Licenciatura en Tecnología Digital
TD4: Redes de Computadoras

Trabajo Práctico

Segundo semestre 2025

Introducción

En este trabajo práctico se explorarán distintos aspectos de la capa de aplicación en el modelo TCP/IP y el funcionamiento de los sockets TCP. El objetivo es implementar un servidor web minimalista que permita la transferencia de archivos entre dispositivos conectados en una misma red local. El servidor deberá permitir subir o descargar archivos según el modo seleccionado, y mostrar en consola un código QR que facilite la conexión al servidor desde otro dispositivo (por ejemplo, un teléfono celular).

Contexto

En muchas aplicaciones actuales, como servicios de almacenamiento en la nube o aplicaciones de mensajería, los archivos se transfieren entre clientes y servidores utilizando protocolos estándar como HTTP sobre TCP. Comprender cómo funcionan estas transferencias a nivel de red permite:

- Diagnosticar problemas de conectividad y rendimiento.
- Implementar servidores seguros y eficientes.
- Gestionar correctamente la información recibida y enviada.

En este TP, implementarán un servidor que puede operar en dos modos:
Download: Servidor que permite a un cliente descargar un archivo específico.
Upload: Servidor que permite a un cliente subir un archivo al servidor.

Para correr el servidor deberán escribir por terminal **`python server_fileTransfer.py download {nombre archivo a descargar}`** en el caso del modo download y **`python server_fileTransfer.py upload`** para correrlo en modo upload. Esto ya se encuentra implementado en el main del archivo proporcionado.

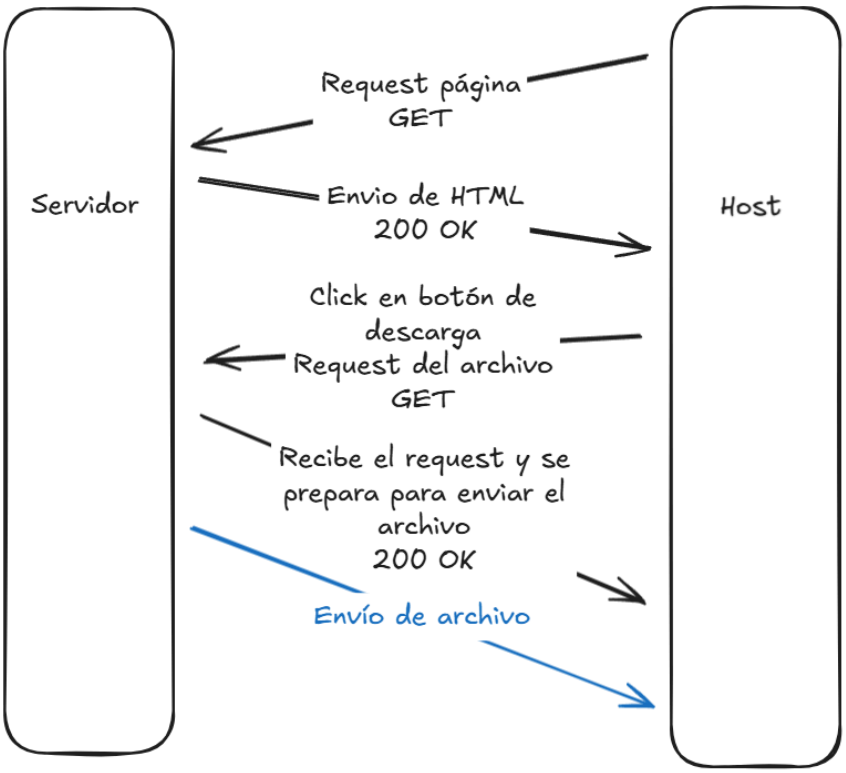
Luego el funcionamiento del servidor deberá ser el siguiente:

El servidor se inicia en modo descarga o carga, y se pone a escuchar en un puerto. Se imprime la URL del servidor y el formato QR de la misma por terminal. La URL del servidor está compuesta por su IP y número de puerto de la forma: **`http://{ip}:{puerto}`**. En este punto el servidor está listo para que un host se conecte escaneando el QR. Cuando el host escanea el QR realiza un **`HTTP GET`** en el que pide la página, el HTML correspondiente a la URL. El servidor debe enviar el HTML al host.

En el caso de una **descarga**, una vez que el host lo recibe puede pulsar el botón de descarga presente en la página lo cual dispara otro **`HTTP GET,`** pero en este caso se pide el archivo a descargar. El servidor, al recibir esta request, debe contestar con un **`200 OK`** en caso de que el archivo pedido esté

presente en la carpeta **`archivos_servidor`** del servidor y luego enviarlo. En caso de no contar con el archivo, deberán devolver un error **`404 Not Found`**.

En el caso de la **carga** de un archivo, el host al pulsar el botón de subida envía un POST con la etiqueta **`Content-Type:multipart/form-data`** que contiene el archivo a guardar en el servidor. El servidor debe recibir este mensaje e interpretarlo para obtener el nombre de archivo y el archivo en sí. Una vez interpretado debe guardarlo en la carpeta **`archivos_servidor`** y responder al host con **`200 OK`** en caso de éxito o **`500 Internal Server Error`** en caso de error al guardar el archivo.



Se proporcionará un código base que deberán completar y ampliar para cumplir con todas las consignas. Si encuentran algún bug en el código por favor avisar.

El código base cuenta con las siguientes funciones auxiliares:

- **`imprimir_qr_en_terminal(url)`**: Dada una URL genera un QR y lo imprime por terminal. Esta es la única función auxiliar que deberán completar.
- **`get_wifi_ip()`**: Obtiene la IP asociada a la interfaz de red del equipo donde lo ejecuten.
- **`parsear_multipart(body, boundary)`**: Dado el body del mensaje **`HTTP POST`** enviado por el host y el boundary extrae el nombre del archivo y su contenido. El boundary es un string que se encuentra en el header del **`POST`** en la misma línea que **`Content-Type:multipart/form-data`**. Se utiliza para separar los bytes del archivo enviado y sus metadatos en el body del **`POST`**.

- **`generar_html_interfaz(modo)`**: Genera un HTML a enviar según el modo en el que se inició el servidor. Se les proporciona un HTML básico pero pueden modificarlo.

Consignas

Nota: *En caso de utilizar una de las computadoras provistas por la Facultad para los alumnos de Tecnología Digital, será necesario implementar el trabajo dentro de una máquina virtual (por ejemplo, Oracle VirtualBox, que ya se encuentra instalada). Esto se debe a que varias de las funcionalidades requeridas en el trabajo están bloqueadas por restricciones de permisos del sistema.*

Completar las funciones del servidor:

- **`imprimir_qr_en_terminal(url)`**: Dada una URL deberán imprimirla por terminal como QR usando la librería **qrcode.**
- **`manejar_descarga(archivo, request_line)`**: Generar la respuesta HTTP para descargar un archivo.
- **`manejar_carga(body, boundary, directorio_destino=".")`**: Procesar un **POST** con tipo de contenido multipart/form-data y guardar el archivo recibido.
- Configurar correctamente el socket TCP en **`start_server()`** para aceptar conexiones y responder al cliente.

Una vez completado deberían poder conectarse al servidor con algún dispositivo conectado en la misma red y poder descargar/cargar algún archivo como una imagen o un pdf. Antes de avanzar con las siguientes consignas realicen un par de intentos con distintos archivos para asegurarse de que esto es posible.

Análisis de protocolos y headers HTTP:

Capturar los paquetes de la comunicación usando Wireshark, capturar los paquetes del protocolo HTTP. Identificar los métodos HTTP, los headers y el payload de la transferencia de archivos.

Implementación de compresión gzip:

El servidor debe ser modificado para incorporar soporte de compresión gzip, cumpliendo con los siguientes requisitos:

- Detección del soporte del cliente: verificar si el cliente acepta compresión mediante el header **`Accept-Encoding: gzip.`**
- Compresión de archivos: comprimir los archivos antes de enviarlos utilizando el módulo **`gzip`** de Python.
- Encabezado de respuesta: incluir el header **`Content-Encoding: gzip`** en las respuestas comprimidas.
- Configuración por línea de comandos: permitir activar o desactivar la compresión mediante un parámetro opcional al ejecutar el servidor. (esto les va a servir a la hora de hacer la experimentación)

Experimentos de compresión

Realizar experimentos con **al menos dos** archivos de distinto tipo, por ejemplo: una imagen (**`.jpg`, `.png`**) y un documento (**`.pdf`, `.docx`**). Para cada tipo de archivo, se deberán probar diferentes tamaños: Pequeño (< 100 KB), Mediano (100 KB – 5 MB), Grande (> 5 MB).

Durante las pruebas, deberán medir y registrar los siguientes valores:

- Tamaño original (en bytes)
- Tamaño comprimido (en bytes)
- Ratio de compresión = tamaño original / tamaño comprimido (puede expresarse en porcentaje haciendo: (1 - ratio de compresión) x100)
- Tiempo de transferencia sin compresión
- Tiempo de transferencia con compresión

Entendiendo tiempo de transferencia como tiempo desde que se recibe la petición el servidor hasta que envía la respuesta completa. También es importante que aclaren si están usando máquina virtual porque puede distorsionar las mediciones.

Realizar **al menos un** gráfico comparativo, a elección del grupo, por ejemplo: un gráfico de barras comparando tamaños originales vs. comprimidos. O bien, un gráfico comparando tiempos de transferencia con y sin compresión.

Análisis de seguridad:

El servidor implementado presenta vulnerabilidades de seguridad que deben ser identificadas y mitigadas. Trudy, una atacante en la misma red local, quiere interceptar, robar archivos de su servidor y no dejar que usen su servidor.

En términos de confidencialidad, integridad y disponibilidad den cuenta de **al menos dos** maneras en las que podría lograr alguno de los objetivos que se propone. Las amenazas deben ser en relación al servidor implementado por ustedes y deben explicitar qué parte/componente del servidor es el que se ve vulnerado y posibilita ese ataque. ¿Qué protocolos están usando que no son seguros? ¿Qué chequeos no están realizando?

Para cada amenaza identificada, proponer **al menos una** medida de seguridad que alivie el problema. No deben implementarla pero debe ser lo más específica posible. Si proponen cambiar un protocolo digan por qué y en qué parte del servidor deberían cambiar algo, en general, no hace falta proponer código. Si proponen chequear algo en qué momento deberían hacerlo, por ejemplo cuando llega una request.

BONUS: implementar un método básico de autenticación en el servidor que limite el acceso únicamente a clientes autorizados. Por ejemplo, **autenticación mediante contraseña,** antes de permitir la descarga o subida de archivos, el servidor solicita al cliente ingresar una contraseña

conocida únicamente por los usuarios de confianza. El servidor verifica que la contraseña recibida coincida con la configurada localmente antes de continuar con la operación.

(Este ejercicio bonus es opcional y su puntaje es adicional al del trabajo. Sólo se considerará si hubo un intento de implementación y no resta puntos en caso de errores o fallas.)

Informe

Deberán realizar un informe de entre 5 y 10 páginas (sin contar la carátula) que documente las decisiones relevantes tomadas durante la implementación, los errores encontrados y, en caso de haberlos solucionado, el proceso seguido para hacerlo. Además, deberán registrar de forma detallada los puntos de análisis y experimentación solicitados en la consigna.

El informe deberá tener como mínimo los siguientes apartados:

Introducción:
Breve descripción del objetivo del trabajo práctico, el contexto de la capa de aplicación en el modelo TCP/IP y la finalidad del servidor implementado.

Implementación:
Descripción técnica de las decisiones de diseño y desarrollo tomadas para cumplir con las consignas. Se deben explicar las funciones principales implementadas (por ejemplo: manejo de sockets, procesamiento de requests, envío/recepción de archivos, generación de QR, manejo de errores). También se deben mencionar los problemas encontrados y las soluciones aplicadas.

Experimentación:
Registro de las pruebas realizadas, incluyendo las mediciones de tiempos de transferencia y tamaños de archivos con y sin compresión gzip. Se espera que presenten los resultados de forma clara, acompañados de tablas y/o gráficos comparativos. También deben incluir una breve interpretación de los resultados obtenidos.

Seguridad:

Desarrollar los temas solicitados en la consigna.

Conclusión:
Resumen final del trabajo, destacando los aprendizajes obtenidos, los principales desafíos técnicos enfrentados y posibles mejoras futuras para el servidor. Se espera que el grupo explique qué se buscó aprender o demostrar con el desarrollo del servidor, y cómo se relaciona con los temas vistos en clase.

Se alienta a incluir también cualquier punto adicional de experimentación, análisis o implementación que consideren pertinente, incluso si no fue requerido explícitamente.

Modalidad de entrega y evaluación

El trabajo debe realizarse en grupos de 3 alumnos. Deberán entregar un solo archivo **`.zip`** con sus apellidos como nombre del mismo que contenga:

- El código fuente (**`.py`**) debidamente comentado.
- El informe en formato PDF.
- Una carpeta de ejemplo (archivos_servidor/) con un archivo de prueba.

La entrega debe realizarse a través del campus virtual, con fecha límite el …. de noviembre de 2025 a las 23:59 hs.

No se aceptarán trabajos entregados fuera de término.

El trabajo será evaluado de la siguiente manera: **Código x 0,4 + Informe x 0,6**. Donde el código debe correr sin problemas y realizar lo pedido, y el informe debe mostrar lo trabajado, con cualquier problema que pueda haber surgido, y cumplir con la experimentación y análisis de seguridad.

Documentación

Les dejamos una guía de documentos donde podrán conseguir información para realizar las implementaciones del servidor y las conexiones con los dispositivos, no es necesario usarlas o hacerlo exactamente como dicen estos documentos, es simplemente a modo de ayuda para que tengan fuentes donde informarse.

Socket Programming in Python:
https://realpython.com/python-sockets/
https://www.geeksforgeeks.org/python/socket-programming-python/
https://docs.python.org/3/library/socket.html#socket-objects

Map files to MIME types:
https://docs.python.org/3/library/mimetypes.html

Break URL strings up in components:
https://docs.python.org/3/library/urllib.parse.html

POST request method:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods/POST#example

FormData Objects:
https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest_API/Using_FormData_Objects

---

QRCODE:
https://pypi.org/project/qrcode/
https://www.geeksforgeeks.org/python/generate-qr-code-using-qrcode-in-python/

GZIP:
https://docs.python.org/3/library/gzip.html