más importante, segmenta el archivo para calcular el hash SHA-256 de cada pieza individual para su posterior verificación.	Código de elaboración propia.  Ubicado en la carpeta raíz.

ID	Descripción	Referencia
Codigo 1.4  Archivo common.py  Ubicado en la carpeta del Proyecto BitTorrentSD: <a href="https://github.com/MarkKriegsbergerit/BitTorrentSD">https://github.com/MarkKriegsbergerit/BitTorrentSD</a>	Archivo de configuración global. Define constantes compartidas por todos los nodos, como BLOCK_SIZE (Tamaño de pieza), puertos por defecto y códigos de operación del protocolo.	Código de elaboración propia.  Ubicado en la carpeta raíz.

## 3. DESARROLLO

### 1. Diagrama o descripción o figuras de la configuración de Red o de la NUBE

Para la infraestructura de red del proyecto, se implementó una arquitectura híbrida que combina computación en la nube (Cloud Computing) con redes privadas virtuales (VPN) para superar las limitaciones de conectividad NAT domésticas.

La configuración se divide en dos segmentos:

#### 1. Nube Pública (AWS EC2):

- Se desplegaron dos instancias virtual **EC2** con sistema operativo Linux (Ubuntu) en Amazon Web Services.
- **IP Elástica:** Se asignó la dirección IP pública estática 3.151.6.85 a la primera instancia y a la segunda se le asignó la IP pública estática 3.137.135.235 para garantizar que el servidor de los Tracker's sea localizable permanentemente por cualquier nodo.
- **Security Groups (Firewall):** Se configuraron reglas de entrada (*Inbound Rules*) para permitir tráfico TCP en el puerto 5000 y 5001 (Tracker) y puertos de rango 9000 (Seeders en la nube), además de habilitar el protocolo ICMP para diagnósticos de red (Ping).

#### 2. Red Superpuesta (Overlay Network con Tailscale):

- Debido a que los nodos clientes se encuentran en redes residenciales detrás de CGNAT, se implementó **Tailscale** (basado en WireGuard).
- Esto crea una red malla (*mesh*) donde cada dispositivo obtiene una IP privada única (rango 100.x.y.z), permitiendo que los peers se conecten directamente entre sí (Peer-to-Peer) como si estuvieran en una red local extendida, sin necesidad de configuración manual de routers.

## CAPTURA DE LA CONSOLA DE AWS MOSTRANDO LA IP ELÁSTICA Y EL GRUPO DE SEGURIDAD

The figure consists of three vertically stacked screenshots of the AWS EC2 Instances console.

**Screenshot 1: Instances (1/3) - BitTorrent-Ser...**

Name	ID de la instancia	Estado de la i...	Tipo de inst...	Comprobación de	Estado de la al:	Zona de dispon...	DNS de IPv4
BitTorrent-Ser...	i-09f8b8d085387b2f8	Detenida	t3.micro	-	Ver alarmas +	us-east-2c	ec2-3-151-6
BitTorrentSD2	i-08107e6a71eab91a5	Detenida	t3.micro	-	Ver alarmas +	us-east-2c	ec2-3-137-1

**Screenshot 2: i-09f8b8d085387b2f8 (BitTorrent-Server)**

**Resumen de instancia**

- ID de la instancia**: i-09f8b8d085387b2f8
- Dirección IPv4 pública**: 3.151.6.85 | dirección abierta ↗
- Direcciones IPv4 privadas**: 172.31.39.215
- Dirección IPv6**: -
- Estado de la instancia**: Detenida
- DNS público**: ec2-3-151-6-85.us-east-2.compute.amazonaws.com | dirección abierta ↗

**Screenshot 3: Instances (1/3) - BitTorrentSD2**

Name	ID de la instancia	Estado de la i...	Tipo de inst...	Comprobación de	Estado de la al:	Zona de dispon...	DNS de IPv4
Playful Paws T...	i-0d4130cd6d9b012af	Detenida	t3.micro	-	Ver alarmas +	us-east-2c	ec2-3-17-9
BitTorrent-Ser...	i-09f8b8d085387b2f8	Detenida	t3.micro	-	Ver alarmas +	us-east-2c	ec2-3-151-6
BitTorrentSD2	i-08107e6a71eab91a5	Detenida	t3.micro	-	Ver alarmas +	us-east-2c	ec2-3-137-1

**i-08107e6a71eab91a5 (BitTorrentSD2)**

**Resumen de instancia**

- ID de la instancia**: i-08107e6a71eab91a5
- Dirección IPv4 pública**: 3.137.135.235 | dirección abierta ↗
- Direcciones IPv4 privadas**: 172.31.41.204
- Dirección IPv6**: -
- Estado de la instancia**: Detenida
- DNS público**: ec2-3-137-135-235.us-east-2.compute.amazonaws.com | dirección abierta ↗
- Tipo de nombre de anfitrión**: Nombre de IP: ip-172-31-41-204.us-east-2.compute.internal
- Nombre DNS de IP privada (solo IPv4)**: ip-172-31-41-204.us-east-2.compute.internal

Figura 1. Panel de instancias EC2 mostrando el despliegue de servidores redundantes para tolerancia a fallos.

**Direcciones IP elásticas (1/3) Información**

Name	Dirección IPv4 asignada	Tipo	ID de asignación	Registro DNS inverso	ID de puerta de enlace de NAT
-	3.137.135.235	IP pública	eipalloc-0ad5afa6568ca4e86	-	i-08
<input checked="" type="checkbox"/>	3.151.6.85	IP pública	eipalloc-0c50c59c636029fd3	-	i-09
-	3.17.97.125	IP pública	eipalloc-06d2ce2090e6d62d3	-	i-0d

**3.151.6.85**

**Resumen**

Dirección IPv4 asignada <a href="#">3.151.6.85</a>	Tipo <a href="#">IP pública</a>	ID de asignación <a href="#">eipalloc-0c50c59c636029fd3</a>	Registro DNS inverso -
ID de asociación <a href="#">eipassoc-046d0a9cd230ae7b</a>	Ámbito <a href="#">VPC</a>	ID de instancia asociada <a href="#">i-09f8b8d085387b2f8</a>	Dirección IP privada <a href="#">172.31.39.215</a>
ID de interfaz de red <a href="#">eni-019e6c0197d9fc30b</a>	ID de la cuenta del propietario de la interfaz de red <a href="#">289591070618</a>	DNS público <a href="#">ec2-3-151-6-85.us-east-2.compute.amazonaws.com</a>	ID de puerta de enlace de NAT -
Grupo de direcciones <a href="#">Amazon</a>	Service managed		

**Direcciones IP elásticas (1/3) Información**

Name	Dirección IPv4 asignada	Tipo	ID de asignación	Registro DNS inverso	ID de puerta de enlace de NAT
<input checked="" type="checkbox"/>	3.137.135.235	IP pública	eipalloc-0ad5afa6568ca4e86	-	i-08
-	3.151.6.85	IP pública	eipalloc-0c50c59c636029fd3	-	i-09
-	3.17.97.125	IP pública	eipalloc-06d2ce2090e6d62d3	-	i-0d

**3.137.135.235**

**Resumen**

Dirección IPv4 asignada <a href="#">3.137.135.235</a>	Tipo <a href="#">IP pública</a>	ID de asignación <a href="#">eipalloc-0ad5afa6568ca4e86</a>	Registro DNS inverso -
ID de asociación <a href="#">eipassoc-0c74e521c06742e98</a>	Ámbito <a href="#">VPC</a>	ID de instancia asociada <a href="#">i-08107e6a71eab91a5</a>	Dirección IP privada <a href="#">172.31.41.204</a>
ID de interfaz de red <a href="#">eni-06950e596a4708b50</a>	ID de la cuenta del propietario de la interfaz de red <a href="#">289591070618</a>	DNS público <a href="#">ec2-3-137-135-235.us-east-2.compute.amazonaws.com</a>	ID de puerta de enlace de NAT -
Grupo de direcciones <a href="#">Amazon</a>	Service managed		

Figura 2. Asignación de IPs estáticas para asegurar la localización permanente de ambos Trackers.

AWS | Buscar [Alt+S] Estados Unidos (Ohio) ID de cuenta: 2895-9107-0618 Cristobal11082005

EC2 > Grupos de seguridad > sg-06be4b8124808cd3 - launch-wizard-4 > Editar reglas de entrada

Las reglas de entrada controlan el tráfico entrante que puede llegar a la instancia.

ID de la regla del grupo de seguridad	Tipo	Protocolo	Intervalo de puertos	Origen	Descripción: opcional
sgr-0070af9dba23995ad	SSH	TCP	22	Person... ▾	0.0.0.0/0 X
sgr-04903e790693c2df5	TCP personalizado	TCP	5000	Person... ▾	0.0.0.0/0 X
sgr-0fc56c6fb340c4154	Todos los ICMP IPv4	ICMP	Todo	Person... ▾	0.0.0.0/0 X
sgr-05d26f26d83fa96ab	TCP personalizado	TCP	9000	Person... ▾	0.0.0.0/0 X
sgr-026ac54ce36fa6d8a	SMTP	TCP	25	Person... ▾	0.0.0.0/0 X
sgr-0144ea3414e3f5d61	HTTPS	TCP	443	Person... ▾	0.0.0.0/0 X
sgr-0e03f856b2f737631	HTTP	TCP	80	Person... ▾	0.0.0.0/0 X

[Agregar regla](#)

AWS | Buscar [Alt+S] Estados Unidos (Ohio) ID de cuenta: 2895-9107-0618 Cristobal11082005

EC2 > Grupos de seguridad > sg-0f02389f541df33d0 - launch-wizard-5 > Editar reglas de entrada

ID de la regla del grupo de seguridad	Tipo	Protocolo	Intervalo de puertos	Origen	Descripción: opcional
sgr-0962c91ce26f70c8c	SSH	TCP	22	Person... ▾	0.0.0.0/0 X
sgr-09c7a3fc9673843c7	HTTPS	TCP	443	Person... ▾	0.0.0.0/0 X
sgr-0135685865f6b335f	TCP personalizado	TCP	5001	Person... ▾	tracker 2
sgr-0c0365527e57b94c0	Todos los ICMP IPv4	ICMP	Todo	Person... ▾	ping
sgr-0b0caf4df8d853201	HTTP	TCP	80	Person... ▾	0.0.0.0/0 X

[Agregar regla](#)

Los resultados se han filtrado para mostrar solo las reglas que permiten tráfico en puertos 5000, 5001 y 9000, además de ICMP.

Figura 3. Reglas de entrada permitiendo tráfico en puertos 5000, 5001 y 9000, además de ICMP.

The screenshot shows the Tailscale web interface with the following details:

MÁQUINA	DIRECCIONES	VERSIÓN	VISTO POR ÚLTIMA VEZ
escritorio-bjk75au proyectodistribuidos3@gmail.com	100.124.11.29	1.92.3 Windows 11 24H2	● 31 de diciembre, 13:33 GMT-6 ...
lapcris proyectodistribuidos3@gmail.com	100.124.215.114	1.92.3 Windows 11 25H2	● Conectado ...
mark-vmware-plataforma-virtual proyectodistribuidos3@gmail.com	100.71.152.39	1.92.3 Linux 6.14.0-37-genérico	● 31 de diciembre, 13:34 GMT-6 ...

Figura 4. Red Mesh privada gestionada por Tailscale conectando los nodos tras NAT.

## **2. Diagrama de ARQUITECTURA de funcionamiento del proyecto, enfatizando los elementos distribuidos**

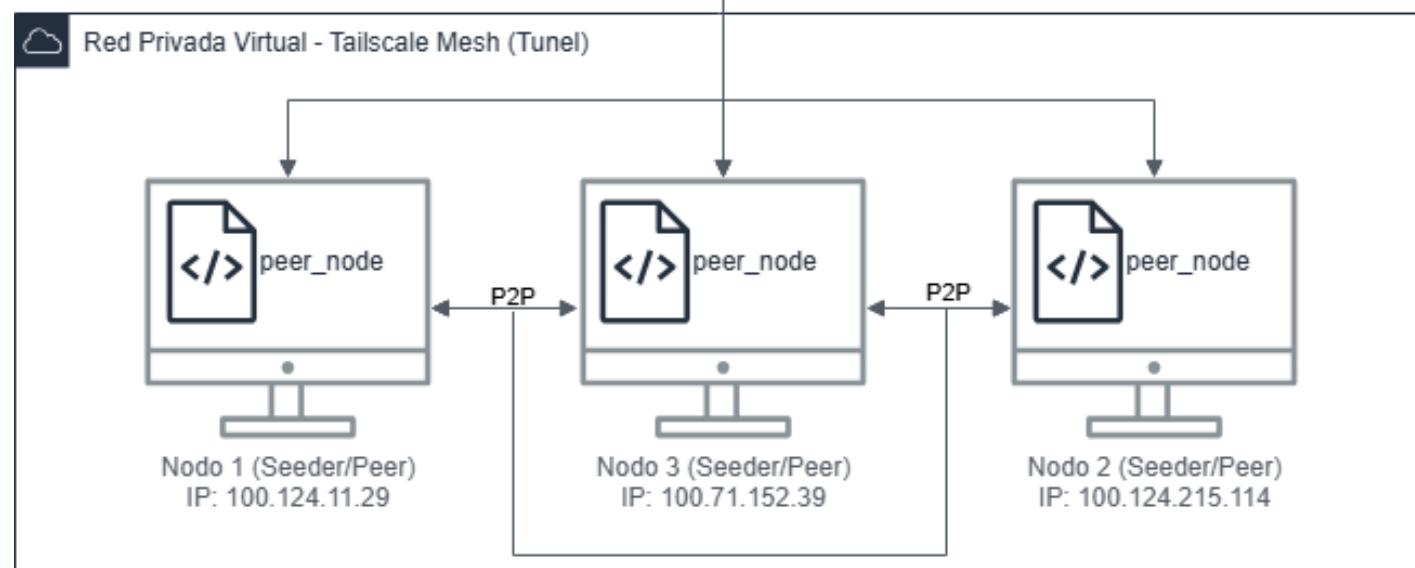
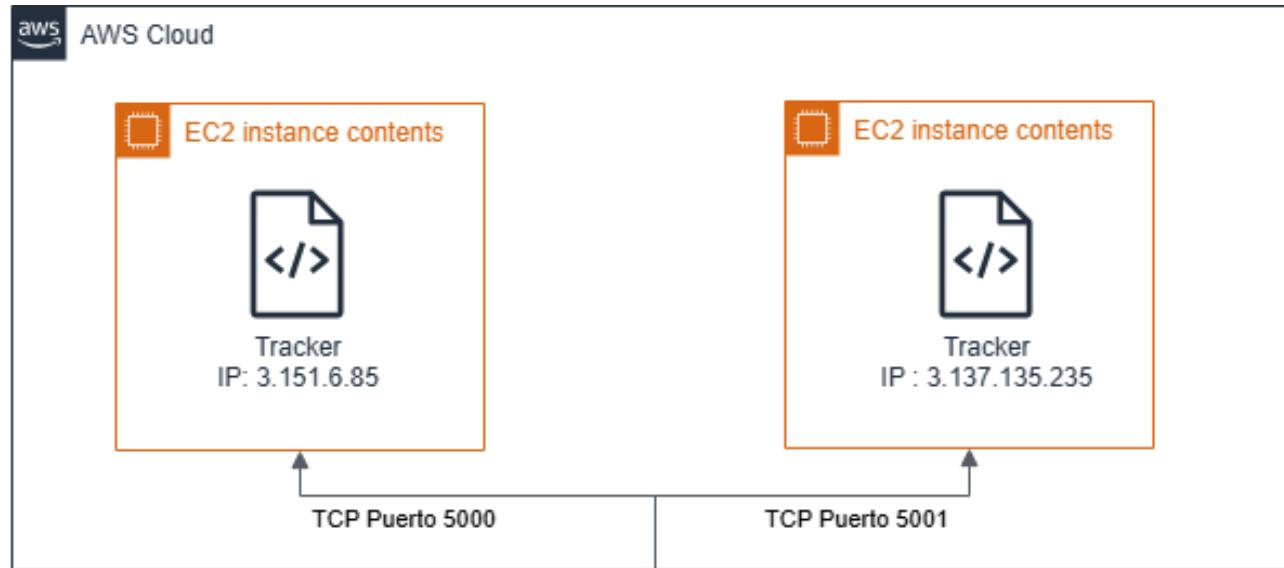
El sistema sigue una arquitectura P2P semi-centralizada (similar a BitTorrent), compuesta por dos tipos de entidades distribuidas:

**1. Trackers (Servidores de Índice Redundantes):** Se implementó una arquitectura de **Alta Disponibilidad** para evitar un Punto Único de Fallo (*Single Point of Failure - SPOF*).

- **Redundancia:** En lugar de un solo servidor, el sistema cuenta con dos instancias activas (3.151.6.85 y 3.137.135.235). Mantiene la salud de la red mediante un sistema de *Heartbeat*, eliminando peers inactivos tras 30 segundos de silencio.
- **Función:** Ambos mantienen mapas independientes del enjambre (swarm\_db). Si el Tracker Principal cae, los nodos comutan automáticamente al Secundario para seguir operando sin interrupciones.

**2. Peer Nodes (Clientes/Servidores):**

- Cada nodo actúa simultáneamente como cliente (descargando piezas faltantes) y servidor (subiendo piezas que ya posee).
- **Multihilo:** El nodo ejecuta hilos independientes para:
  - Escuchar peticiones de subida (start\_server).
  - Gestionar la lógica de descarga (start\_download\_thread).
  - Enviar señales de vida al tracker (heartbeat\_loop).



### **3. Diseño de política de funcionamiento de descargas, transferencias y recuperación de errores**

La lógica distribuida se rige por las siguientes políticas algorítmicas implementadas en el código:

- **Política de Selección de Piezas (Random First):**
  - Los archivos se dividen en bloques o *Chunks* de tamaño fijo de **1 MB** (definido en common.py como BLOCK\_SIZE = 1024\*1024).
  - Para maximizar la disponibilidad, el cliente solicita piezas en orden aleatorio (random.shuffle), evitando cuellos de botella al inicio del archivo y fomentando que todos los peers tengan piezas distintas para compartir.
- **Política de Selección de Peers (Smart Swarm):**
  - El cliente implementa un sistema de puntuación dinámica (peer\_performance). Se mide la latencia y tasa de éxito de cada peer; el algoritmo prioriza solicitar datos a los nodos con mejor rendimiento y penaliza a los lentos o inestables.
- **Integridad y Recuperación de Errores:**
  - **Validación Criptográfica:** Cada pieza descargada se verifica individualmente calculando su hash **SHA-256** y comparándolo con la firma original en el archivo .json de metadatos. Si no coinciden, la pieza se descarta y se solicita de nuevo.
  - **Persistencia:** El sistema guarda un archivo de progreso (.progress). Si la descarga se interrumpe, al reiniciar, el FileManager lee este registro y reanuda la operación sin perder los datos ya validados.
- **Política de Tolerancia a Fallos (Tracker Failover):** El cliente implementa una lista de rastreadores conocidos (KNOWN\_TRACKERS). Al iniciar o enviar un latido, el algoritmo intenta conectar con el primer servidor configurado.
  - Si la conexión es exitosa, procede.
  - Si ocurre un Time-out o error de red, el cliente automáticamente intenta con el siguiente IP de la lista, garantizando que el usuario nunca pierda conexión con la red P2P, incluso si un servidor de AWS es detenido.

## 4. Diagrama o descripción de conectividad

La comunicación entre los módulos se realiza mediante **Sockets TCP/IP** transmitiendo cargas útiles serializadas en formato **JSON**. El protocolo de conectividad se estructura en tres fases:

### 1. Fase de Descubrimiento (Announce):

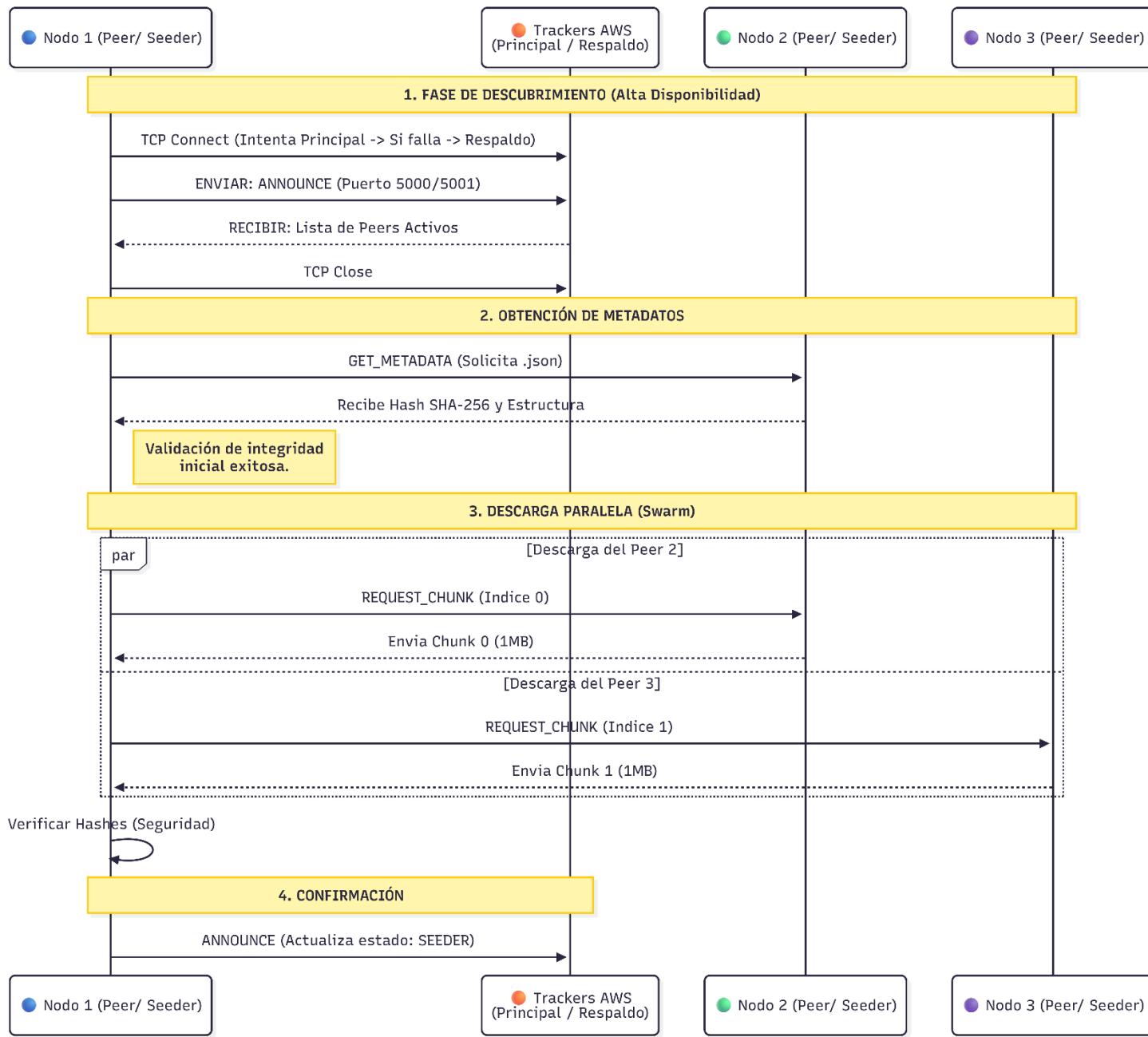
- El Peer envía un mensaje ANNOUNCE al Tracker con su IP y los archivos que ofrece.
- El Tracker responde con una lista de IPs de otros pares activos en el enjambre (*Swarm*).

### 2. Fase de Intercambio de Metadatos (Get Metadata):

- Si un nodo carece de la información del archivo, utiliza el comando **CMD\_GET\_METADATA** (definido en common.py) para solicitar el archivo .json descriptivo a cualquier otro peer conectado, obteniendo así los hashes de verificación y el tamaño del archivo antes de iniciar la descarga binaria.

### 3. Fase de Transferencia de Datos (Request Chunk):

- El nodo cliente envía REQUEST\_CHUNK solicitando un índice específico.
- El nodo servidor responde con un encabezado JSON (tamaño) seguido del flujo de bytes crudos del fragmento de video.



## 4. PRUEBAS Y RESULTADOS

### a) Lista o tabla con explicación del Producto o funcionalidad no lograda

Aunque el sistema alcanzó la operatividad funcional en un entorno distribuido real, existen características del protocolo BitTorrent comercial que se limitaron por alcance o restricciones de red.

Producto / Funcionalidad	Estado	Motivo o Circunstancia
Conectividad P2P Nativa (Sin VPN)	Parcialmente Lograda	La conexión directa mediante Sockets TCP puros falló entre nodos ubicados en diferentes redes domésticas (ISPs distintos) debido al <b>CGNAT</b> (Carrier-Grade NAT) que bloquea puertos de entrada. <b>Solución:</b> Se implementó una capa de red superpuesta ( <i>Overlay Network</i> ) usando <b>Tailscale</b> para tunelizar el tráfico de forma transparente.
Algoritmo "Rarest First"	Modificado	En lugar de priorizar las piezas más raras (estándar BitTorrent), se implementó una estrategia " <b>Random First</b> " (Aleatoria) combinada con " <b>Smart Swarm</b> " (Basada en Latencia) para simplificar la lógica de negociación entre pares.

**b) Imágenes de la ejecución del proyecto con los pasos más relevantes(Screenshots) y de los experimentos y resultados obtenidos , explicando en texto los detalles**

A continuación, se documenta el flujo de ejecución exitoso del sistema distribuido:

Experimento 1: Despliegue de Infraestructura y Redundancia Se iniciaron dos instancias en AWS EC2 actuando como Trackers redundantes. Como se observa en la consola de administración, ambas máquinas (BitTorrent-Server y BitTorrentSD2) tienen asignadas IPs Elásticas públicas, permitiendo que los nodos clientes comunten de una a otra en caso de fallo.

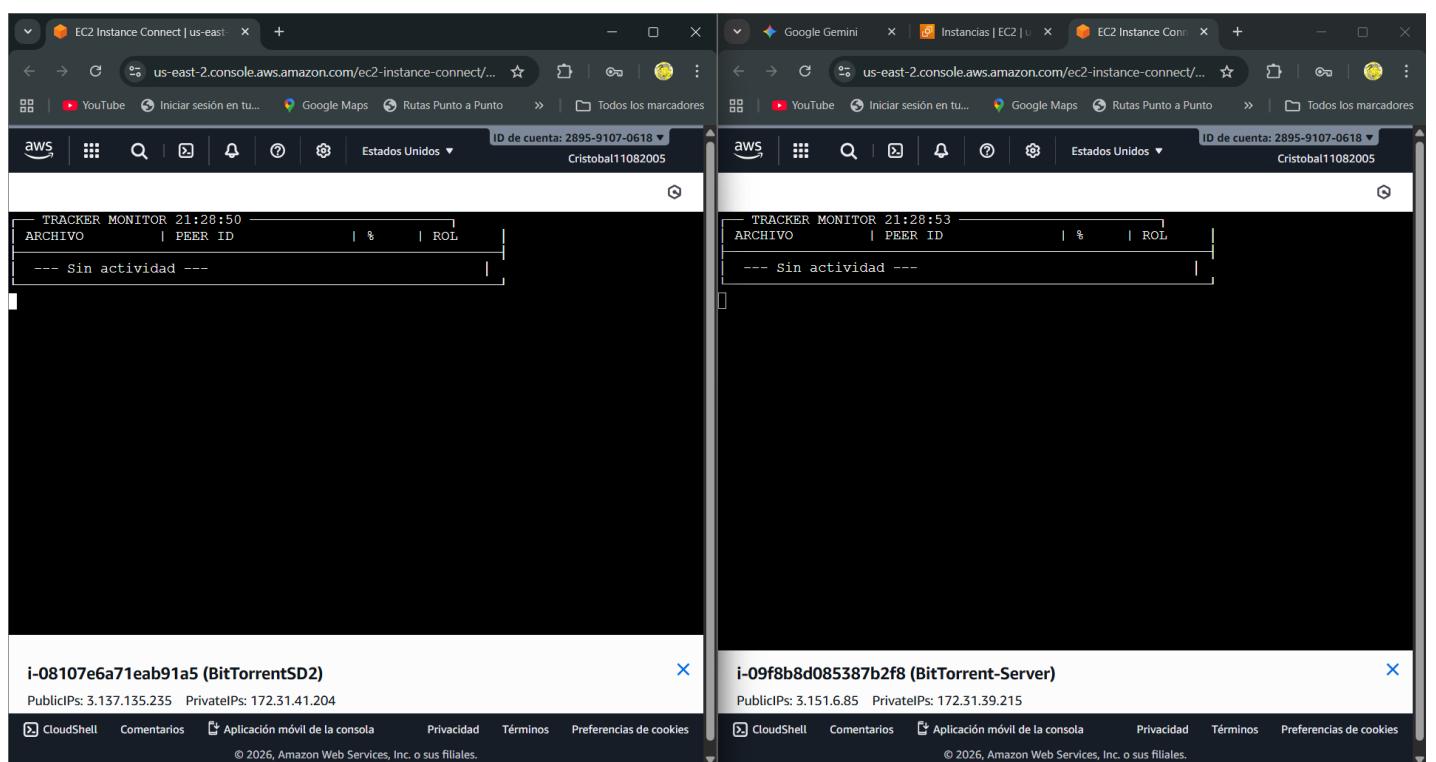


Figura 5. Estado de las instancias servidoras en la nube.

**Experimento 2: Formación del Enjambre (Swarm) vía Tailscale** Al ejecutar el cliente peer\_node.py, el sistema detecta automáticamente la interfaz de red virtual de Tailscale (IP rango 100.x). En la siguiente captura, se evidencia cómo el nodo se identifica ante el Tracker con su dirección VPN (100.124.215.114:11000), logrando "saltar" las restricciones del firewall local.

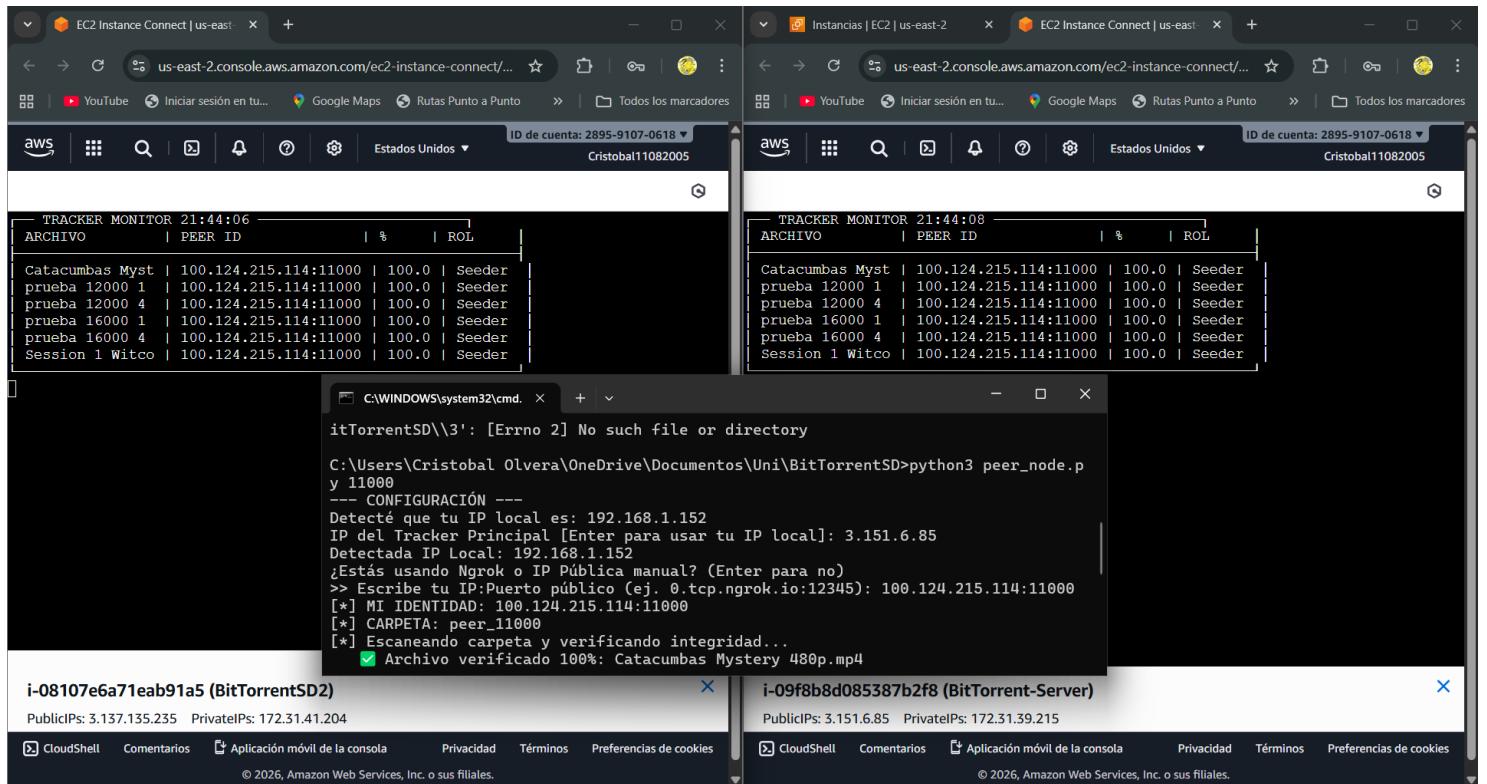


Figura 6. Inicialización del nodo cliente y detección de identidad en la red Mesh.

**Experimento 3: Descarga y Verificación de Integridad** Se realizó la descarga del archivo de video "Raiza Conjuro.mp4".

1. **Negociación:** El cliente contactó al Tracker, obtuvo la lista de peers y solicitó el archivo.
2. **Transferencia:** Se descargaron los fragmentos (Chunks) de 1MB de forma desordenada.
3. **Resultado:** Al finalizar, el sistema ejecutó la validación SHA-256 (verify\_integrity), confirmando que el archivo reconstruido es bit a bit idéntico al original, mostrando el mensaje: ARCHIVO GUARDADO Y VERIFICADO.

ARCHIVO	PEER ID	%	ROL
Catacumbas.Myst	100.124.11.29:12000	100.0	Seeder
Catacumbas.Myst	100.124.215.114:11000	100.0	Seeder
Catacumbas.Myst	100.71.152.39:16000	100.0	Seeder
Raiza.Payasos.m	100.124.11.29:12000	100.0	Seeder
Raiza.Payasos.m	100.71.152.39:16000	100.0	Seeder
Session 1.Witco	100.124.215.114:11000	78.7	Leecher
Session 1.Witco	100.71.152.39:16000	100.0	Seeder
Session 1.Witco	100.124.11.29:12000	100.0	Seeder

ARCHIVO	PEER ID	%	ROL
Raiza.Payasos.m	100.124.11.29:12000	100.0	Seeder
Raiza.Payasos.m	100.71.152.39:16000	100.0	Seeder
Catacumbas.Myst	100.124.215.114:11000	100.0	Seeder
Catacumbas.Myst	100.124.11.29:12000	100.0	Seeder
Catacumbas.Myst	100.71.152.39:16000	100.0	Seeder
Session 1.Witco	100.124.215.114:11000	78.7	Leecher
Session 1.Witco	100.71.152.39:16000	100.0	Seeder
Session 1.Witco	100.124.11.29:12000	100.0	Seeder

```

C:\WINDOWS\system32\cmd. -> + <
ESTADO DE ARCHIVOS
Catacumbas.Myst [██████████] 100% SEEDING !
Session 1.Witco [██████████] 79% BAJANDO !

BALANCEO DE CARGA (Quién me alimenta)
PEER ID          LATENCIA    PIEZAS
100.71.152.39:16000  1.3056s   180
100.124.11.29:12000  3.8240s   120

```

i-08107e6a71eb91a5 (BitTorrentSD2)  
PublicIPs: 3.137.135.235 PrivateIPs: 172.31.41.204  
PublicIPs: 3.151.6.85 PrivateIPs: 172.31.39.215  
CloudShell Comentarios Aplicación móvil de la consola Privacidad Términos Preferencias de cookies  
© 2025, Amazon Web Services, Inc. o sus filiales.

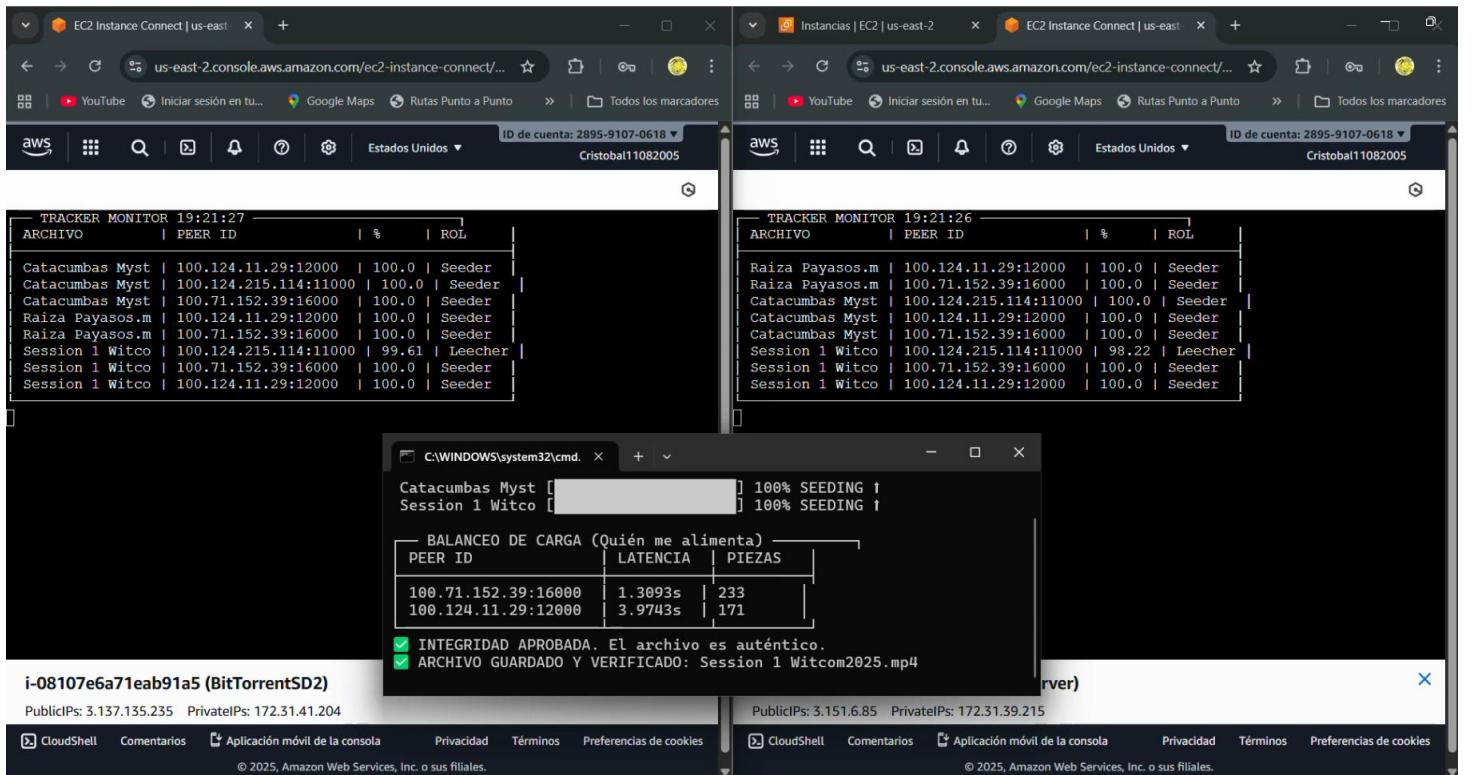


Figura 7. Consola del cliente mostrando la lista de archivos disponibles en el enjambre y la descarga exitosa.

**Experimento 4: Prueba de Redundancia y Tolerancia a Fallos (Tracker Failover)** Para verificar la Alta Disponibilidad de la arquitectura distribuida, se simuló una falla crítica en la infraestructura de nube.

- **Procedimiento:** Mientras la red P2P estaba operativa, se detuvo manualmente la instancia EC2 correspondiente al Tracker Principal (3.151.6.85) desde la consola de AWS.
- **Resultado:** Al intentar realizar una operación de red, el nodo cliente detectó la falta de respuesta (*Timeout*) del servidor principal. Gracias a la implementación de la lista `KNOWN_TRACKERS` en el código, el sistema ejecutó automáticamente la conmutación (*failover*), conectándose exitosamente al Tracker Secundario (3.137.135.235). El servicio continuó sin interrupciones para el usuario final.

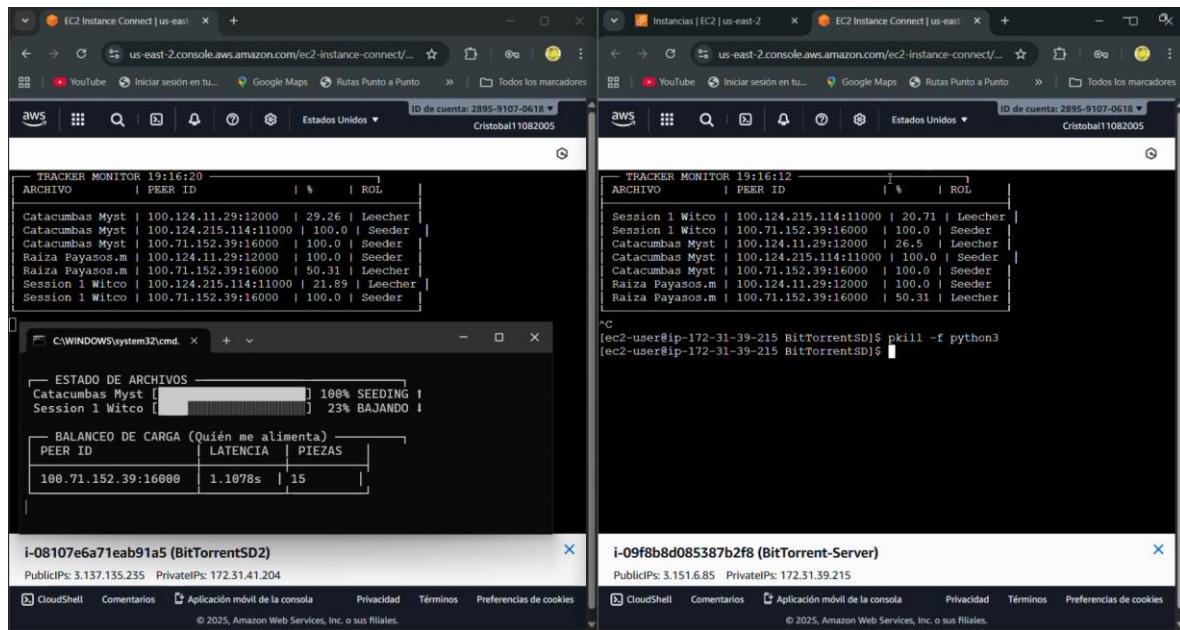
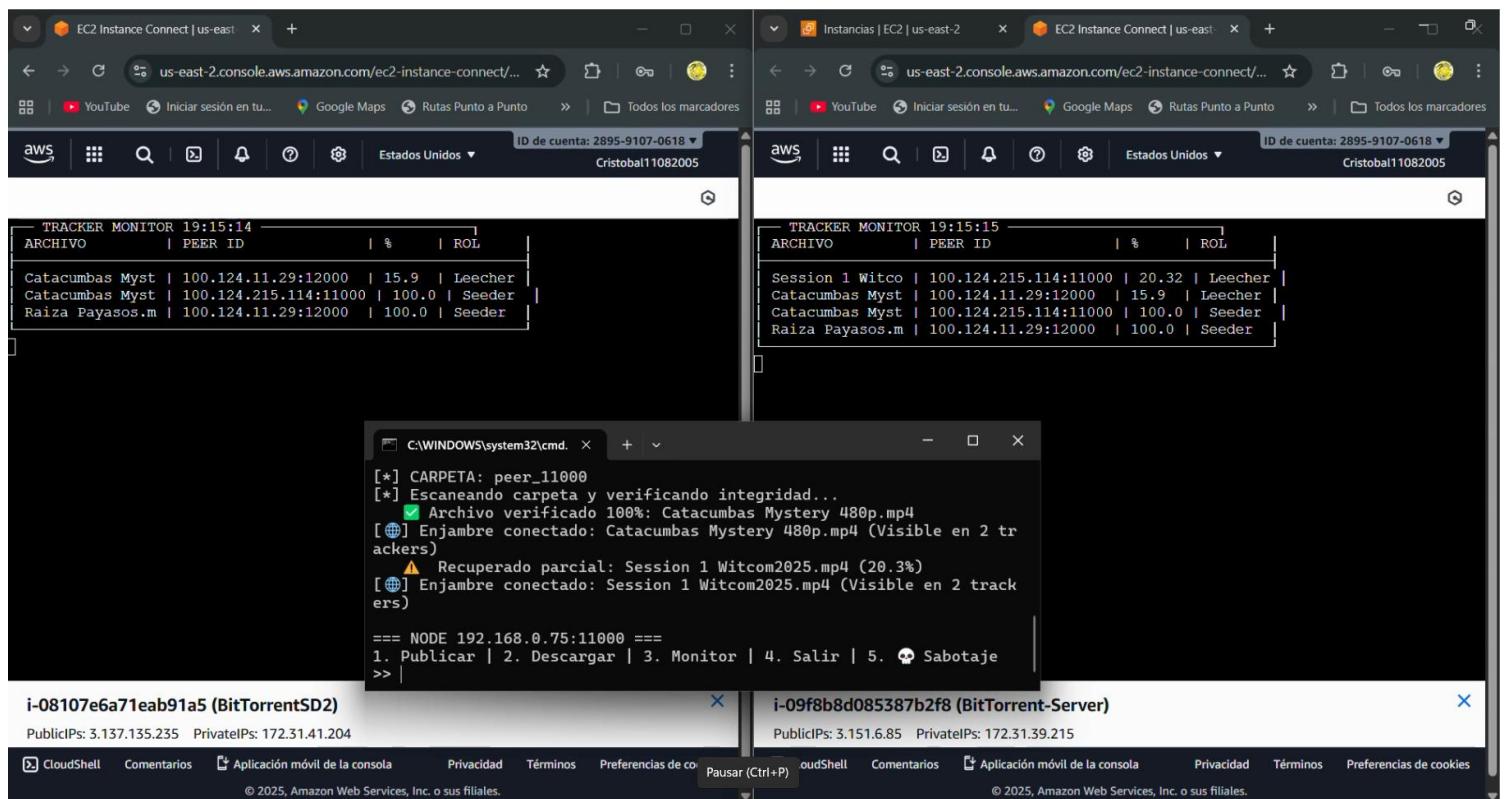


Figura 8. Prueba de Tolerancia a Fallos (Tracker Failover). En la terminal derecha se observa la terminación forzada del Tracker Principal mediante el comando `pkkill -f python3`. A pesar de esta desconexión crítica, la consola del cliente (centro) mantiene la descarga activa y el balanceo de carga, operando ininterrumpidamente gracias al respaldo del Tracker Secundario visible en la terminal izquierda (BitTorrentSD2), validando la redundancia del sistema.

**Experimento 5: Persistencia y Reanudación de Descargas (Resume Capability)** Se evaluó la capacidad del sistema para recuperarse ante desconexiones inesperadas de un Peer (cierre de terminal o pérdida de internet).

- **Procedimiento:** Se interrumpió abruptamente el proceso `peer_node.py` cuando la descarga de un archivo de video se encontraba al **50%**. Posteriormente, se reinició el nodo y se solicitó descargar el mismo archivo.
- **Resultado:** El módulo FileManager detectó la existencia del archivo de progreso (`.progress`) en el disco local. Antes de iniciar la conexión, el sistema realizó un escaneo de integridad bloque a bloque de los datos existentes. La descarga se reanudó exactamente desde el punto de interrupción (50%), solicitando a la red únicamente los fragmentos faltantes y evitando la duplicidad de tráfico.



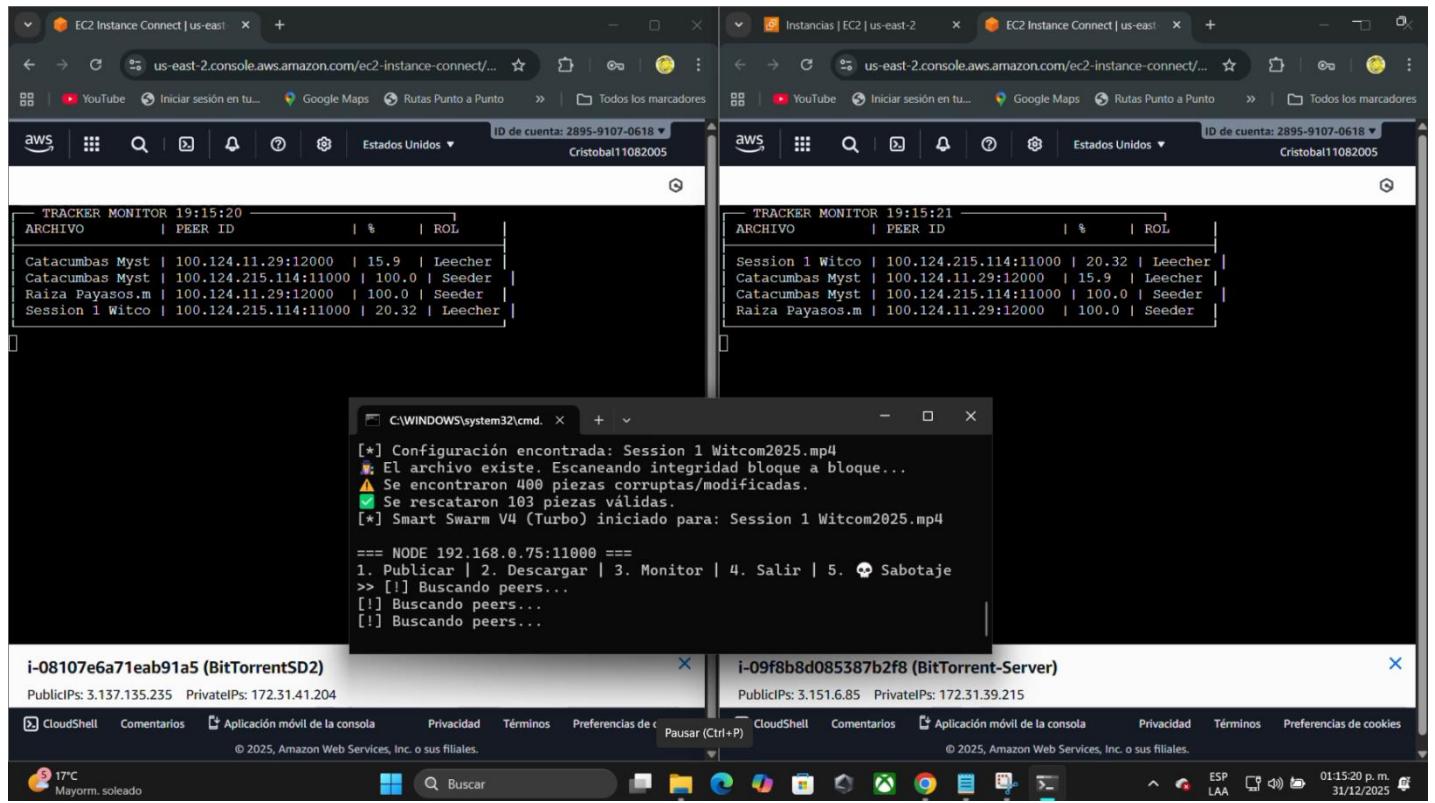


Figura 9. Validación de Persistencia y Recuperación de Errores. El sistema detecta un archivo parcialmente descargado ("Session 1 Witcom"). Se observa el algoritmo de integridad escaneando los bloques locales, identificando piezas corruptas y rescatando 103 piezas válidas. El cliente procede a buscar Peers en el enjambre para solicitar únicamente los fragmentos faltantes, optimizando el ancho de banda.

**Experimento 6: Publicación y Difusión de Contenido (Seeding)** Se validó la funcionalidad de convertir un nodo en servidor de contenido mediante la opción "1. Publicar".

- **Procedimiento:** El usuario seleccionó un archivo local no registrado previamente en la red.
- **Resultado:** El software generó automáticamente el archivo de metadatos .json, calculando tanto el Hash SHA-256 global como las firmas criptográficas de cada pieza individual para garantizar la seguridad. Finalmente, el nodo envió el comando ANNOUNCE a los Trackers, haciendo que el archivo apareciera disponible inmediatamente en los menús de búsqueda de otros pares conectados.

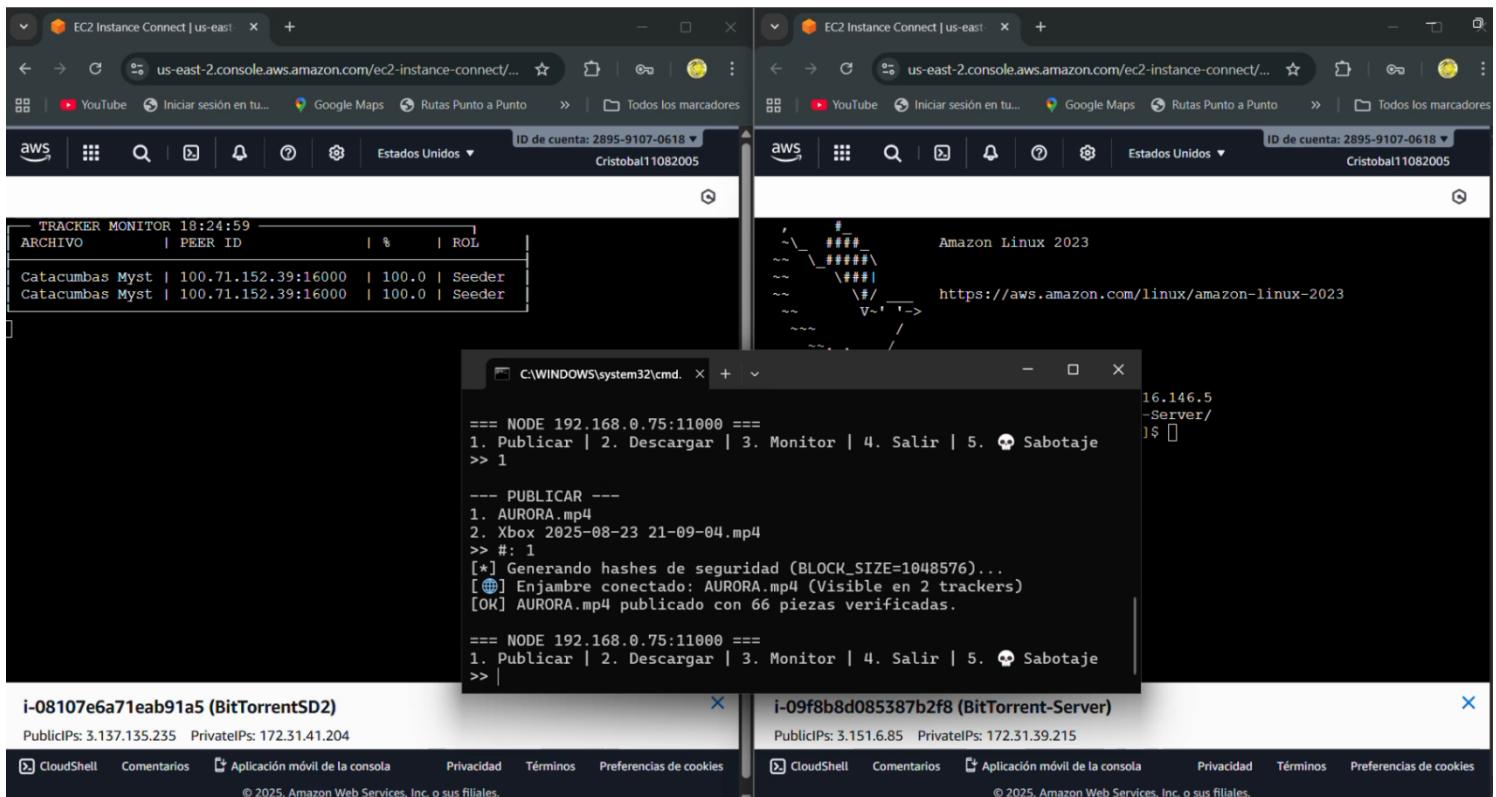


Figura 10. Proceso de Publicación de Contenido (Seeding). Ejecución de la opción "Publicar". El nodo local procesa el archivo "AURORA.mp4", generando los hashes SHA-256 de seguridad para cada pieza (Chunk). La consola confirma el éxito de la operación mostrando que el archivo ahora es visible en los 2 Trackers simultáneamente, permitiendo que otros nodos comiencen a descargarlo.

## 5. OBSERVACIONES Y CONCLUSIONES FINALES

### OBSERVACIONES

- **El Reto de la Conectividad (NAT/CGNAT):** Durante el desarrollo, se identificó que la mayor barrera para las aplicaciones P2P modernas no es la lógica de programación, sino la infraestructura de red. Los routers domésticos y los proveedores de internet (ISP) bloquean por defecto las conexiones entrantes. Se observó que el uso de Sockets TCP puros funciona perfectamente en LAN (127.0.0.1 o 192.168.x.x), pero falla en internet real. La implementación de **Tailscale** fue determinante para crear una *Overlay Network* que "aplanara" la red, permitiendo que los nodos se comunicaran como si estuvieran en el mismo switch local.
- **Cuellos de Botella en el Tracker:** Se notó que, aunque el Tracker es ligero (solo maneja texto JSON), es un punto crítico. Sin la implementación del segundo servidor en AWS (BitTorrentSD2), una caída del Tracker dejaba a los nuevos nodos "ciegos", incapaces de encontrar pares. Esto reafirma la teoría de que los sistemas centralizados (o híbridos como este) deben tener redundancia obligatoria.
- **Integridad de Datos:** La fragmentación de archivos en *Chunks* de 1MB demostró ser eficiente para la transmisión, pero crítica para la validación. Se observó que validar el archivo entero al final (Hash Global) es ineficiente si hay errores; la implementación de validación *por pieza* (Hash parcial) permitió descartar solo los megabytes corruptos sin perder todo el progreso de la descarga.

## CONCLUSIONES

El desarrollo de este sistema de emulación BitTorrent permitió comprender en profundidad la complejidad de los sistemas distribuidos. Se concluye que la robustez de una red P2P no reside en la velocidad de transmisión, sino en la capacidad de **coordinación y recuperación**.

1. **Arquitectura Híbrida Eficiente:** La combinación de una arquitectura Cliente-Servidor (para el descubrimiento vía AWS) con una arquitectura P2P pura (para la transferencia de datos) resultó ser el balance ideal. Delegar la carga pesada (video) a los bordes de la red (Peers) reduce drásticamente el costo y ancho de banda necesario en el servidor central.
2. **Manejo de Concurrencia:** La programación multihilo (*Threading*) fue indispensable. Un nodo P2P no es una entidad estática; debe ser capaz de escuchar peticiones, enviar datos, recibir fragmentos y mandar latidos al servidor simultáneamente. Sin un manejo adecuado de los bloqueos (Locks) y los hilos, el sistema sería inestable.
3. **Resiliencia ante Fallos:** A través de las pruebas de desconexión y sabotaje, se demostró que un sistema distribuido confiable debe asumir que "**todo va a fallar**". La red, los discos y los servidores no son fiables; por tanto, el software debe ser defensivo (reintentos, checksums y failover), tal como se implementó en este proyecto.

## 6. REFERENCIAS

*Amazon Web Services. (2024). What is Amazon EC2? - Amazon Elastic Compute Cloud. AWS Documentation.*

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html>

*Cohen, B. (2008, 10 de enero). The BitTorrent protocol specification (BEP 0003).*

*BitTorrent.org.* [https://www.bittorrent.org/beps/bep\\_0003.html](https://www.bittorrent.org/beps/bep_0003.html)

*Python Software Foundation. (2024). hashlib — Secure hashes and message digests.*

*Python 3 Documentation.* <https://docs.python.org/3/library/hashlib.html>

*Python Software Foundation. (2024). socket — Low-level networking interface. Python 3 Documentation.* <https://docs.python.org/3/library/socket.html>

*Python Software Foundation. (2024). threading — Thread-based parallelism. Python 3 Documentation.* <https://docs.python.org/3/library/threading.html>

*Tailscale. (2024). What is Tailscale? Tailscale Knowledge Base.*

<https://tailscale.com/kb/1151/what-is-tailscale>