Module **`p2p`**

Classes

```
class P2P (host='127.0.0.1', port=5000, shared_dir='shared',
         downloads_dir='downloads')
```

Peer-to-Peer File Sharing Node

A complete P2P node that can discover peers, share files, search the network, and transfer files in chunks with integrity verification.

Attributes

`host` : `str`
: IP address this node listens on

`port` : `int`
: Port number this node listens on

`server_socket` : `socket`
: Server socket for accepting connections

`running` : `bool`
: Server running state

`peers` : `dict`
: Dictionary of connected peers {peer_id: peer_info}

`peer_id` : `str`
: Unique identifier for this peer (host:port)

`heartbeat_interval` : `int`
: Seconds between heartbeat checks

`shared_dir` : `Path`
: Directory containing files to share

`file_index` : `dict`
: Index of shared files {filename: metadata}

`search_results` : `dict`
: Temporary storage for search operations

search_timeout : `int`
: Timeout for search operations in seconds

search_lock : `threading.Lock`
: Thread-safe lock for search operations

chunk_size : `int`
: Size of file chunks in bytes (8KB)

downloads_dir : `Path`
: Directory where downloaded files are saved

Initialize a P2P node

Args

host : `str`
: IP address to bind to. Default: '127.0.0.1'

port : `int`
: Port number to listen on. Default: 5000

shared_dir : `str`
: Directory name for shared files. Default: 'shared'

downloads_dir : `str`
: Directory name for downloads. Default: 'downloads'

Note

Port number is appended to directory names to allow multiple nodes on the same machine (e.g., 'shared5000', 'downloads5001')

Methods

`def accept_connections(self)`

Accept incoming peer connections (runs in background thread)

Continuously accepts new connections and spawns a new thread to handle each client using handle_client().

Runs while self.running is True

`def add_peer(self, peer_info)`

Add a peer to the known peers list. Updates self.peers dictionary with peer information and current timestamp. Won't add itself to peer list.

Args

`peer_info` : `dict`
: Peer information containing: - peer_id: Unique identifier (host:port) - host: IP address - port: Port number

`def `**`aggregate_search_results`**`(self, search_id)`

Aggregate search results from multiple peers. Groups results by filename and collects all peers that have each file. Removes duplicate peer entries for the same file.

Args

`search_id` : `str`
: Unique ID for the search operation

Returns

`list`
: Aggregated results grouped by filename: [{'filename': str, 'size': int, 'hash': str, 'peers': [{'peer_id': str, 'host': str, 'port': int}]}]

`def `**`announce_to_peers`**`(self)`

Announce presence to all known peers

Attempts to connect to each known peer, triggering peer discovery. Used to refresh connections or announce rejoining the network.

Silently ignores connection failures to individual peers

`def `**`calculate_file_hash`**`(self, filepath)`

Calculate SHA256 hash of a file Reads file in chunks to handle large files efficiently without loading entire file into memory. Chunk Size: 4096 bytes for hashing

Args

`filepath` : `str or Path`
: Path to the file

Returns

`str`
: Hex string of SHA256 hash or None if error

`def` **`cleanup_dead_peers`**`(self)`

Remove unresponsive peers from peer list (runs in background thread)

Checks every 15 seconds for peers that haven't responded to heartbeat in more than 30 seconds and removes them from the peer list.

Timeout: 30 seconds without heartbeat response Check Interval: 15 seconds Runs while self.running is True

Removes dead peers from self.peers dictionary Prints cleanup messages to console

`def` **`connect_to_peer`**`(self, peer_host, peer_port)`

Connect to another peer and perform peer discovery

Args

`peer_host` : `str`
: IP address of peer to connect to

`peer_port` : `int`
: Port number of peer

Returns

`bool`
: True if connection successful, None if failed

`def` **`display_search_results`**`(self, results)`

Pretty print search results

Args

`results` : `list`
: Search results from search_network() or aggregate_search_results()

`def` **`download_file`**`(self, peer_host, peer_port, filename)`

Download a file from a peer using chunked transfer

Downloads the file in chunks, displays progress, and verifies integrity using SHA256 hash comparison.

Args

`peer_host` : `str`
: IP address of the peer to download from

`peer_port` : `int`
: Port number of the peer

`filename` : `str`
: Name of the file to download

Returns

`bool`
: True if download successful and integrity verified, False otherwise

`def **get_file_info**(self, filename)`

Get metadata for a specific file

`def **get_file_list**(self)`

Get list of all available filenames

`def **handle_client**(self, client_socket, address)`

Handle communication with a connected peer. Runs until client disconnects or error occurs

Args

`client_socket` : `socket`
: Connected socket for this peer

`address` : `tuple`
: (host, port) of the connected peer

Special Handling: - REQUEST_CHUNK messages are handled directly and close connection after sending - All other messages are passed to process_message() for handling

`def **list_files**(self)`

Display all shared files with detailed information

`def **list_peers**(self)`

Display list of all connected peers

`def **loop_heartbeat**(self)`

Continuously send heartbeat to all peers (runs in background thread)

`def **process_message**(self, message, address)`

Process incoming messages and generate appropriate responses

Args

`message` : `dict`
: JSON message received from peer

`address` : `tuple`
: (host, port) of the sender

Returns

`dict`
: JSON response message or None

Supported Message Types: - PEER_DISCOVERY: Peer announces itself, returns peer list - HEARTBEAT: Health check, returns acknowledgment - GET_PEERS: Request peer list - MESSAGE: Generic message with content - SEARCH_REQUEST: Search for files - GET_FILE_LIST: Request list of shared files - GET_FILE_INFO: Request metadata for specific file - REQUEST_CHUNK is handled separately in handle_client()

`def **refresh_index**(self)`

Refresh the file index by rescanning the shared folder

`def **request_chunk**(self, peer_host, peer_port, filename, chunk_index)`

Request a specific chunk of a file from a peer

Args

`peer_host` : `str`
: IP address of the peer

`peer_port` : `int`
: Port number of the peer

`filename` : `str`
: Name of the file

`chunk_index` : `int`

Index of the chunk to download (0-based)

Returns

`bytes`
: Chunk data or None if failed

Protocol

1. Send REQUEST_CHUNK message with JSON header
2. Receive CHUNK_DATA header with chunk size
3. Receive raw binary chunk data

`def` **`request_file_info`**`(self, peer_host, peer_port, filename)`

Request detailed metadata for a specific file from a peer Prints file information to console

Args

`peer_host` : `str`
: IP address of the peer

`peer_port` : `int`
: Port number of the peer

`filename` : `str`
: Name of the file to query

Returns

`dict`
: File metadata {'path': str, 'size': int, 'hash': str} or None if not found

`def` **`request_file_info_for_download`**`(self, peer_host, peer_port, filename)`

Request file metadata from a peer before downloading

Args

`peer_host` : `str`
: IP address of the peer

`peer_port` : `int`

Port number of the peer

filename : `str`
Name of the file to query

Returns

`dict`
File metadata {'path': str, 'size': int, 'hash': str} or None if failed

`def` **`request_peer_file_list`**`(self, peer_host, peer_port)`

Request list of all files from a specific peer. Prints formatted list of files to console.

Args

peer_host : `str`
IP address of the peer

peer_port : `int`
Port number of the peer

Returns

`list`
List of filenames available on the peer or empty list if failed

`def` **`scan_shared_folder`**`(self)`

Scan the shared folder and index all files

Creates file_index dictionary with metadata for each file: - filename (key) - path: absolute path to file - size: file size in bytes - hash: SHA256 hash for integrity verification

Updates self.file_index with current state of shared folder Prints indexing progress to console

`def` **`search_local`**`(self, query)`

Search for files in the local file index

Args

query : `str`

Search term (case-insensitive, partial match)

Returns

`list`
: List of matching files with metadata: [{'filename': str, 'size': int, 'hash': str, 'peer_id': str, 'peer_host': str, 'peer_port': int}]

`def search_network(self, query)`

Search for files across all connected peers. Broadcasts search request and aggregates results. Results from multiple peers are grouped by filename.

Args

`query` : `str`
: Search term (case-insensitive, partial match)

Returns

`list`
: Aggregated search results: [{'filename': str, 'size': int, 'hash': str, 'peers': [{'peer_id': str, 'host': str, 'port': int}]}]

`def send_chunk(self, client_socket, filename, chunk_index, chunk_size)`

Send a specific chunk of a file to a requesting peer. If file not found, sends error message instead

Args

`client_socket` : `socket`
: Connected socket to send chunk through

`filename` : `str`
: Name of the file to send chunk from

`chunk_index` : `int`
: Index of the chunk to send (0-based)

`chunk_size` : `int`
: Size of chunk to read in bytes

`def `**`send_heartbeat`**`(self, peer_host, peer_port)`

Send heartbeat message to a specific peer Silently fails if connection cannot be established

Args

`peer_host` : `str`
IP address of peer

`peer_port` : `int`
Port number of peer

`def `**`send_message`**`(self, peer_socket, message)`

Send a generic JSON message to a peer and wait for response This is a utility method for simple request-response patterns

Args

`peer_socket` : `socket`
Connected socket to send message through

`message` : `dict`
Message dictionary to send (will be JSON encoded)

Returns

`dict`
Response message or None if failed

`def `**`send_search_request`**`(self, peer_host, peer_port, query, search_id)`

Send a search request to a specific peer Updates self.search_results[search_id] with peer's results Increments responses_received counter This method is called in a separate thread for each peer

Args

`peer_host` : `str`
IP address of the peer

`peer_port` : `int`
: Port number of the peer

`query` : `str`
: Search term

`search_id` : `str`
: Unique ID for this search operation

`def` **`start_server`**`(self)`

Start the P2P server

Initializes the server socket, binds to host:port, and starts three background threads: - accept_connections: Accepts incoming peer connections - loop_heartbeat: Sends periodic heartbeat to all peers - cleanup_dead_peers: Removes unresponsive peers

Also scans the shared folder to index available files.

`def` **`stop_server`**`(self)`

Stop the server and close all connections

Classes

`P2P`

`accept_connections`
`add_peer`
`aggregate_search_results`
`announce_to_peers`
`calculate_file_hash`
`cleanup_dead_peers`
`connect_to_peer`
`display_search_results`
`download_file`
`get_file_info`
`get_file_list`
`handle_client`
`list_files`
`list_peers`
`loop_heartbeat`
`process_message`
`refresh_index`
`request_chunk`
`request_file_info`

`request_file_info_for_download`
`request_peer_file_list`
`scan_shared_folder`
`search_local`
`search_network`
`send_chunk`
`send_heartbeat`
`send_message`
`send_search_request`
`start_server`
`stop_server`