FileSync

Technical Documentation

Jakub Jagodziński

Version 1.0
July 3, 2025

Contents

1 Introduction

1.1 Project Objective

The objective of the FileSync project is to develop a File Synchronization Service (FSS) that enables reliable and automated replication of a designated archive directory from a client to a server. The system is designed to:

- detect and transmit modified files based on their last modification timestamps,
- support nested directory structures in the archive,
- tolerate disconnections and resume operation upon reconnection,
- operate without prior knowledge of the server's IP address,
- and maintain efficient communication through a custom-defined protocol using JSON messages.

The service consists of a client-side and server-side application, with both sides handling separate roles in initiating, negotiating, executing, and completing synchronization sessions.

1.2 Scope of the Application

The application focuses on unidirectional synchronization from client to server. Each client maintains a local archive directory whose contents (including subdirectories) are compared against the server-side archive. Based on differences in file metadata (name, path, and modification date), only changed or new files are transmitted.

The client locates the server dynamically via multicast-based UDP discovery. If a connection is lost or fails, the client automatically resumes the discovery and synchronization attempt loop. The server handles only one client connection at a time, queuing additional clients and resuming them sequentially.

This project does not include conflict resolution logic, file versioning, or bidirectional sync.

2 System Overview

2.1 Client-Server Architecture

The FileSync system is designed using a client-server architecture, where the server controls access to the synchronization service and the client initiates the synchronization process.

The server listens for client discovery messages via UDP multicast and responds with connection offers containing a TCP port number. Only one client can be served at a time — additional clients are placed into a thread-safe queue and served sequentially.

The server runs multiple threads:

- a UDP multicast listener for responding to `DISCOVER` messages,
- a TCP listener that accepts client connections,
- a session manager that queues and serves clients one at a time,
- a dedicated synchronization handler for each active client session.

On the client side, multiple threads are also used:

- a multicast discoverer for locating the server,
- a TCP manager handling data transmission,
- sender and receiver subthreads for communication and file transfer.

This separation ensures that both components remain responsive and robust against network disruptions.

2.2 Communication Protocol Summary

Communication between client and server is performed in two phases:

1. Discovery Phase (UDP Multicast):

- The client sends a `DISCOVER` message to a predefined multicast group and port.
- The server responds with an `OFFER` message that includes the TCP port number available for synchronization.

2. Synchronization Phase (TCP):

1. The client connects to the provided TCP port.
2. The server replies with a `BUSY` or `READY` message.
3. If ready, the client sends a `CLIENT_FILES_INFO` message with archive metadata.
4. The server compares file lists and replies with a `TASKS` message.
5. The client uploads the necessary files using `FILE_HEADER` and binary content.

6. After transmission, the server sends a `NEXT_SYNC` message with a Unix timestamp for the next sync.

If a connection is lost at any point, the client restarts the discovery process and attempts reconnection automatically.

3 Client Application

3.1 Startup Parameters

Upon launching the client application, the user is prompted to provide the following configuration parameters:

- **Client ID:** A unique identifier for the client (optional). If not provided, the client's IP address is used.
- **[Optional]** Manual server endpoint: the user may provide a specific IP address and TCP port to bypass automatic discovery.

The client's archive path is not provided directly. Instead, it is dynamically constructed by the application using the configured base path:

```
<BASE_REPOSITORY_PATH>/client_{<ClientID> or <ClientIP>}
```

This ensures that each client is assigned a unique folder on the server side without requiring explicit input from the user.

3.2 Multicast Discovery Mechanism

The client application initially does not know the IP address or hostname of the FileSync server. Instead, it periodically broadcasts a `DISCOVER` message over a predefined multicast IP group and port using UDP.

- This operation is handled in a dedicated thread to ensure non-blocking behavior.
- If a valid `OFFER` message is received from a server, the client records the sender's IP address and the TCP port offered.
- If no response is received within 5 seconds, the thread sleeps for 10 seconds before retrying.
- If a TCP connection becomes active, the discovery thread is suspended and resumed only upon disconnection.
- All discovery attempts and responses are logged to the console for transparency.

3.3 TCP Management

Once a valid TCP endpoint is discovered, the client attempts to connect to the server. This process is subject to a 5-second timeout. If unsuccessful, discovery resumes.

Upon establishing a connection:

- The client waits for a control message:
  - If it receives `BUSY`, it remains connected and waits for a future `READY` signal.
  - If it receives `READY`, the synchronization process begins.
- The client sends a `CLIENT_FILES_INFO` message containing file metadata.
- It waits for a `TASKS` message listing required files, then sends those files using `FILE_HEADER` followed by the file content.
- After successful transmission, the client receives a `NEXT_SYNC` message with the scheduled time of the next sync.

If the TCP connection is interrupted at any point, the client gracefully closes all active threads and restarts discovery from the beginning.

4 Server Application

4.1 Startup Configuration

When the FileSync server is launched, it prompts the user to define the following configuration values:

- **Synchronization Interval:** the number of seconds that determines how often each client should initiate synchronization.
- **TCP Listening Port:** the port on which the server listens for client connections.

After initialization, the server starts two background threads:

- a TCP listener thread that handles incoming client connections,
- a UDP multicast listener thread that waits for `DISCOVER` messages and responds with `OFFER` messages containing the TCP port.

These two threads allow the server to dynamically announce its availability and wait for new connections without requiring hardcoded IP addresses.

4.2 Session Queue

The FileSync server is designed to handle only one client synchronization session at a time. If multiple clients attempt to connect simultaneously, the server uses a queue to manage their sessions in a first-come, first-served manner.

- When a new client connects:
  - If no session is active, the server sends a `READY` message and immediately begins synchronization.
  - If another client is already being served, the new client receives a `BUSY` message and is placed in the internal waiting queue.
- Once the active session ends:
  - The server checks the queue.
  - If any clients are waiting, the next one is selected and receives a `READY` message to begin its session.

The queue ensures fairness and prevents multiple overlapping file transfers. The server is responsible for managing session handover and cleanup if connections are dropped. Clients that are disconnected are expected to reinitiate the discovery and connection process from the beginning.

5 Communication Protocol

5.1 Message Types

This section describes the structure and purpose of all JSON-formatted messages exchanged between the client and the server during discovery and synchronization.

5.1.1 DISCOVER

The `DISCOVER` message is broadcast by the client over UDP using multicast. Its purpose is to locate the available servers on the local network. Upon receiving this message, the server responds with an `OFFER` message containing the TCP port for synchronization.

```
{
  "type": "DISCOVER"
}
```

5.1.2 OFFER

The `OFFER` message is sent by the server in response to a `DISCOVER` message from the client. It informs the client that the server is available and provides the TCP port number on which it accepts synchronization requests.

```
{
  "type": "OFFER",
  "port": 12345
}
```

5.1.3 SERVER_STATUS

The `SERVER_STATUS` message is used by the server to inform the client about its current availability. Typical values for the `status` field include `"READY"` or `"BUSY"`, allowing the client to decide whether to proceed with synchronization or wait.

```
{
  "type": "SERVER_STATUS",
  "status": "READY"
}
```

5.1.4 CLIENT_FILES_INFO

The `CLIENT_FILES_INFO` message is sent by the client to provide the server with a snapshot of its local archive. It includes the client's identifier and a list of files, each represented by metadata such as name, path, and last modification time. The server uses this information to determine which files require synchronization.

```
{
  "type": "CLIENT_FILES_INFO",
  "client_id": "client123",
  "files": [
    {
      "name": "document.txt",
      "path": "data/document.txt",
      "last_mod": "2025-05-06T14:23:00"
    },
    {
      "name": "image.jpg",
      "path": "data/images/image.jpg",
      "last_mod": "2025-05-05T11:00:00"
    }
  ]
}
```

5.1.5 FILE_INFO

The `FILE_INFO` structure is used to describe a single file in the client's archive. It contains metadata such as file name, relative path, and last modification timestamp. This structure is commonly nested within other messages like `CLIENT_FILES_INFO` to represent file collections.

```
{
  "name": "file1.txt",
  "path": "data/file1.txt",
  "last_mod": "2025-04-30T09:45:00"
}
```

5.1.6 TASKS

The `TASKS` message is sent by the server to the client after receiving the client's archive metadata. It specifies which files are needed to be transmitted during synchronization. The `files_to_send` field contains a list of relative file paths selected for transfer based on comparison with the server's archive.

```
{
  "type": "TASKS",
  "files_to_send": [
    "data/file1.txt",
    "data/file2.csv"
  ]
}
```

5.1.7 FILE_HEADER

The `FILE_HEADER` message is sent by the client to the server before transmitting the binary content of a file. It provides essential metadata about the file to be transferred, such as its name, path, size in bytes, and last modification time. This message allows the server to prepare to receive and store the file correctly.

```
{
  "type": "FILE_HEADER",
  "name": "file1.txt",
  "path": "data/file1.txt",
  "size": 2048576,
  "last_mod": "2025-05-06T15:30:00"
}
```

5.1.8 NEXT_SYNC

The `NEXT_SYNC` message is sent by the server after a synchronization session is completed. It informs the client about the scheduled time for the next synchronization attempt. The `time` field contains an ISO 8601 formatted timestamp indicating when the next sync should occur.

```
{
  "type": "NEXT_SYNC",
  "time": "2025-05-07T16:30:00"
}
```