Protocol Design

1. \*\*What kinds of messages will be exchanged across the control channel?\*\*

- Commands from the client to the server, such as "put", "get", "ls", and "exit".

- Responses from the server to the client, including status messages (e.g., "OK", "File not found", or "Command not recognized"), and data like ephemeral port numbers for data transfer.

2. \*\*How should the other side respond to the messages?\*\*

- The server should respond to control commands with appropriate actions, such as sending back an ephemeral port number for "put" and "get", or sending a list of files for "ls". For each action, the server should also send a status message indicating the success or failure of the command.

- The client should handle server responses by either initiating a data connection on the provided ephemeral port or by displaying the received information.

3. \*\*What sizes/formats will the messages have?\*\*

- Control messages are typically strings encoded in a certain character set (like UTF-8) and are delimited by a newline or null character to indicate the end of a message.

- Data messages, such as files being transferred, often start with a fixed-size header indicating the length of the data, followed by the actual data bytes.

4. \*\*What message exchanges have to take place in order to set up a file transfer channel?\*\*

- A file transfer is initiated by the client sending a "put" or "get" command. The server responds with an ephemeral port number for the client to connect to. Once the data connection is established on this port, file transfer can proceed.

5. \*\*How will the receiving side know when to start/stop receiving the file?\*\*

- The sender includes the size of the file in the header of the message, enabling the receiver to know exactly how many bytes to expect. The receiver reads from the socket until it has received the number of bytes specified in the header.

6. \*\*How to avoid overflowing TCP buffers?\*\*

- Implement flow control mechanisms, such as reading and writing in chunks (often done in sockets with a buffer size, e.g., 1024 or 4096 bytes) to prevent sending more data than the receiver can process. Additionally, the sender can wait for acknowledgments from the receiver before sending more data, ensuring that the receiver's buffer does not overflow.

These points are key considerations when designing and implementing any network communication protocol, ensuring that the communication is robust, efficient, and error-free.