# Introducción a los Sistemas Distribuidos (75.43)
# TP N°2: File Transfer

Esteban Carisimo y Juan Ignacio Lopez Pecora

Facultad de Ingeniería, Universidad de Buenos Aires

15 de mayo de 2022

**Resumen**

El presente trabajo práctico tiene como objetivo la creación de una aplicación de red. Para tal finalidad, será necesario comprender cómo se comunican los procesos a través de la red, y cuál es el modelo de servicio que la capa de transporte le ofrece a la capa de aplicación. Además, para poder lograr el objetivo planteado, se aprenderá el uso de la interfaz de sockets y los principios básicos de la transferencia de datos confiable (del inglés Reliable Data Transfer, RDT).

***Palabras clave***— Socket, protocolo, tcp, udp

## 1. Propuesta de trabajo

Este trabajo práctico se plantea como objetivo la comprensión y la puesta en práctica de los conceptos y herramientas necesarias para la implementación de un protocolo RDT. Para lograr este objetivo, se deberá desarrollar una aplicación de arquitectura cliente-servidor que implemente la funcionalidad de transferencia de archivos mediante las siguientes operaciones:

- **UPLOAD**: Transferencia de un archivo del cliente hacia el servidor
- **DOWNLOAD**: Transferencia de un archivo del servidor hacia el cliente

Dada las diferentes operaciones que pueden realizarse entre el cliente y el servidor, se requiere del diseño e implementación de un protocolo de aplicación básico que especifique los mensajes intercambiados entre los distintos procesos.

## 2. Herramientas a utilizar y procedimientos

La implementación de las aplicaciones solicitadas deben cumplir los siguientes requisitos:

- La aplicaciones deben ser desarrolladas en lenguaje Python [1] utilizando la librería estandard de sockets [2].
- La comunicación entre los procesos se debe implementar utilizando UDP como protocolo de capa de transporte.
- Las aplicaciones cliente/servidor pueden ser desplegadas en *localhost*.
- Para lograr una transferencia confiable al utilizar el protocolo UDP, se pide implementar una versión utilizando el protocolo *Stop & Wait* y otra versión utilizando el protocolo *Selective Repeat*.
- El servidor debe ser capaz de procesar de manera concurrente la transferencia de archivos con múltiples clientes.

Para poder validar que el protocolo desarrollado provee garantía de entrega es necesario forzar la pérdida de paquetes. Para poder simular distintas condiciones de red se pide utilizar la herramienta `comcast`[3]. En el repositorio de `comcast` se encuentran las instrucciones de instalación. La herramienta esta escrita en Go, por lo que se requiere instalar Go en primer lugar.

A modo de ejemplo, en el listado 1 se muestra la simulación de una tasa de perdida de paquetes del 10 %:

```
$ comcast --device=lo0 --packet-loss=10%
```

Listing 1: Simulación de pérdida de paquetes mediante comcast

Una vez que terminada la simulación, se debe detener `comcast` para desactivar las reglas setteadas. El listado 2 muestra la finalización de la simulación:

```
comcast --stop
```

Listing 2: Fin de la simulación de pérdida de paquetes

## 2.1. Interfaz del cliente

La funcionalidad del cliente se divide en dos aplicaciónes de línea de comandos: `upload` y `download`. El comando `upload` envía un archivo al servidor para ser guardado con el nombre asignado. El listado 3 especifica la interfaz de linea de comandos para la operación de upload:

```
> python upload -h
usage: upload [-h] [-v | -q] [-H ADDR] [-p PORT] [-s FILEPATH] [-n FILENAME]

<command description>

optional arguments:
  -h, --help     show this help message and exit
  -v, --verbose  increase output verbosity
  -q, --quiet    decrease output verbosity
  -H, --host     server IP address
  -p, --port     server port
  -s, --src      source file path
  -n, --name     file name
```

Listing 3: Interfaz de linea de comandos de upload

El comando `download` descarga un archivo especificado desde el servidor. El listado 4 especifica la interfaz de linea de comandos para la operación de download:

```
> python download -h
usage: download [-h] [-v | -q] [-H ADDR] [-p PORT] [-d FILEPATH] [-n FILENAME]

<command description>

optional arguments:
  -h, --help     show this help message and exit
  -v, --verbose  increase output verbosity
  -q, --quiet    decrease output verbosity
  -H, --host     server IP address
  -p, --port     server port
  -d, --dst      destination file path
  -n, --name     file name
```

Listing 4: Interfaz de linea de comandos de download

### 2.2. Interfaz del servidor

El servidor provee el servicio de almacenamiento y descarga de archivos. El listado 5 especifica la interfaz de linea de comandos para el inicio del servidor:

```
> python start-server -h
usage: start-server [-h] [-v | -q] [-H ADDR] [-p PORT] [-s DIRPATH]

<command description>

optional arguments:
  -h, --help     show this help message and exit
  -v, --verbose  increase output verbosity
  -q, --quiet    decrease output verbosity
  -H, --host     service IP address
  -p, --port     service port
  -s, --storage  storage dir path
```

Listing 5: Interfaz de linea de comandos del servidor

## 3. Análisis

Comparar la performance de la versión Selective Repeat del protocolo y la versión Stop&Wait utilizando archivos de distintos tamaños y bajo distintas configuraciones de pérdida de paquetes.

## 4. Preguntas a responder

1. Describa la arquitectura Cliente-Servidor.
2. ¿Cuál es la función de un protocolo de capa de aplicación?
3. Detalle el protocolo de aplicación desarrollado en este trabajo.
4. La capa de transporte del stack TCP/IP ofrece dos protocolos: TCP y UDP. ¿Qué servicios proveen dichos protocolos? ¿Cuáles son sus características? ¿Cuando es apropiado utilizar cada uno?

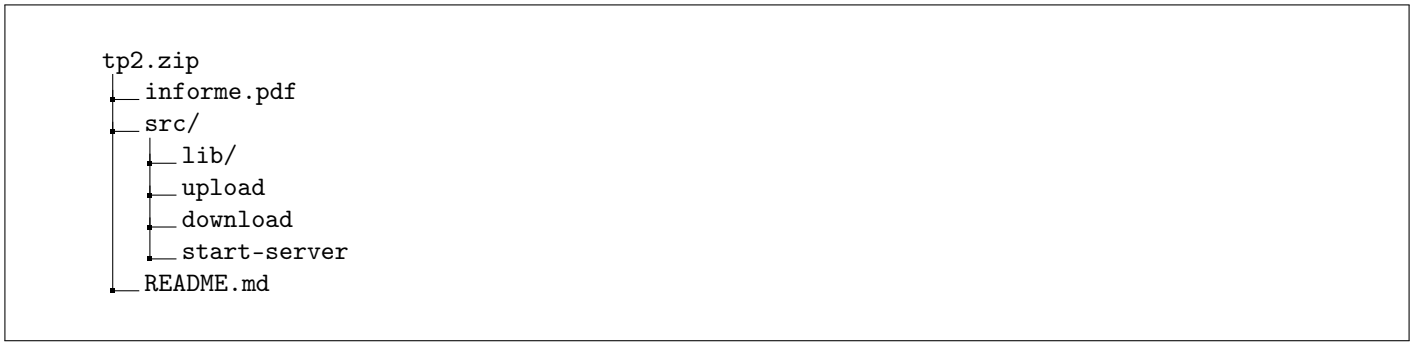


Figura 1: Estructura del archivo zip entregable

# 5. Entrega

La entrega consta de un informe, el código fuente de la aplicación desarrollada y un archivo README en formato MarkDown[4] con los detalles necesarios para ejecutar la aplicación. La codificación debe cumplir el standard PEP8 [5], para ello se sugiere utilizar el linter flake8 [6]. La figure 1 muestra la estructura y el contenido del archivo ZIP entregable.

## 5.1. Fecha de entrega

La entrega se hará a través del campus. La fecha de entrega está pautada para el día viernes 03 de Junio de 2022 a las 19.00hs. **Cualquier entrega fuera de término no será considerada.**

## 5.2. Informe

La entrega debe contar con un informe donde se demuestre conocimiento de la interfaz de sockets, así como también los resultados de las ejecuciones de prueba (capturas de ejecución de cliente y logs del servidor). El informe debe describir la arquitectura de la aplicación. En particular, se pide detallar el protocolo de red implementado para cada una de las operaciones requeridas.

El informe debe tener la siguientes secciones:

- Introducción
- Hipótesis y suposiciones realizadas
- Implementación
- Pruebas
- Preguntas a responder
- Dificultades encontradas
- Conclusión

# Referencias

[1] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[2] Python Software Foundation. *socket — Low-level networking interface*, 2020. `https://docs.python.org/3/library/socket.html` [Accessed: 15/10/2020].

[3] Tyler Treat. *Comcast*, 2020. `https://github.com/tylertreat/comcast` [Accessed: 23/03/2020].

[4] John Gruber. *Markdown*, 2020. `https://daringfireball.net/projects/markdown/` [Accessed: 15/10/2020].

[5] Guido van Rossum, Barry Warsaw, and Nick Coghlan. Style guide for Python code. PEP 8, Python Software Foundation, 2001.

[6] Ian Stapleton Cordasco. *Flake8: Your Tool For Style Guide Enforcement*, 2020. `https://flake8.pycqa.org/en/latest/` [Accessed: 15/10/2020].