Universidad Mariano Galvez de Guatemala
Facultad de Ingeniería en Sistemas y Ciencias de la Computación
Redes I
Ingeniero Boche

Proyecto final de Redes I

Diana Paola Rivas Arana
0907-22-15036
José Andrés Alvarez Cardona
0907-22-11608
Juana Yessenia Ramírez Santiago
0907-22-13755

29/10/2025

Proyecto Final de Curso Redes de Computadoras II

Como parte del Proyecto Final de Redes se realizó una implementación de un protocolo de red propio basado en el modelo de referencia OSI que permite ver cómo se encapsulan los datos, cómo se gestiona la sesión y cómo se transfieren datos fiables en red. Se ha construido desde cero las capas fundamentales de comunicación en la red las cuales son:

1) Capa de Aplicación: la cual se relaciona con cliente.py y permite ver la interfaz de usuario.
2) Capa de Sesión: Encargada de gestionar las sesiones y el UUID.
3) Capa de Transporte: Encargada del Protocolo confiable y la Fragmentación.
4) Capa de Red: Estructura preparada para routing.
5) Capa de Enlace: Encargada para el manejo de tramas

El Proyecto se desarrolló con Python 3.7 utilizando bibliotecas asincronicas permitiendo que la web maneje diferentes operaciones, tiene frontend web (React) integrado, se implementó Docker para el despliegue, Websocket para el chat en el tiempo real, uso y Fast API el cual es el encargado de coordinar las comunicaciones. El código se puede encontrar en el siguiente enlace:

https://github.com/Andres3f/Proyecto_Redes

Avance presentado el 18/10/2025:

Como primer avance se presentó la estructura básica y funcional de nuestro código, el cual cumple con el intercambio de datos básico (formato texto) y se implementó el modo FIABLE y el SEMI-FIABLE para el envío de mensajes de control y la transmisión de archivos con límite de MB, también tiene la funcionalidad de detectar pérdidas . Se definieron los encabezados básicos de nuestras capas.

La estructura inicial del código es:

```
.
├── docker-compose.yml          # Configuración de servicios Docker
├── Dockerfile.python           # Dockerfile para servicios Python
├── ejecutar_programa.py        # Script principal del servidor
├── image_server.py             # Servidor de imágenes
├── requirements.txt            # Dependencias Python principales
├── docs/                       # Documentación
│   ├── especificaciones.md
│   └── ROADMAP.md
├── frontend/                   # Interfaz de usuario web
│   ├── Dockerfile
│   ├── src/
│   │   ├── App.jsx
│   │   └── main.jsx
│   ├── index.html
│   └── nginx.conf
├── frontend_api/               # API para el frontend
│   ├── main.py
│   └── requirements.txt
├── src/                        # Código fuente principal
│   ├── app/
│   │   └── cliente.py
│   ├── enlace/
│   ├── red/
│   ├── sesion/
│   └── transporte/
│       ├── fragmentation.py
│       └── reliable.py
└── tests/                      # Pruebas unitarias
    ├── test_fragmentation.py
    └── test_transfer.py
```

(La estructura se puede encontrar en el repositorio de Github: https://github.com/Andres3f/Proyecto_Redes)

Como se puede observar, cada capa se encuentra organizada en su respectiva carpeta y cada una incluye su respectivo encabezado. Los encabezados en el código son la información de control que cada capa agrega a los datos originales con el fin de garantizar la entrega de datos.

Encabezados Implementados en Cada Capa:

- **Capa de Aplicación (src/app/cliente.py):** Identifica qué se está enviando y en qué formato.
  Encabezado implementado:
  app_header = {
  "filename": os.path.basename(filepath), //Guarda el nombre del archivo enviado.
  "size": os.path.getsize(filepath), //Puede validar que el tamaño recibido sea correcto
  "mime_type": mimetypes.guess_type(filepath)[0], //interpreta los datos (texto, imagen, video) y localiza si se necesita descomprimir el archivo después de recibir.
  "timestamp": datetime.now().isoformat() }

- **Capa de sesión (src/sesion/pruebasesion.py):** Identifica a quién pertenece esta comunicación y mantener el contexto.
  import uuid
  session_header = {
  "session_id": str(uuid.uuid4()), //Permite distintos usuarios simultáneamente gracias al identificador UUID además de funcionar como un contador de mensajes para la base de datos.
  "created_at": datetime.now().isoformat(),//Permite identificar cuando se inicio sesion
  "user": username,//Permite guardar el nombre del usuario y reconectar la sesión gracias el identificador
  "active": True//Permite saber si la sesión sigue activa
  }

- **Capa de Transporte (`src/transporte/reliable.py y fragmentation.py`):** Divide los datos grandes y garantiza la entrega ordenada y confiable del archivo.
  def create_fragment(data, fragment_id, total):
  transport_header = {
  "fragment_id": fragment_id, //Identifica el número de fragmento actual dentro de la secuencia
  "total": total, // Indica cuántos fragmentos componen el mensaje completo, lo que permite al receptor saber cuándo ha recibido todos.
  "checksum": calculate_crc32(data),//Verificar que los datos no se corrompieron
  "size": len(data),
  "timestamp": time.time()
  }
  return {
  "header": transport_header,
  "data": data
  }
- **Capa de Red (`src/red/`):** Enrutar el paquete desde origen hasta destino a través de múltiples redes.
  {
  "network_layer": {
  "source_ip": "192.168.1.100",// De dónde viene y adónde va
  "destination_ip": "192.168.1.105",
  "ttl": 64,
  "protocol": "custom_reliable",
  "packet_id": "pkt_12345"
  }
  }
- **Capa de Enlace:(src/enlace/):** Encargada de la encapsulación a nivel de enlace y el acceso a redes compartidas.

Observaciones: Por recomendaciones del Ingeniero, se pidió la implementación de mensajes instantáneos entre los usuarios en distintos dispositivos además de la visualización del flujo de los encabezado implementados en el envío de los mensajes.

El día sábado 25 de octubre realizamos la entrega final del proyecto, en el cual se implementó Websocket para la comunicación bidireccional y la transferencia instantánea de los mensajes, la visualización de las capas en la interfaz del usuario, la autenticación de los usuario y por último el envío de los mensajes en tiempo real en distintos dispositivo ofreciendo un soporte para ngrok que permite el acceso remoto.