# RPC File Transfer – Practical Work 2

Group 6 – Distributed Systems

December 6, 2025

## 1 Introduction

This practical work extends the TCP file transfer system from Practical Work 1 to a remote procedure call (RPC) based solution. Instead of sending raw bytes over a custom TCP protocol, the client now invokes RPC methods on a server to upload a file. We use Python's `xmlrpc.server` and `xmlrpc.client` libraries to implement the RPC service.

## 2 RPC Service Design

### 2.1 Service Interface

The RPC server exposes a simple interface:

- `upload_file (name, data)`: receives a file name (string) and file contents (binary) and stores the file on the server.

Files are sent using `xmlrpc.client.Binary` so that arbitrary binary data (text or binary files) can be transferred.

### 2.2 RPC Call Flow

Figure 1 shows the RPC call sequence from client to server.

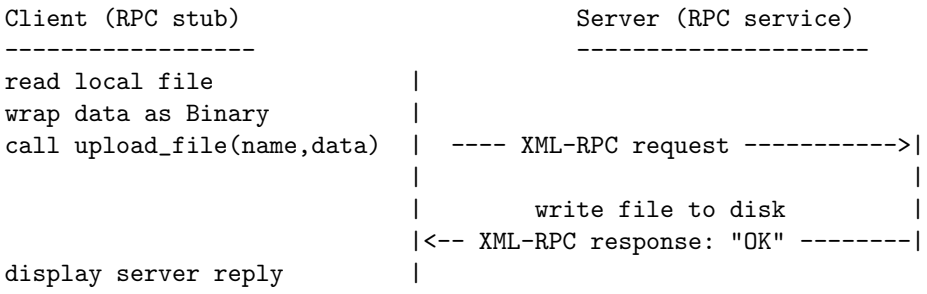


Figure 1: RPC request–response flow for file upload.

# 3 System Organization

The system is organized into an RPC client and an RPC server, communicating over HTTP using XML-RPC. The underlying transport is TCP, but the details are handled by the RPC library instead of our own protocol.
Figure 2 shows the architecture.

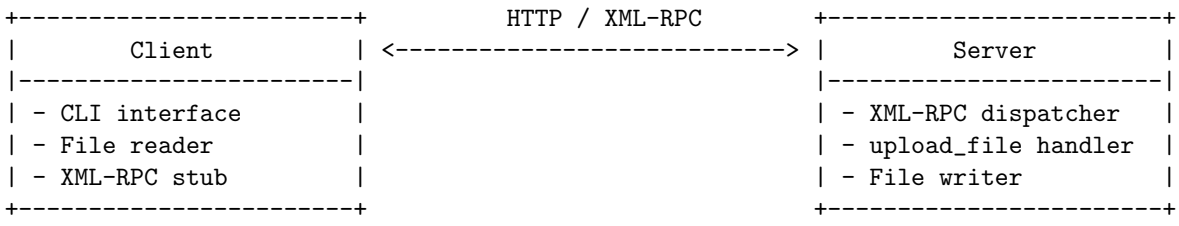


Figure 2: System organization of the RPC file transfer system.

# 4 Implementation

## 4.1 Server Code Snippet

The RPC server registers one method, `upload_file`, which writes the received data to a file on disk.

```
from xmlrpc.server import SimpleXMLRPCServer
from xmlrpc.client import Binary

HOST = "0.0.0.0"
PORT = 8000

def upload_file(name, data):
    # data is of type xmlrpc.client.Binary
    with open(name, "wb") as f:
        f.write(data.data)
    print(f"Received file: {name}")
    return "OK"

server = SimpleXMLRPCServer((HOST, PORT), allow_none=True)
server.register_function(upload_file, "upload_file")

print(f"RPC server listening on {HOST}:{PORT}")
server.serve_forever()
```

## 4.2 Client Code Snippet

The client reads a local file, wraps its contents in `Binary`, and calls the remote `upload_file` method.

```
import os
from xmlrpc.client import ServerProxy, Binary

SERVER_URL = "http://127.0.0.1:8000/"
FILENAME = "test.txt"

filesize = os.path.getsize(FILENAME)
print(f"Sending {FILENAME} ({filesize} bytes)")

with open(FILENAME, "rb") as f:
    data = f.read()

proxy = ServerProxy(SERVER_URL, allow_none=True)
result = proxy.upload_file(FILENAME, Binary(data))

print("Server reply:", result)
```

This implementation reuses the general idea from the TCP solution (one client, one server, file stored on the server side) but delegates framing, encoding and transport details to the RPC library.

# 5 Conclusion

I upgraded our original TCP file transfer system to an RPC-based design using Python's XML-RPC facilities. The new solution simplifies protocol handling: the client only needs to call a remote method, while the library takes care of serialization, message framing and transport over TCP. The system correctly transfers both text and binary files and illustrates how RPC can be used to build higher-level distributed services on top of sockets.