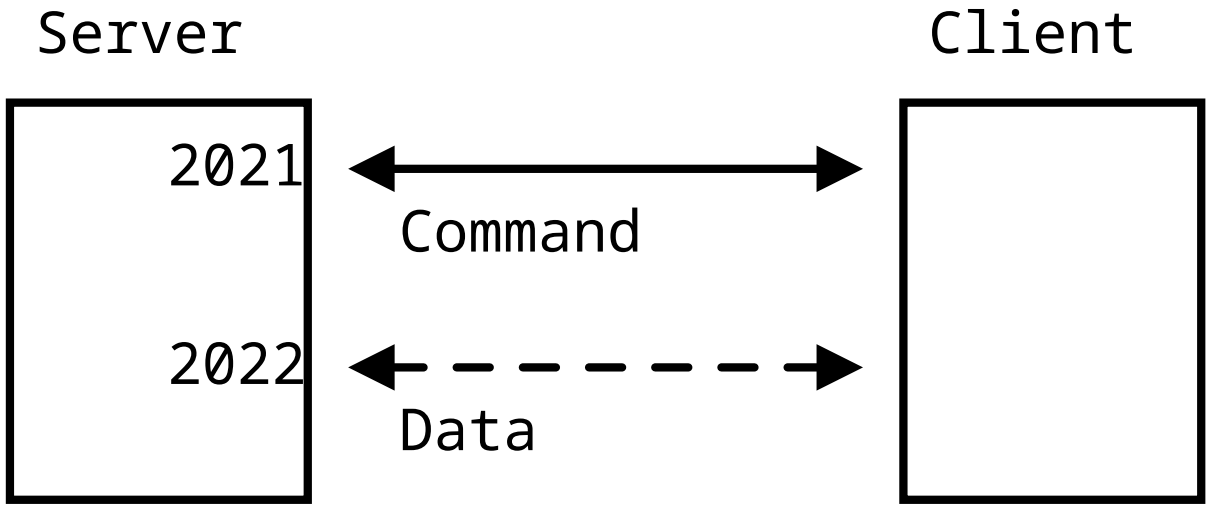
Protocolo FTP

Matías Roqueta

Ingeniería en Telecomunicaciones, Instituto Balseiro

Objetivo

Diseñar arquitectura Cliente $\leftrightarrow$ Servidor



- Puerto 2021: Command channel, permanece abierto mientras la sesión esté activa
- Puerto 2022: Data channel, abierto únicamente en respuesta a comandos que lo requieran

Cliente → Servidor

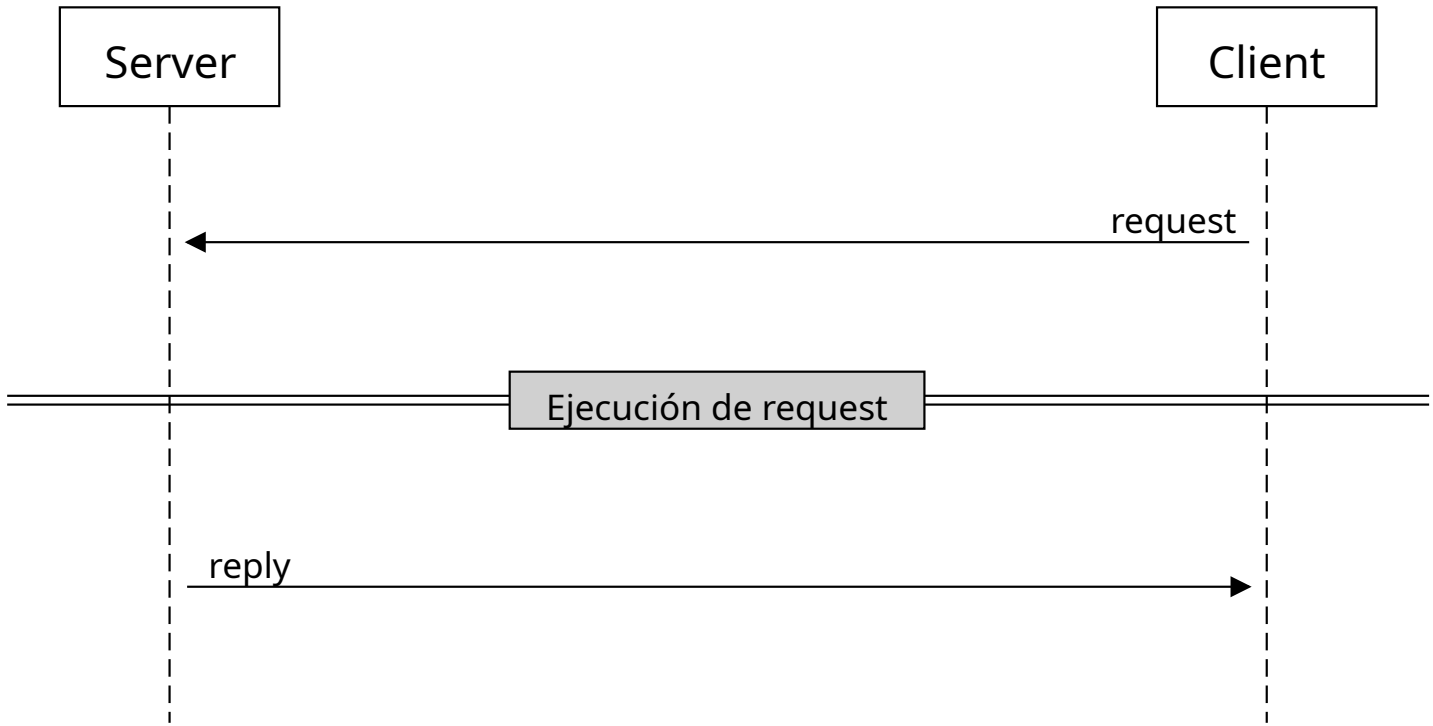
Comandos que usan solamente Command channel:

- `USER <user>`
- `PASS <pass>`
- `CWD <dir>`
- `CDUP`
- `MKD <dir>`
- `RMD <dir>`
- `DELE <file>`
- `QUIT`

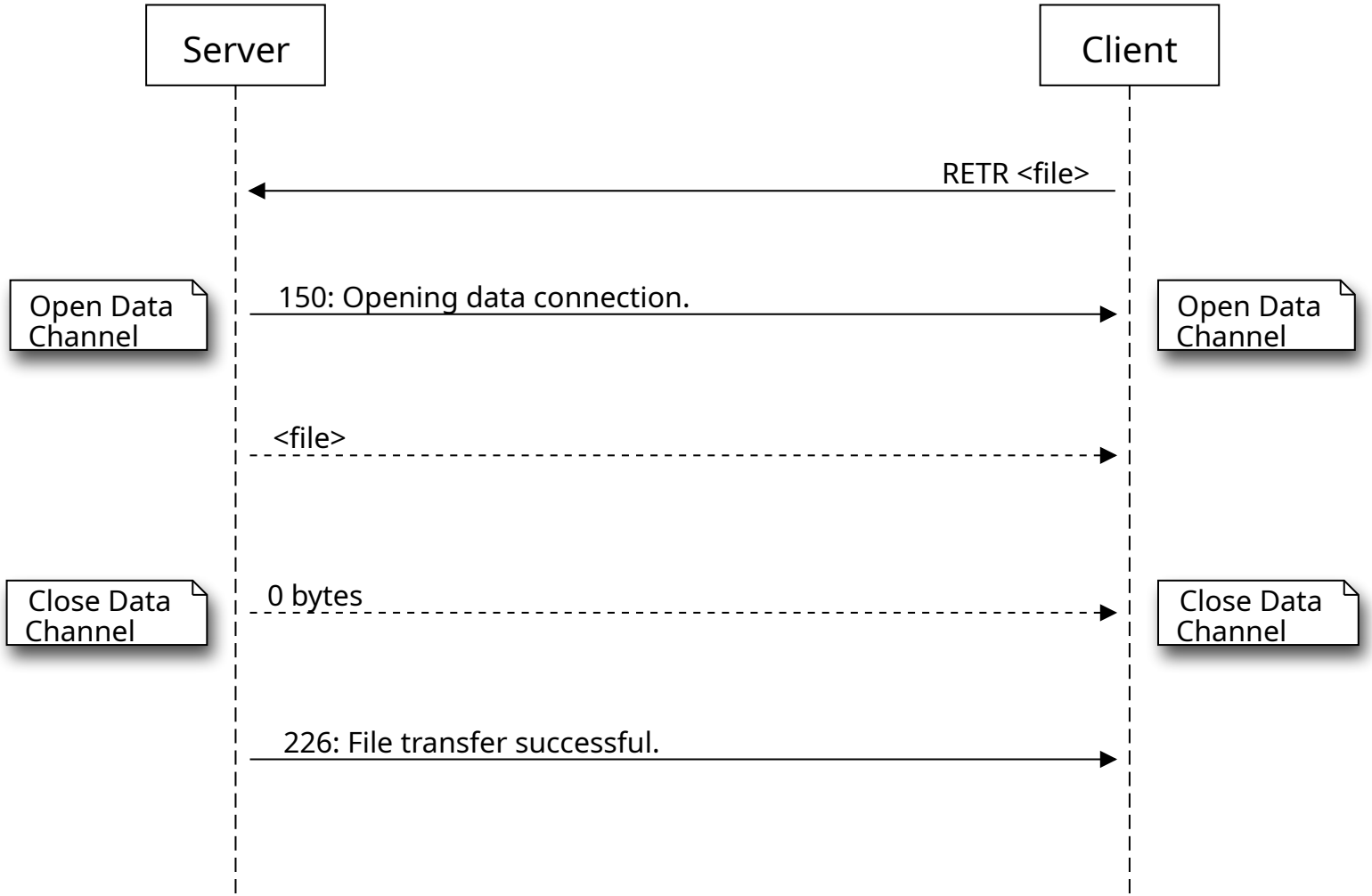
Comandos que usan ambos Command y Data channel:

- `LIST`
- `RETR <file>`
- `STOR <file>`

Loop general



Comando RETR <file>



Server side

```
void handleRETRCommand(int controlClientSocket, const std::string& args) {
    sendResponse(controlClientSocket, "150 Opening data connection.\r\n");
    int dataSocket = createSocket(dataPort);
    int dataClientSocket = establishDataConnection(controlClientSocket,
                           dataSocket, dataAddress, dataPort);
    if (sendFile(dataClientSocket, filename)) {
        std::string response = "226 File transfer successful.";
    } else {
        std::string response = "451 File transfer failed.";
    }
    closeSocket(dataClientSocket);
    closeSocket(dataSocket);
    sendResponse(controlClientSocket, response);
    return;
}
```

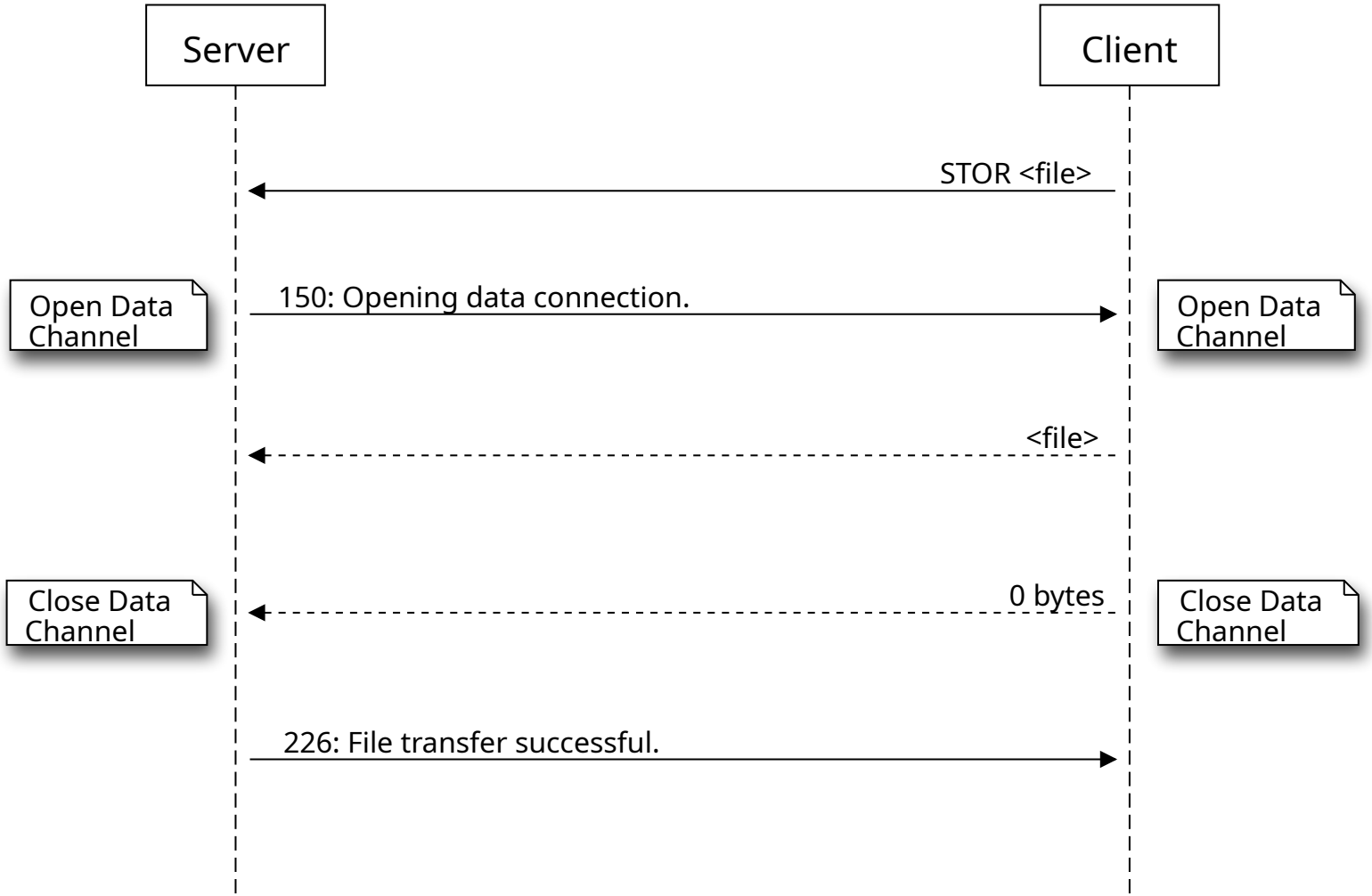
Server side

```cpp
bool sendFile(int socket, const std::string& filename) {
    std::ifstream file(filename, std::ios::binary);
    if (!file) {
        std::cerr << "Failed to open file: " << filename << std::endl;
        return false;
    }
    std::vector<char> buffer(std::istreambuf_iterator<char>(file), {});
    file.close();
    ssize_t bytesSent = send(socket, buffer.data(), buffer.size(), 0);
    if (bytesSent == -1) {
        std::cerr << "Failed to send file contents." << std::endl;
        return false;
    }
    return true;
}
```

Client side

```
void receiveFile(int controlSocket, const std::string& filename) {
    std::string response;
    receiveResponse(controlSocket, response);
    if (response.substr(0, 3) != "150") {
        return;
    }
    int dataSocket = establishDataConnection(controlSocket);
    const int bufferSize = 1024;
    std::vector<char> buffer(bufferSize);
    ssize_t bytesRead;
    while (bytesRead = recv(dataSocket, buffer.data(), bufferSize, 0) > 0) {
        file.write(buffer.data(), bytesRead);
    }
    file.close();
    receiveResponse(controlSocket, response);
    closeSocket(dataSocket);
    return;
}
```

Comando STOR <file>



Server side

```
void handleSTORCommand(int controlClientSocket, const std::string& args) {
    sendResponse(controlClientSocket, "150 Opening data connection.\r\n");
    int dataSocket = createSocket(dataPort);
    int dataClientSocket = establishDataConnection(controlClientSocket,
                           dataSocket, dataAddress, dataPort);
    if (recvFile(dataClientSocket, filename)) {
      response = "226 File transfer successful.";
    } else {
      response = "451 File transfer failed.";
    }
    closeSocket(dataClientSocket);
    closeSocket(dataSocket);
    sendResponse(controlClientSocket, response);
    return;
}
```

Server side

```cpp
bool recvFile(int dataSocket, const std::string& filename) {
    const int bufferSize = 1024;
    std::vector<char> buffer(bufferSize);
    std::ofstream file(filename, std::ios::binary);
    ssize_t bytesRead;
    while (bytesRead = recv(dataSocket, buffer.data(), bufferSize, 0) > 0) {
        file.write(buffer.data(), bytesRead);
        std::cout << "Read " << bytesRead << " bytes" << std::endl;
    }
    if (bytesRead < 0) {
        file.close();
        return false;
    }
    file.close();
    return true;
}
```

Client Side

```
void sendFile(int controlSocket, const std::string& filename) {
    std::string response;
    receiveResponse(controlSocket, response);
    if (response.substr(0, 3) != "150") {
        return;
    }
    int dataSocket = establishDataConnection(controlSocket);
    std::ifstream file(filename, std::ios::binary);
    std::vector<char> buffer(std::istreambuf_iterator<char>(file), {});
    file.close();
    ssize_t bytesSent = send(dataSocket, buffer.data(), buffer.size(), 0);
    closeSocket(dataSocket);
    receiveResponse(controlSocket, response);
    return;
}
```

Loop Básico

```
while(true) {
    if (authenticated) {
        // Selección de comandos RETR, STOR, etc..
    } else {
        // Selección de comandos USER, PASS, o rechazo
    }
}
```