COMP3331 Assignment Report BitTrickle

z5421277 - Tony (Hexuan) Wang

Details of Language & Code Organisation:

Client Module (client.py): Contains the client-side operations, including server authentication, command handling, file publication, search, file transfer, and heartbeat management.
Server Module (server.py): Manages server-side operations, such as user authentication, command processing, tracking of active users, heartbeat validation, file publication, and storing published file details. I have chosen python also because of the self contained nature of the language, that requires no compilation steps and is easily executable in different directories making it perfect to test different clients.

Overall Program Design:

The program design for BitTrickle works by a client to server system, the client communicates to the server via UDP protocols to receive server validated data, but within the client itself, the client is both a TCP server and client to communicate to other clients. The client only uses the server as a ledger to receive validated active users and their TCP ports to request and send files to the other client's TCP server.

Client Module (client.py):

- Manages connection to the server and supports commands such as pub (publish file), get (request file), sch (search for files), and xit (exit).
- Starts a heartbeat thread to keep the server informed of its active status.
- Uses the FileTransferServer and FileTransferClient classes to handle file uploads and downloads between peers.

Server Module (server.py):

- Listens for client connections over UDP.
- Validates client authentication and processes commands such as file publication, retrieval, and searching.
- Manages file metadata, active clients, and peer-to-peer connections through structured data dictionaries.

The server initialises by loading a list of valid credentials from credentials.txt and pre-loading previously published files from published_files.json. Once activated it:

- Accepts client connections and authenticates based on credentials.
- Starts a thread to monitor heartbeats, removing clients if their heartbeat signals are timed out over 3 seconds.

```
inactive_users = [user for user, last_beat in active_users.items() if now - last_beat >
timedelta(seconds=3)]
```

Client

The client sets up a UDP connection to the server for command communication and an independent TCP port for file transfers. After successful authentication, it starts a heartbeat thread to maintain its active state on the server.

```
tcp_socket = socket(AF_INET, SOCK_STREAM
tcp_socket.bind(("0.0.0.0", 0)) # Bind to an available port
self.tcp_port = tcp_socket.getsockname()[1]
```

COMP3331 Assignment Report BitTrickle
z5421277 - Tony (Hexuan) Wang

Program Components and Interaction
Client Components: The Client class contains functions for command handling, authentication, heartbeat signalling, and file transfers.
Authentication:
The client provides a username, password, and TCP port to the server.
The server validates the credentials against credentials.txt, allowing access only upon a match.
Command Handling: Commands include pub, get, sch, lpf, and xit.
Client side: pub verifies that the file exists locally before sending the command to the server. Helps prevent a file not found error when trying to send it out when requested in future.

```
if os.path.isfile(filename):
    self.udp_socket.sendto(command.encode(), self.server_address)
```

FileTransferServer class listens for file requests on the client's TCP port and sends the file if it exists.
FileTransferClient class connects to a peer's TCP port to download a requested file.

Server Components: The **Server** class performs authentication, command processing, and file management.
Authentication: Credentials are validated against credentials.txt. Each user provides a unique TCP port for file transfers.

```
        if username in credentials and credentials[username] == password:
                active_users[username] = datetime.now()
                client_tcp_ports[username] = tcp_port
```

Command Processing: Commands include file publication (pub), file request (get), and file search (sch). The get command retrieves the IP and port of a peer sharing the requested file, facilitating direct peer-to-peer file transfer.
Server side: validates that command arguments are valid, if not sends error, prevents timing out and getting the program logically stuck.
Heartbeat Monitoring: Heartbeat signals are monitored in a separate thread. Clients that miss heartbeats are removed from active sessions.

Data Structure Design:
active_users: Tracks each client's last heartbeat timestamp to identify active users and remove inactive ones.
Schema:
- **Key**: username (string) – The unique username of the client.
- **Value**: last_heartbeat (datetime) – The timestamp of the client's last heartbeat.

Dictionary: active_users = { "username1": "2024-11-05T14:23:45.123456", "username2": "2024-11-05T14:23:45.123456"

client_addresses: Maps each client's UDP address (IP and port) to their username, enabling the server to identify clients based on their network address.
Schema:
- **Key**: client_address (tuple) – A tuple containing the IP address (string) and UDP port (integer) of the client.
- **Value**: username (string) – The username associated with the client's IP and UDP port.

Dictionary: client_addresses = { ("127.0.0.1", 54321): "username1", ("127.0.0.1", 54322): "username2" }

client_tcp_ports: Tracks each client's TCP port for file transfer, allowing the server to provide connection details for peer-to-peer file transfers.

COMP3331 Assignment Report BitTrickle
z5421277 - Tony (Hexuan) Wang

Schema:

- **Key**: username (string) – The unique username of the client.
- **Value**: tcp_port (integer) – The TCP port number the client is using for file transfers.

Dictionary: client_tcp_ports = { "username1": 60000, "username2": 60001}

published_files: Stores each user's published files, enabling the server to manage and track which files each client has made available for sharing. This dictionary is also serialised to published_files.json for persistence.

Schema:

- **Key**: username (string) – The unique username of the client who published the file.
- **Value**: file_set (set of strings) – A set of filenames that the user has published.

Dictionary:Sets: published_files = {"username1": {"file1.txt", "file2.txt"}, "username2": {"file3.txt", "file4.txt"} }

Application Layer Protocol Message Format(s):
Client to Server Communication

- **Authentication**: "login:<username> <password> <tcp_port>"
- **Heartbeat**: "HBT <username>"
- **Commands**:
  - pub <filename>: Publish file
  - get <filename>: Request file
  - sch <substring>: Search for files with substring
  - xit: Exit session

Server to Client Responses

- **Success**: "OK:<message>"
- **Error**: "ERR:<error_message>"
- File request response example: OK:<peer_ip>:<peer_port>

Peer-to-Peer File Transfer

- **Request**: The requesting client connects to the peer's TCP port and sends the filename.
- **Response**: If the file exists, the server responds with "OK" and streams the file data. If the file is missing, the response is "ERR: File not found".

Known Limitations:
I have not added a client timeout on the UDP socket thus if the server doesn't respond to a client command (e.g., due to network delays or server issues), setting a timeout allows the client to retry or report the issue without waiting indefinitely. No time out on the TCP file transfer server, if a peer client connects but doesn't initiate a file request within a specified time, the server does not close the connection to avoid idle connections.