

Practical Work 2: RPC File Transfer

Group ID: 18

1 RPC Service Design

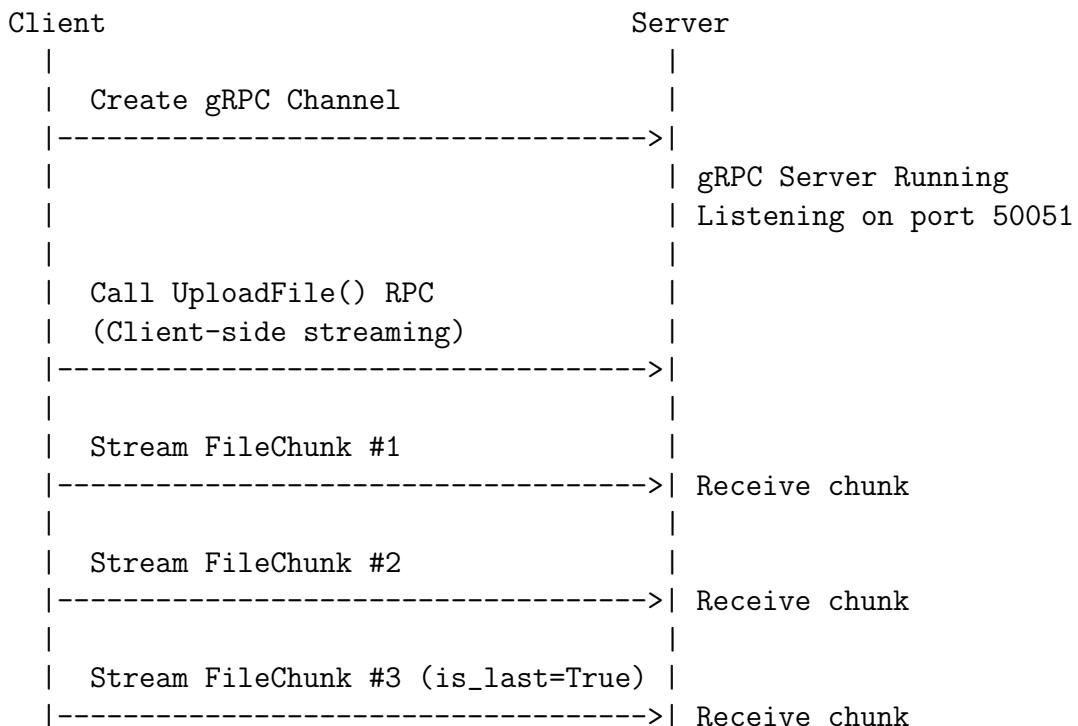
This file transfer system uses **gRPC** (Google Remote Procedure Call) framework with Protocol Buffers for efficient communication.

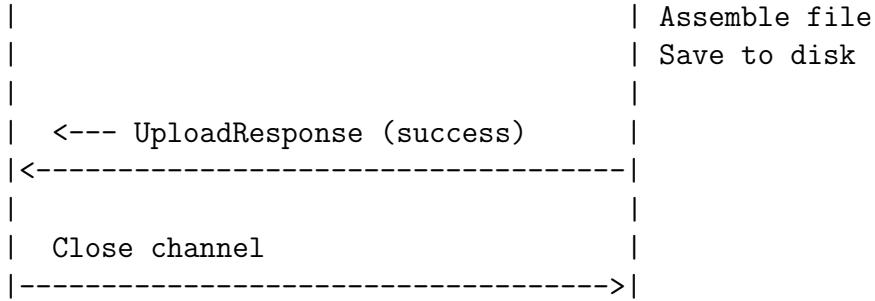
1.1 Architecture Overview

The RPC architecture consists of:

- **Protocol Buffer Definition** (.proto file): Defines the service interface and message types
- **gRPC Server**: Implements the RPC methods and handles file reception
- **gRPC Client**: Calls RPC methods to upload files
- **Generated Code**: Auto-generated from .proto file for serialization/deserialization

1.2 RPC Communication Flow





1.3 Protocol Buffer Messages

The system uses the following message types:

```

1 message FileChunk {
2     string filename = 1;           // File name
3     bytes content = 2;           // Chunk data
4     bool is_last = 3;            // Last chunk flag
5 }
6
7 message UploadResponse {
8     bool success = 1;            // Success status
9     string message = 2;          // Response message
10 }
11
12 service FileTransferService {
13     rpc UploadFile(stream FileChunk)
14         returns (UploadResponse);
15 }
```

Listing 1: Protocol Buffer Definition

2 System Organization

2.1 Directory Structure

```

practical2/RPC/
|-- proto/
|   |-- file_transfer.proto      # Protocol Buffer definition
|-- generated/
|   |-- __init__.py
|   |-- file_transfer_pb2.py    # Generated messages
|   |-- file_transfer_pb2_grpc.py # Generated service
|-- server.py                  # gRPC server
|-- client.py                  # gRPC client
|-- test_file.txt              # Test file
|-- requirements.txt           # Dependencies
|-- 02.rpc.file.transfer.tex   # This report
```

2.2 Component Interaction

1. **.proto file**: Defines the contract between client and server
2. **protoc compiler**: Generates Python code from .proto file
3. **Generated code**: Provides message classes and service stubs
4. **Server**: Implements FileTransferServicer class
5. **Client**: Uses FileTransferServiceStub to call RPC methods

2.3 Data Flow

[Client File]

```
--> Read in chunks (4096 bytes)
--> Create FileChunk messages
--> Stream via gRPC channel
--> [Server receives stream]
--> Assemble chunks
--> Write to disk
--> [Received File]
```

3 Implementation

3.1 Server Implementation

The server implements the FileTransferServicer class:

```
1  class FileTransferServicer(
2      file_transfer_pb2_grpc.FileTransferServiceServicer):
3
4      def UploadFile(self, request_iterator, context):
5          """
6              RPC method to receive file from client
7              request_iterator: stream of FileChunk from client
8          """
9          filename = None
10         received_data = bytearray()
11
12         # Receive chunks from client stream
13         for chunk in request_iterator:
14             if filename is None:
15                 filename = chunk.filename
16
17             # Append chunk data
18             received_data.extend(chunk.content)
19
20             if chunk.is_last:
21                 break
22
23         # Save file to disk
```

```

24     output_filename = f"received_{filename}"
25     with open(output_filename, 'wb') as f:
26         f.write(received_data)
27
28     # Return response
29     return file_transfer_pb2.UploadResponse(
30         success=True,
31         message=f"File uploaded successfully"
32     )

```

Listing 2: Server - RPC Method Implementation

```

1 def serve():
2     # Create gRPC server with thread pool
3     server = grpc.server(
4         futures.ThreadPoolExecutor(max_workers=10)
5     )
6
7     # Register servicer
8     file_transfer_pb2_grpc.add_FileTransferServiceServicer_to_server(
9         FileTransferServicer(), server
10    )
11
12    # Bind to address and start
13    server.add_insecure_port('127.0.0.1:50051')
14    server.start()
15    server.wait_for_termination()

```

Listing 3: Server - Start gRPC Server

3.2 Client Implementation

The client uses streaming to send file chunks:

```

1 def generate_file_chunks(filename):
2     """
3     Generator function to create stream of FileChunk
4     """
5     file_basename = os.path.basename(filename)
6
7     with open(filename, 'rb') as f:
8         while True:
9             chunk_data = f.read(4096)
10
11            if not chunk_data:
12                # Send last chunk with empty content
13                yield file_transfer_pb2.FileChunk(
14                    filename=file_basename,
15                    content=b'',
16                    is_last=True
17                )

```

```

18         break
19
20     # Send chunk
21     yield file_transfer_pb2.FileChunk(
22         filename=file_basename,
23         content=chunk_data,
24         is_last=False
25     )

```

Listing 4: Client - Generate File Chunks

```

1 def upload_file(filename):
2     # Create gRPC channel
3     with grpc.insecure_channel('127.0.0.1:50051') as channel:
4         # Create stub (client)
5         stub = file_transfer_pb2_grpc.FileTransferServiceStub(
6             channel)
7
8         # Call RPC method with streaming
9         response = stub.UploadFile(generate_file_chunks(filename))
10
11    # Handle response
12    if response.success:
13        print(f"Success: {response.message}")

```

Listing 5: Client - Call RPC Method

3.3 Comparison: TCP vs RPC

Aspect	TCP (Practical 1)	RPC (Practical 2)
Protocol	Raw TCP sockets	gRPC over HTTP/2
Data Format	Raw bytes	Protocol Buffers
Communication	Manual send/recv	RPC method calls
Streaming	Manual chunking	Built-in streaming
Error Handling	Manual	Built-in RPC errors
Code Generation	None	Auto-generated from .proto

4 Who Does What

4.1 Protocol Buffer Compiler (protoc)

- Reads `file_transfer.proto`
- Generates `file_transfer_pb2.py` (message classes)
- Generates `file_transfer_pb2_grpc.py` (service classes)

4.2 Server Responsibilities

- Implements `FileTransferServicer` class

- Overrides `UploadFile()` RPC method
- Receives streaming `FileChunk` messages
- Assembles chunks into complete file
- Saves file to disk
- Returns `UploadResponse` to client

4.3 Client Responsibilities

- Creates gRPC channel to server
- Creates `FileTransferServiceStub`
- Reads file from disk in chunks
- Generates stream of `FileChunk` messages
- Calls `UploadFile()` RPC method
- Handles `UploadResponse` from server

4.4 gRPC Framework Responsibilities

- Manages network connections (HTTP/2)
- Handles serialization/deserialization (Protocol Buffers)
- Manages streaming data flow
- Provides error handling and status codes
- Thread pool management for concurrent requests

5 Testing

5.1 Setup

Install dependencies:

```
pip install grpcio grpcio-tools
```

Generate code from .proto file:

```
python -m grpc_tools.protoc -I./proto
--python_out=./generated
--grpc_python_out=./generated
./proto/file_transfer.proto
```

5.2 Running the System

1. Open first terminal, run server:

```
cd practical2/RPC  
python server.py
```

2. Open second terminal, run client:

```
cd practical2/RPC  
python client.py
```

3. Verify that `received_test_file.txt` is created

4. Compare original and received files to verify correctness

5.3 Expected Output

Server output:

```
[SERVER] gRPC Server started on 127.0.0.1:50051  
[SERVER] Waiting for clients...  
[SERVER] Receiving file: test_file.txt  
[SERVER] Received last chunk  
[SERVER] File saved successfully: received_test_file.txt  
[SERVER] Total bytes received: 407
```

Client output:

```
[CLIENT] Connecting to server 127.0.0.1:50051  
[CLIENT] Preparing to send file: test_file.txt (407 bytes)  
[CLIENT] Sending chunk 1 (407 bytes)  
[CLIENT] Success: File test_file.txt uploaded successfully
```