

P2P File Sharing Application System Functional Specification

CMP2204 Term Project Spring 2021

1 System Definition

P2P File Sharing application consists of two major components: (i) discovering all available users in the network and the contents they carry, and (ii) downloading a content (all chunks of this content) from -different users in the network. Figure 1 demonstrates the four processes that together make up the P2P file sharing application; the left two processes correspond to the discovery of content, and the right two processes correspond to downloading content between pairs.

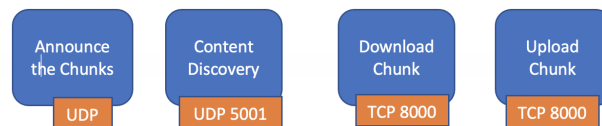


Figure 1: The four processes of the peer-to-peer file exchange application

1.1 Operational Scenarios

The following are the use cases supported by the P2P File Sharing Application:

Chunk Announcement - Chunk Discovery: Upon connecting to the Local Area Network, every peer starts to periodically broadcast the list of all files they have (Figure 2).

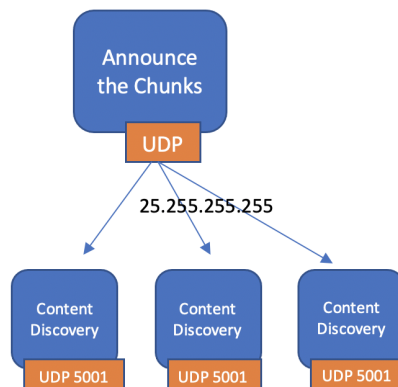


Figure 2: The Chunk Announcer broadcasts its chunks to all nodes in the network.

Upon connecting to the Local Area Network, every node also starts listening for peers announcements in the LAN. Upon hearing an announcement, the contents are stored in a local dictionary of “which users have which files”.

Downloading a file: The end user specifies a content to download, and the downloading process looks up its local dictionary to see which users have the chunks of that content. Then it launches TCP sessions with the users that hold subparts of the requested file, to download all subparts of that file (Figure 3). When a download of a subpart is complete, TCP session is closed.

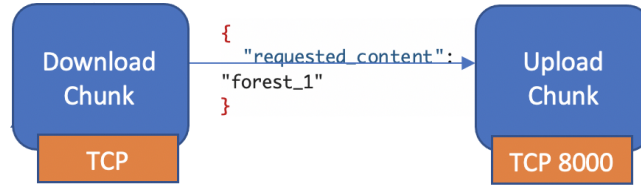


Figure 3: The Chunk Announcer broadcasts its chunks to all nodes in the network.

Serving (parts of) files: Every user has one original file, which, they will divide into 5 chunks. In addition, every user will altruistically serve all pieces of other users' files that they downloaded. When a new TCP connection request is received, this connection request is immediately accepted, the received message is parsed, and the requested file is sent to the requesting node.

Download history: End user can view the download and upload history (date/time, which user, content name, chunk index).

2 Requirements

Throughout this section, the term **shall** indicates an obligatory requirement that must be met to comply with the specification, whereas the term **may** indicates an item that is truly optional.

2.1 Chunk Announcement Requirements

Req. #	Requirement
2.1.0-A	When launched, Chunk_Announcer shall ask the user to specify the file it will initially host (<i>i.e.</i> , that user's original file.) Chunk_Announcer shall divide the specified file into N -byte chunks and store them as separate files with indexed naming. (The code for this will be provided to you.) Once this is done, Chunk_Announcer shall display a message on terminal, informing the end user about the number of chunks, and stating it is starting to announce these files. To simplify the design, let's assume each file in our system always has 5 chunks.
2.1.0-B	After preparing the chunks, Chunk_Announcer shall start to periodically send broadcast UDP messages in the network (to announce the chunks on this host). The period shall be once per minute. For the broadcast IP address, please use 25.255.255.255, which is the broadcast address within my private network in Hamachi.
2.1.0-C	Chunk_Announcer shall be able to read the names of the files under a specified directory, and insert them into the message in JSON array format.
2.1.0-D	Chunk_Announcer's periodic broadcasts shall contain a JSON including the list of hosted files. It is <u>very important</u> that the key is "chunks" (all lowercase) and that the format is a valid JSON format; otherwise you may have parsing issues when working with your peers. An example would look like: <code>'{"chunks": ["forest_1", "forest_2", "forest_3", "flower_2", "bee_1", "city_1", "city_2"]}'</code> .

2.2 Content Discovery Requirements

Req. #	Requirement
2.2.0-A	Content_Discovery shall listen for UDP broadcast messages on port 5001.
2.2.0-B	Upon receiving a broadcast message, Content_Discovery shall : (i) parse the message contents using a JSON parser in Python, (ii) get the UDP broadcast sender's IP address using recvfrom() method.
2.2.0-C	Content_Discovery shall store the list of files (parsed from the JSON message) in a dictionary. Let's call this <i>the content dictionary</i> . The dictionary keys shall be the <i>content chunk name</i> (e.g., forest_1) and the value shall be an array containing the list of IP addresses having that chunk (that you fetched using recvfrom()). This dictionary shall be shared with the Chunk_Downloader process. You may store it in a local text file that is shared between the Content_Discovery and Chunk_Downloader components.
2.2.0-D	Upon every insertion into <i>content dictionary</i> , Content_Discovery may display the detected user (IP address) and their hosted content on the console (e.g., "192.168.2.5 : vid_1, vid_2, vid_3"). This would also help you with debugging.

2.3 Chunk Downloader Requirements

Req. #	Requirement
2.3.0-A	When launched, Chunk_Downloader shall prompt the user to specify which content it wants to download. For the user-entered filename, (for example “forest.png”), Chunk_Downloader shall initiate 5 sequential download procedures for each chunk of this file, as described in the following requirements.
2.3.0-B	To download each of the 5 chunks of the content that the user wants to download, Chunk_Downloader shall <i>first</i> lookup its <i>content dictionary</i> to fetch the list of IP addresses having a certain chunk. For this, it’ll lookup the dictionary with the key set to chunk name (e.g. “forest_1”), which requires a lookup to the local file that Chunk_Discovery wrote the content dictionary into. Chunk_Downloader shall try downloading a chunk from the first IP address in the array that is in the <i>content dictionary</i> for this chunk name.
2.3.0-C	For downloading each chunk, Chunk_Downloader shall initiate a TCP session with the IP address (determined in the previous requirement). The message shall contain a JSON that contains the user-specified content name. It is <u>very important</u> that the field name is exactly “requested_content” and the format is a valid JSON format; otherwise you may have parsing issues when working with your peers. An example would look like: {“requested_content”: “forest_1” } (Please note that the script I provide does not add the type suffix such as .png, so we will not use suffix when requesting chunks.)
2.3.0-D	If download is successful, Chunk_Downloader shall move on to the next chunk. If it is not successful, Chunk_Downloader shall try downloading from the other users in the array until download of that chunk is successful. If all users in array have been tried and that chunk cannot be downloaded, Chunk_Downloader shall display a warning message to the user informing about the problem. (e.g., “CHUNK forest_3 CANNOT BE DOWNLOADED FROM ONLINE PEERS.”)
2.3.0-E	Chunk_Downloader shall close a TCP session upon receiving the chunk it requested.
2.3.0-F	Without running a validation on the downloaded content, we’ll assume the file is correctly downloaded when all 5 chunks have been downloaded. After the 5 th chunk is downloaded, Chunk_Downloader shall combine these 5 chunks into a single file. (I’ll provide the code for this, which you’ll integrate in your Chunk_Downloader code.) Once the file is ready, the Chunk_Downloader shall inform the user via the terminal that the file has been successfully downloaded. Please do not remove individual chunks after obtaining the merged content – we may use the chunks on the demo day.
2.3.0-G	Chunk_Downloader shall dump all downloaded filenames in a Download log (a text file) under the same directory. Each entry shall specify timestamp, chunk_name, downloaded_from_IP_address.
2.3.0-H	After a TCP session is closed, Chunk_Downloader shall persist; the service shall not terminate.

2.4 Chunk Uploader Requirements

Req. #	Requirement
2.4.0-A	Chunk_Uploader shall listen for TCP connections on port 8000.
2.4.0-B	Chunk_Uploader shall parse the JSON in the message to learn which chunk is being requested by the sender, and it shall send this file to the requester over TCP.
2.4.0-C	Chunk_Uploader shall accept TCP connection request before it times out, and shall successfully send the content requested by the process at the other end. For this, the process shall be able to locate the file and transfer its bytes as a stream.
2.4.0-D	Upon sending a chunk, Chunk_Uploader shall log the file's info in a text file under the same directory. Each entry shall specify chunk name, timestamp, destination IP address.
2.4.0-E	After a TCP session is closed, Chunk_Uploader shall persist; the service shall not terminate.

2.5 Performance Requirements

Req. #	Requirement
2.5.0-A	All four processes shall run on Python 3.
2.5.0-B	Chunk_Announcer shall correctly form the JSON from the contents in its local directory.
2.5.0-C	Chunk_Discovery shall be able to detect all online users and the files they host.
2.5.0-D	Content dictionary shall be able to keep up to 5 users' contents, with each having up to 5 chunks.
2.5.0-E	Chunk_Downloader shall be able to download a file from any online user, with no perceivable delay.
2.5.0-F	Any unspecified configuration is a plus – displaying download progress, displaying error message when file chunks can't be downloaded from peers, etc.