Jack Zhang(jzhang34), Ivian Zhang(yzhang41)

# CS117: Filecopy Design Document

**Overview**

We propose a filecopy protocol based on the popular file transfer protocol widely used today, User Datagram Protocol (UDP), with optimization on the amount and specificity of information transferred by each packet. There are three main components to this protocol: the client end, server end, and network transmission end. Since the underlying packet transfer protocol for the provided sockets is UDP, we don't expect any corruption in the delivered packets during the transmission process. On the client side, it is responsible for copying all files into the system buffer as small packets and sending them over the network sockets. On the server side, it is responsible for receiving the packet representations for each file and writing them into the local file system in the correct order (see Figure 1).

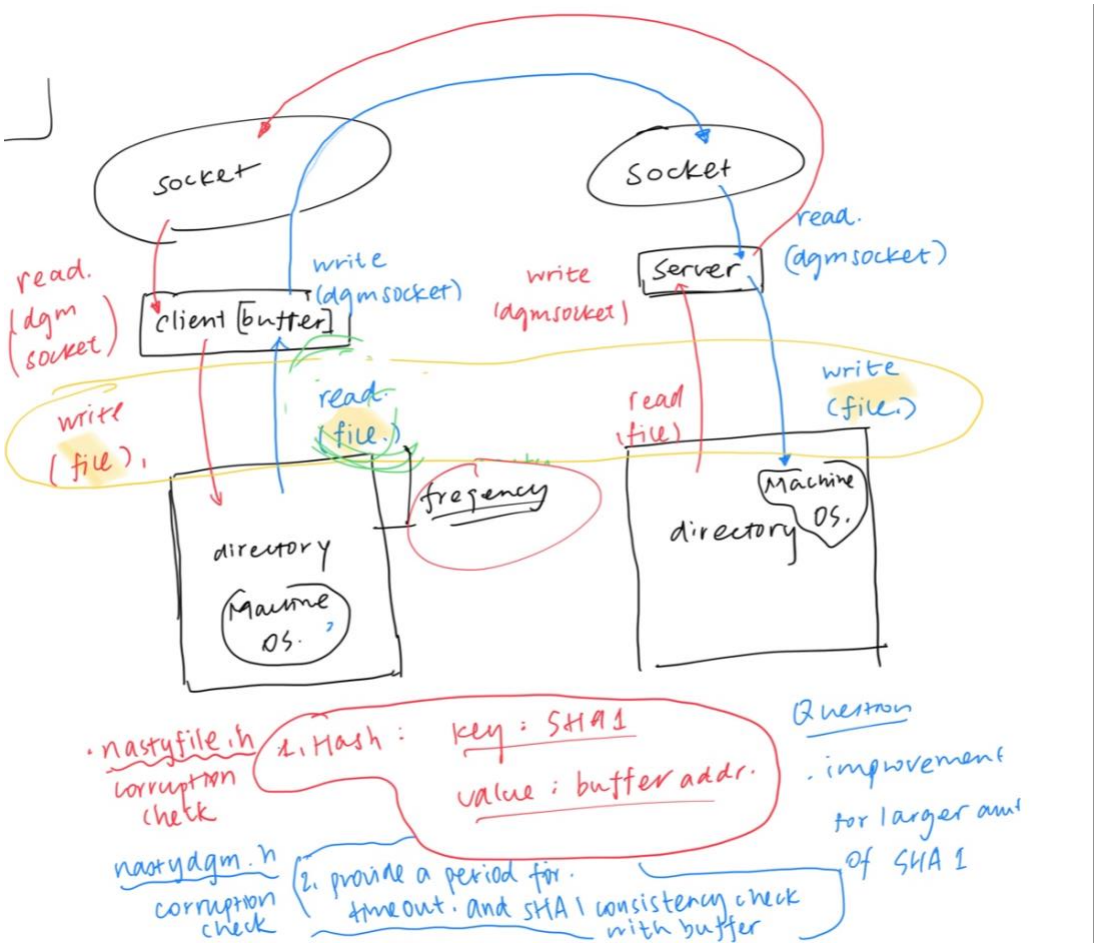


Figure 1. filecopy operation logic demonstration

**Client Logic**

The client is responsible for two steps of the file copy operation: *copying* the file from the local directory into the memory buffer and *sending* the file packets over the network. To counteract any corruption or nastiness, from the reading stage, the client is responsible for breaking a large file into small packets and ensuring correctness in reading for each part. The packet representation for the file also becomes beneficial and efficient during the network transmission stage. Since multiple packets can be dropped with UDP (i.e., in a scenario with an unstable network), the client can quickly resend the missing packets instead of resending the whole file.

A packet representation is a struct of 512 bytes in size and contains the following contents:

```
typedef struct Packet {
    unsigned char content[490];
    size_t seqNum;
    uint32_t packet_status;
    uint32_t totalFileNum;
```

Figure 2. Packet Struct definition

- The first character array is to store the 490 bytes or fewer of the file content.
- seqNum to represent the index of the current packet in the overall file
- packet_status to represent the purpose of the current packet in the file
- the last field represents the total number of files in the current directory.

There is a list of numbers used to represent different packet_status in this object:

- For example, 0 is to indicate a regular content packet.
- 1 is to show the last packet in the current file.
- 2 is to display the current packet with the file name.

The packet_status is further encrypted to represent the index of the current file in the directory: the numeric digits represent the actual status, and the decimals represent the current file index.

The purpose of packing several fields of description with the content is to reduce additional individual packets transmitted. This is a similar practice to the content header in HTTP protocols, providing content descriptions to the message's recipient.

For each chunk, or 490 bytes, of the file read in from the file system, we generate a SHA1 on this partial data. Multiple read on the same section in the file is done. The design is to rely on the low collision rates of nasty reads SHA1. Compared to reading the content repetitively correctly, there is a much smaller chance in a corrupted file reading system for multiple nasty reads on one file to have the same SHA1, given a large number of bytes per file. Hence, we choose the buffer that has generated a SHA1 previously seen in the table to be the correct content buffer.

**Server Logic**

The server is responsible for another two operations of the process: *copying* all packet representations of the provided file into a local buffer and *reconstructing* the file in the correct order to write into the file system. The primary responsibilities of the server are to confirm that all packets representing one file have been received correctly and written into the file system in their correct order. Since each packet of file content comes with its sequence number, the server will use an unordered map, or hash table, to track the packets that have arrived. If the server has not received packets for a period longer than 5 seconds, it will check the current file's completeness. For any file missing, it will send the sequence number back to the client and request another resend. The server will use information from the last packet, with a packet_status of 2, to identify the total number of packets expected for this file.

It is important to note that during the file transmission process, only the server will let the client knows that the current file is entirely received. Upon receiving the complete message, the client will move to send packets for the next file. Therefore, the server will need to parse any incoming message to note if the client has moved on to send packets for the following file by parsing decimal digits of the packet_status.

The server will again rely on the SHA1 for individual file packets to account for unreliable write and reads. Multiple reads back are complete for each packet content written to the file. Similar logic follows as the Readafile in Client: if the read back SHA1 matches the original packet content or is repetitively observed in the hashtable, the server will move on to the following location of the file. Since the read may be corrupted and lead to unmatched SHA1, multiple reads (~20) are carried out until one of the conditions is met, and the server will move on to the next packet. Since either end of the file stream may be unreliable, the server will never know whether the writes or read are successful. Therefore, this is a heuristic approach in the writing of the file.

**File Transmission Logic**

This protocol aims to reduce unnecessary packets delivered over the network. This accounts for the high number of packets dropping on an unstable network. There are two critical design choices for file transmission:

(a) combine metadata about the current file in one filename packet, transmitting at the beginning of each file transmission;

(b) encrypt the packet status field and sequence number fields for each packet read in.

Since the client preprocesses the file and stores all packets in individual packets, the client only has to receive two types of message: (1) completion for an individual file, and (2) resend request for current file missing packets from the server. The client only moves forwards to sending packets for the following file upon receiving (1)completion or (2) have waited for over 50 seconds in waiting for the file. It is then the server's job to redistribute the packets into the corresponding file by recognizing the correct sequence number. More details were elaborated on in the client logic section. The metadata with the filename packet provides information on the total number of packets represented for the current file.