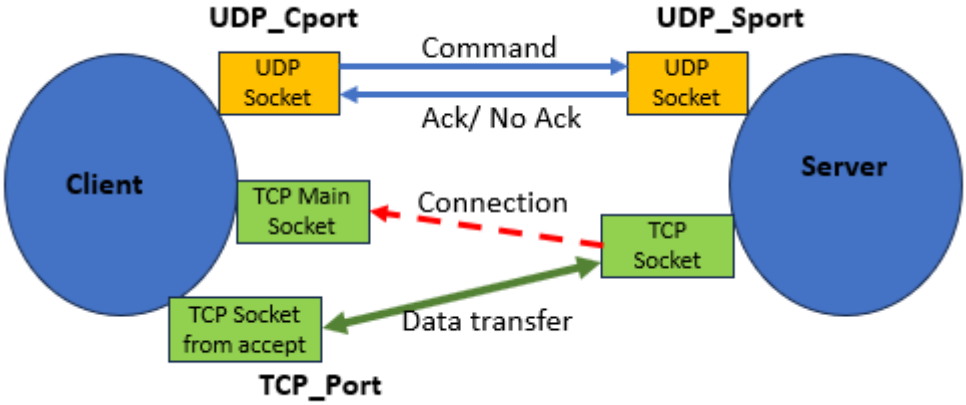
NOVA SCHOOL OF
SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

zFTP Transfer File Service

(TPC1 – Redes de Computadores)

1. Introduction

The objective of this TPC is to implement the client and server programs of a File Transfer Service. In short, the client sends commands to the server using UDP datagrams, while file data is transferred using TCP sockets. The server is sequential, handling only one client at a given time.



A client/server interaction is composed of the following sequence of commands entered at the client. Commands are described below.

- A successful *open* command, where the client informs the server of the TCP port used for subsequent transfers.
- Zero or more *get* or *put* commands that transfer files between the client and the server.
- One successful *close* command, that the client does before finishing and that allows the server to wait for a new client.

The server is launched before the client using the following command:

python zftp-server UDP_SPort

where *UDP_Sport* is the UDP port used by the server to receive commands.

Clients are invoked by a command line such as:

python zftp-client server_name UDP_Sport

where server_name is the name of the server and *UDP_Sport* is the server port.

2. Client commands

The client commands are:

- **open** – this command starts an interaction session, sending to the server the TCP port number where the client will be waiting for connections.
  Syntax: *open <port>*
  Errors: (1) invalid number of arguments;
  (2) invalid port number;
  Example: *open 55555*

- **close** – this command ends an interaction session.
  Syntax: *close*
  Example: *close*

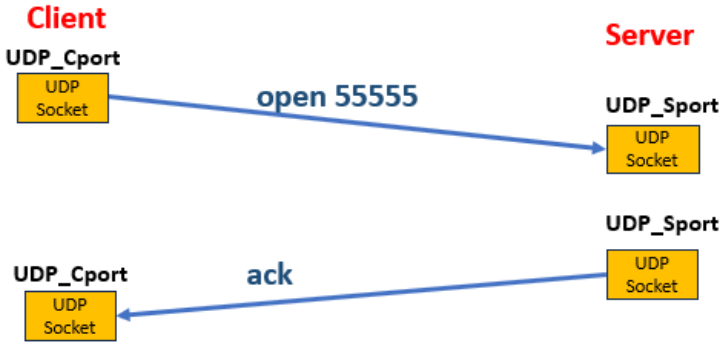
- **get** – this command downloads a file from the server, this means to transfer a file from the server file system to the client´s file system.
  Syntax: *get <remote_filename> <local_filename>*
  Errors: (1) invalid number of arguments;
  (2) a file with the indicated name already exists on the client;
  (3) the indicated file does not exist on the server;
  Example: *get test1.pdf my_test1.pdf*

- **put** – this command uploads a file to the server, this means transfer a file in the client file system to the server´s file system.
  Syntax: *put <local_filename> <remote_filename>*
  Errors: (1) invalid number of arguments;
  (2) the indicated file does not exist on the client;
  (3) a file with the indicated name already exists on the server;
  Example: *put my_test1.pdf test1_server.pdf*

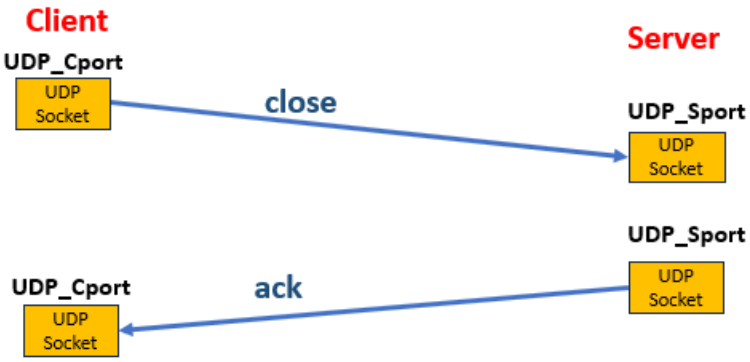
3. Client Server Interaction

In the following a description of the client/server interaction is given for each command. Only successful interactions are presented in the pictures.

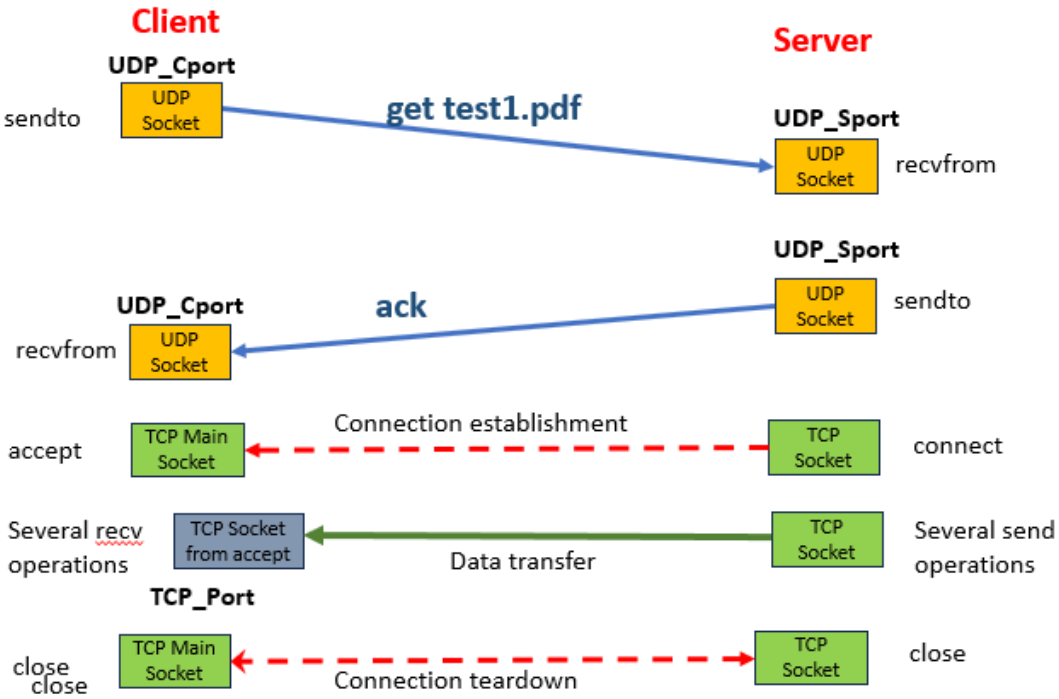
- **open** – The client sends the string open followed by a space and a 5-digit port number. The server replies with the byte array corresponding to the strings *ack* or *nack*. Example:



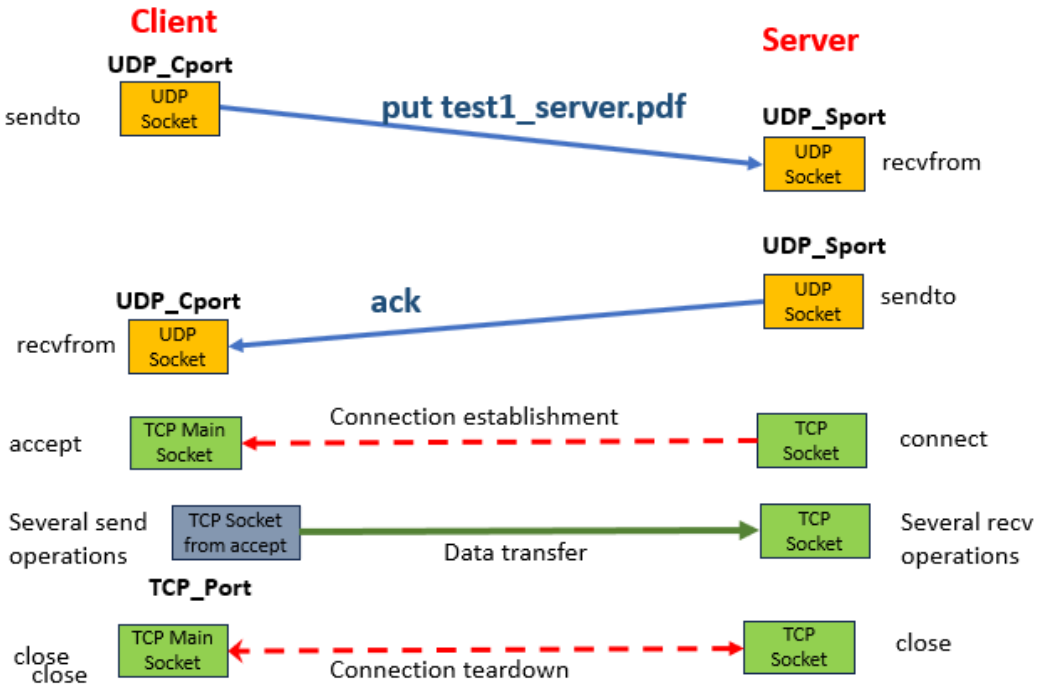
- **close** – The client sends the string *close*. The server replies with the byte array corresponding to the strings *ack* or *nack*. Example:



- **get** – The client sends the string *get* followed by a space and the name of the file in the server's file system. The server replies *nack* if the named file does not exist or *ack* in the opposite case. If the file exists, the server creates a TCP socket, connects it to the TCP_Port of the client, and sends the file contents. After sending all the bytes of the file, closes (and destroys) the socket. The client receives the bytes and writes them in its local system; the closing of the socket by the server indicates the end of the file. Example:



- **put** – The client sends the string *put* followed by a space and the name of the file in the server's file system. The server replies *nack* if the named file already exists or *ack* in the opposite case. In the case of success, the server creates a TCP socket, connects it to the TCP_Port of the client and receives the file contents. After sending all the bytes of the file, the client closes (and destroys) the socket. The server receives the bytes and writes them to its local system; the closing of the socket by the client indicates the end of the file. Example:



Note: the *nack* reply is a message with the format *nack error_number*. For example, when trying to get a file that does not exist on the server it will send the message "*nack 3*", where 3 is the error number.

4. Delivery

The delivery will use a Google form. Details will be sent later. Should be done before 10:00 on 3rd October, 2023.