BECE204L

MICROPROCESSORS AND MICROCONTROLLERS

PROJECT - REPORT

SYRENT- PYTHON BASED TORRENT CLIENT

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

21BCE3137 SHANTANU ANAND

Under the Supervision of

Dr. KALYANBRATA GHOSH

Assistant Professor Senior Grade 1

School of Electronics Engineering

November 2024

TABLE OF CONTENTS

ABSTRACT

Syrent is a Python-based torrent client developed to provide an enhanced, feature-rich, and user-friendly alternative to traditional torrent clients. By leveraging modern tools and libraries, Syrent delivers a seamless experience for managing torrent files and magnet links while maintaining a lightweight and highly modular architecture. The primary goal of Syrent is to simplify torrent management tasks, integrate support for magnet link conversion to .torrent files, and provide a visually appealing graphical user interface (GUI).

The GUI, built using PyQt5, ensures ease of use and offers features like drag-and-drop file addition, customizable download directories, and a theme toggle between light and dark modes. The intuitive interface is designed to allow users to add, pause, resume, or remove torrents effortlessly. The core functionalities are powered by asynchronous operations using Python's asyncio, ensuring responsiveness and efficiency, even with multiple concurrent downloads. Key torrent-related tasks, such as file piece tracking, are managed efficiently using the bitarray library.

Syrent introduces an advanced feature for converting magnet links to .torrent files. This capability, implemented via the integration of the Node.js-based m2t tool, simplifies the process of obtaining .torrent files directly from magnet links, providing users with greater flexibility. This feature is integrated seamlessly with the application, ensuring smooth user interaction without requiring external configurations.

The backend of Syrent is built using several robust libraries. aiohttp handles network communication, including interactions with trackers and peers, while bencodepy manages torrent file parsing and manipulation. async-timeout is utilized for managing network task timeouts, ensuring smooth and responsive operations. The modular design allows for scalability, making Syrent adaptable for future enhancements like distributed file-sharing models or encrypted peer-to-peer communication.

Syrent also supports robust state management, allowing users to save and restore torrent states, including paused downloads. It emphasizes error handling and recovery mechanisms, ensuring reliability. The project follows modern coding practices and provides clear documentation for developers, making it an excellent learning tool for understanding torrent protocols and Python-based asynchronous programming.

To ensure platform compatibility, Syrent integrates the m2t tool via a subprocess call, requiring Node.js and npm for proper functionality. This dependency is clearly documented to allow easy setup on different host machines. Paths and configurations are designed to be adaptable to the host environment, ensuring flexibility across systems.

This report provides a detailed overview of the design and implementation of Syrent, including its technical architecture, dependencies, and user interface. The project demonstrates the potential of Python and modern libraries to build efficient, scalable, and user-centric applications. Syrent is not only a practical tool for torrent management but also a demonstration of how modern software design principles can be applied to enhance legacy protocols like BitTorrent. The project serves as a foundation for further exploration in peer-to-peer networks and distributed systems, showcasing the versatility of Python for such applications.

1. INTRODUCTION

1.1 Background

The advent of peer-to-peer (P2P) file-sharing systems revolutionized how data is distributed across networks, with the BitTorrent protocol emerging as one of the most popular methods for efficient and scalable file sharing. BitTorrent operates on the principle of decentralized distribution, allowing multiple peers to contribute to the sharing process. Over the years, several torrent clients have been developed, catering to users' varying needs. However, these clients often present usability challenges, limited customization options, or resource-heavy implementations. Syrent was conceived as a modern, lightweight, and user-friendly torrent client built using Python, which leverages contemporary programming practices and asynchronous operations to deliver a responsive and reliable application. With an emphasis on simplicity, functionality, and a visually appealing interface, Syrent addresses several limitations of existing torrent clients while introducing new features such as magnet link conversion.

1.2 Motivations

The motivation behind Syrent stems from the following key drivers:

I. **Enhancing User Experience:** Most existing torrent clients provide limited flexibility in managing torrent files and magnet links. Syrent aims to create a more intuitive and customizable interface that simplifies these processes.

II. **Educational Value:** The project serves as an excellent learning experience for exploring Python’s capabilities in network communication, asynchronous programming (asyncio), and GUI development (PyQt5).

III. **Integration of Modern Tools:** Incorporating tools like Node.js’s m2t allows users to seamlessly convert magnet links to .torrent files, a feature often unavailable in traditional clients.

IV. **Improving Efficiency:** Syrent’s backend leverages Python libraries like aiohttp, bencodepy, and bitarray for lightweight, efficient handling of torrent protocols, ensuring smooth operations even with concurrent downloads.

V. **Open-Source Contribution:** Syrent provides a modular and extensible design, making it a valuable open-source project for developers interested in learning or contributing to the BitTorrent ecosystem.

By addressing these areas, Syrent not only improves usability for end-users but also promotes innovation within the developer community by demonstrating how Python can be used to build scalable, real-world applications.

1.3 Scope of the Project

Syrent focuses on delivering a modern torrent client with the following capabilities:

- **Torrent Management:** Users can add, pause, resume, and remove torrent downloads through an intuitive GUI interface.
- **Magnet Link Conversion:** The integration of the m2t tool allows users to convert magnet links to .torrent files effortlessly, reducing dependency on external tools.
- **Thematic Customization:** Syrent offers light and dark theme toggling, enhancing the visual appeal and accessibility for users with different preferences.
- **Download Management:** Users can specify download directories and selectively download files from multi-file torrents, offering greater control over the process.
- **Scalable Backend:** Leveraging asyncio for asynchronous operations ensures the application remains responsive even when handling multiple downloads simultaneously.
- **Platform Compatibility:** Syrent is designed to operate seamlessly across systems, with clear documentation for dependencies like Node.js, Python libraries, and required configurations.

While the primary focus is on torrent management, Syrent's extensible architecture provides a foundation for future enhancements. Potential extensions include encrypted peer-to-peer communication, distributed file-sharing models, and advanced analytics on download patterns. By addressing the challenges of usability, flexibility, and scalability, Syrent establishes itself as a robust and versatile solution in the BitTorrent ecosystem.

2. PROJECT DESCRIPTION AND GOALS

2.1 Literature Review

BitTorrent, a peer-to-peer file-sharing protocol, was introduced by Bram Cohen in 2001 to optimize file distribution over the internet. Since then, several torrent clients have been developed, each addressing specific aspects of torrent management. Notable examples include:

I. **µTorrent (uTorrent):** A lightweight client offering a compact user interface but often criticized for bundling unwanted software and displaying intrusive advertisements.

II. **qBittorrent:** An open-source alternative to µTorrent, providing a clean interface and advanced functionality like RSS feed support and integrated search.

III. **Deluge:** A highly modular client offering plugins for added customization but requiring significant technical expertise for advanced configurations.

IV. **Transmission:** Known for its minimalistic design, Transmission focuses on low resource usage but lacks certain features expected by power users.

While these clients address basic torrenting needs, they often fail to provide flexibility in integrating new tools or customizing user experiences. They also rarely focus on developer usability, which limits their potential for educational purposes or community-driven enhancements.

Additionally, tools like m2t for magnet-to-torrent conversion have demonstrated how external utilities can simplify common torrenting tasks. However, such tools are usually standalone and lack seamless integration into a unified torrent management system.

2.2 Research Gap

From the review, several gaps emerge in the existing landscape of torrent management tools:

I. **Lack of Integration:** Most torrent clients do not natively support magnet link-to-torrent conversion, requiring users to rely on external tools. This creates inefficiencies in workflow.

II. **High Learning Curve for Developers:** Existing open-source torrent clients often feature monolithic, complex architectures, making it difficult for developers to study or contribute.

III. **Limited GUI Customization:** Few clients offer modern user interfaces or visual customization like light/dark themes, which is critical for accessibility and user preference.

IV. **Resource Intensity:** Many clients, especially feature-rich ones, consume significant system resources, making them unsuitable for lightweight systems or older hardware.

V. **Lack of Modular Design:** Existing solutions lack extensible architectures, restricting their adaptability for modern needs such as integrating analytics or advanced encryption protocols.

These gaps highlight the need for a modern torrent client that emphasizes integration, customization, efficiency, and developer-friendly design.

2.3 Objectives

The Syrent project aims to address the identified gaps with the following objectives:

- **Seamless Magnet Link Conversion:** Incorporate the m2t tool into Syrent, enabling users to convert magnet links to .torrent files directly from the GUI.

- **Enhanced User Interface:** Provide a clean, customizable GUI with light and dark theme support for better user experience.

- **Developer-Friendly Design:** Ensure the application is modular and well-documented, allowing developers to study and extend its functionality.

- **Efficient Resource Utilization:** Utilize Python libraries like aiohttp and asyncio to ensure the application remains lightweight and responsive, even under heavy load.

- **Intuitive File Management:** Enable users to selectively download files from multi-file torrents and define download directories, providing greater control over the process.

- **Cross-Platform Compatibility:** Ensure the application runs seamlessly on all major operating systems (Windows, macOS, and Linux) with clear setup instructions.

- **Extensibility:** Design the architecture to support future enhancements, such as encrypted communication or distributed file-sharing models.

By achieving these objectives, Syrent will serve as a robust, user-friendly torrent client while offering developers a foundation for innovation in the BitTorrent ecosystem.

2.4 Problem Statement

Despite the widespread adoption of peer-to-peer file-sharing protocols, existing torrent clients continue to exhibit significant limitations that hinder their effectiveness for end-users and developers alike. While clients like µTorrent, qBittorrent, and Transmission provide basic torrenting functionality, they fail to comprehensively address modern requirements for integration, resource efficiency, and customization.

A major shortcoming is the lack of seamless integration for commonly used tools, such as magnet-to-torrent converters like m2t. Most clients do not natively support this feature, forcing users to rely on external command-line utilities, which creates unnecessary friction in the user experience. Furthermore, existing clients frequently prioritize feature density over resource efficiency, making them unsuitable for lightweight systems or users requiring minimalistic solutions.

For developers, current torrent clients present a steep learning curve due to their monolithic and tightly coupled architectures. This complexity discourages customization and prevents easy adaptation to modern technological needs, such as encrypted peer-to-peer communication, advanced analytics, or distributed file-sharing models. Limited documentation and developer support further exacerbate these challenges.

In addition, while graphical user interfaces (GUIs) are integral to user engagement, most torrent clients lack modern, visually appealing, and customizable interfaces. Features like light and dark themes, which significantly enhance accessibility and user satisfaction, are often overlooked. The inability to selectively download specific files from multi-file torrents also restricts user control and efficiency.

The lack of modularity in current solutions limits their scalability and adaptability, making it difficult to integrate emerging features or technologies. This is particularly problematic in an era where security, privacy, and user control are paramount.

Expanded Problem Statement:

"How can a modern torrent client be designed to overcome the usability, integration, and efficiency challenges of existing solutions, while providing an extensible, developer-friendly architecture? The solution must cater to both end-users and developers, incorporating features such as seamless magnet-to-torrent conversion, an intuitive and customizable GUI, resource-efficient performance, and a modular design that supports future technological advancements in the peer-to-peer ecosystem."

This problem serves as the foundation for Syrent's development, aiming to create a comprehensive, efficient, and extensible torrent client tailored to meet evolving user and developer needs.

2.5 Project Plan

The development of Syrent will follow a structured project plan, divided into distinct phases to ensure systematic progress and timely delivery. The plan accounts for both functional requirements and technical challenges, focusing on user-centric design and extensibility.

Phase 1: Requirement Analysis and Research

- **Objective: Understand the needs of end-users and developers, and identify the technological stack.**
- **Tasks:**
  - Conduct a detailed literature review of existing torrent clients.
  - Identify gaps in current solutions.
  - Finalize dependencies, including Python libraries (PyQt5, aiohttp, bencodepy) and external tools (m2t for magnet-to-torrent conversion).
  - Define clear objectives and deliverables.

Phase 2: Architecture Design

- **Objective: Develop a modular and extensible architecture for Syrent.**
- **Tasks:**
  - Design the core components: GUI, control management, magnet-to-torrent integration, and file management.
  - Define interfaces for future integration of advanced features (e.g., encryption, analytics).
  - Ensure compatibility with Windows, macOS, and Linux platforms.

Phase 3: Development

- **Objective: Build Syrent's core functionalities.**
- **Tasks:**
  - Implement the GUI using PyQt5, focusing on usability and customization (light/dark themes).
  - Integrate m2t for seamless magnet-to-torrent conversion.
  - Develop a control management system using aiohttp and asyncio for efficient torrent handling.
  - Implement selective file download and directory management features.

    - Optimize resource utilization to ensure lightweight performance.

Phase 4: Testing and Debugging

- **Objective: Validate Syrent's functionality and identify potential issues.**
- **Tasks:**
    - Perform unit testing for individual modules (e.g., GUI responsiveness, magnet-to-torrent conversion).
    - Conduct integration testing to ensure seamless interaction between components.
    - Test performance under varying workloads to optimize resource usage.
    - Validate cross-platform compatibility and address any environment-specific issues.

Phase 5: Documentation and User Guide

- **Objective: Provide comprehensive documentation for end-users and developers.**
- **Tasks:**
    - Write a user guide detailing installation, setup, and usage of Syrent.
    - Document the codebase to assist developers in understanding and extending the application.
    - Include information on dependencies and platform-specific configurations.

Phase 6: Deployment and Feedback

- **Objective: Release Syrent and gather feedback for future improvements.**
- **Tasks:**
    - Package the application for distribution on GitHub.
    - Highlight key features and setup instructions in the repository.
    - Collect user and developer feedback to identify areas for enhancement.

Project Milestones:

A. **Week 1-2: Requirement Analysis and Research.**

B. **Week 3-4: Architecture Design.**

C. **Week 5-8: Core Development.**

D. **Week 9-10: Testing and Debugging.**

E. **Week 11: Documentation and User Guide.**

F. **Week 12: Deployment and Feedback Collection.**

3. TECHNICAL SPECIFICATION

3.1 Requirements

3.1.1 Functional Requirements

I. **Torrent Download Management:**
   - Support for downloading torrent files via magnet links or .torrent files.
   - Ability to pause, resume, and remove downloads.
   - Selective file downloading from multi-file torrents.

II. **Magnet-to-Torrent Conversion:**
   - Seamless integration of the m2t tool for converting magnet links into .torrent files.
   - Save generated .torrent files to a user-specified directory.

III. **Graphical User Interface (GUI):**
   - User-friendly interface built with PyQt5.
   - Toggle between light and dark themes.
   - Display of download progress, file sizes, speed, and remaining time.

IV. **Directory and File Management:**
   - Allow users to specify custom directories for downloads.
   - Provide a file-tree visualization for multi-file torrents.

V. **Cross-Platform Compatibility:**
   - Ensure compatibility with Windows, macOS, and Linux platforms.

VI. **Feedback and Notifications:**
   - Display success, error messages, and progress updates for user actions.

3.1.2 Non-Functional Requirements

1. **Performance:**
   - Optimized resource usage for low memory and CPU consumption.
   - Asynchronous handling of download tasks to ensure smooth GUI performance.
2. **Security:**
   - Safe handling of user directories and torrent data.
   - No sensitive data should be stored permanently.
3. **Scalability:**
   - Modular architecture to accommodate future enhancements like encrypted downloads and advanced analytics.
4. **Usability:**
   - Intuitive design with minimal learning curve for users.
   - Comprehensive user guide for installation and usage.
5. **Extensibility:**
   - Open-source codebase for community contributions.
   - Simplified developer setup for future enhancements.

3.2 Feasibility Study

3.2.1 Technical Feasibility

- The project uses Python as the primary language, leveraging widely used libraries (PyQt5, aiohttp, bencodepy) for GUI and torrent management.
- Seamless integration of m2t, a Node.js-based tool, ensures magnet-to-torrent conversion functionality.
- Compatibility with multiple platforms (Windows, macOS, and Linux) is achievable with the chosen libraries.
- Asynchronous programming (asyncio) ensures efficient management of network requests and file operations.
- The lightweight architecture ensures the software runs effectively on modern systems with modest hardware specifications.

3.2.2 Economic Feasibility

- **Development Costs:**
    - The project is built using open-source tools and libraries, minimizing costs.
    - Python and Node.js ecosystems provide free libraries (m2t, PyQt5), eliminating licensing fees.
- **Maintenance Costs:**
    - The modular design reduces future maintenance overhead.
    - Community contributions (via GitHub) can further reduce maintenance costs.
- **Operational Costs:**
    - Users require only a one-time setup of dependencies (m2t, Python libraries).
    - Minimal hardware requirements ensure the project can run on most personal computers.

3.2.3 Social Feasibility

- **Ease of Use:**
    - The user-friendly GUI ensures the software can be used by both tech-savvy and non-technical users.
    - Features like light/dark themes improve accessibility for users with different visual preferences.
- **Open-Source Contribution:**
    - As an open-source project, Syrent fosters community collaboration, encouraging innovation and knowledge sharing.
- **Addressing Market Gaps:**
    - Provides a lightweight and customizable alternative to existing bloated torrent clients, addressing unmet needs in the peer-to-peer ecosystem.

3.3 System Specification

3.3.1 Hardware Specification

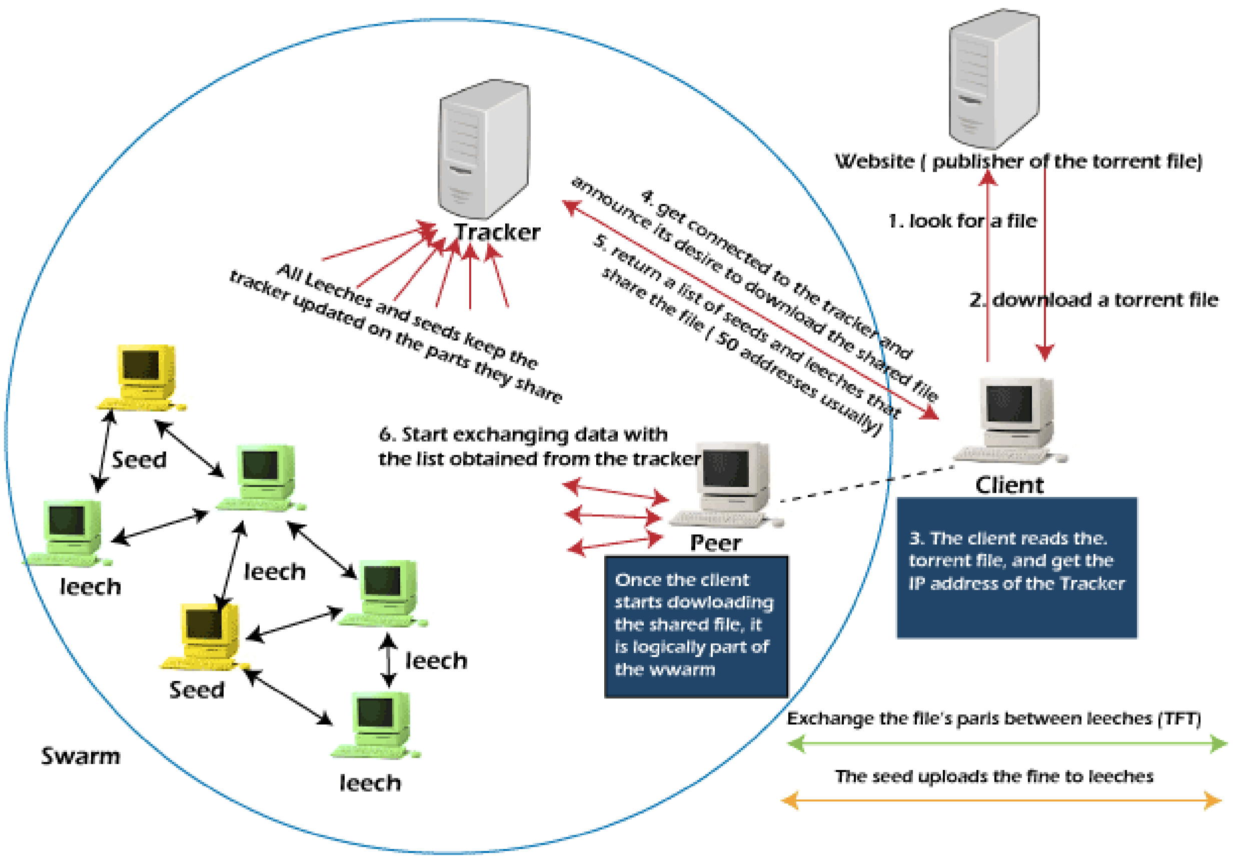
- **Processor:** Dual-core processor or higher (Intel i3 or equivalent).
- **RAM:** Minimum 2GB (4GB recommended for smooth performance).
- **Storage:** 50MB for application installation, plus additional space for downloaded files.
- **Graphics:** Basic graphics capability for rendering the PyQt5 GUI.
- **Network:** Stable internet connection for torrent downloads.

3.3.2 Software Specification

- **Operating System:**
  - Windows 10 or later.
  - macOS 10.14 or later.
  - Linux distributions (Ubuntu 18.04 or later, Fedora, etc.).
- **Dependencies:**
  - **Python Libraries:**
    - PyQt5 (for GUI design).
    - aiohttp (for asynchronous HTTP requests).
    - bencodepy (for parsing torrent files).
    - asyncio (for asynchronous task handling).
  - **Node.js Tool:**
    - m2t (for magnet-to-torrent conversion).
- **Python Version:** 3.9 or later.
- **Node.js Version:** v16.0.0 or later.
- **npm Version:** v7.0.0 or later.

4. DESIGN APPROACH AND DETAILS

4.1 System Architecture


Tracker
Website (publisher of the torrent file)
1. look for a file
2. download a torrent file
4. get connected to the tracker and announce its desire to download the shared file
5. return a list of seeds and leeches that share the file (50 addresses usually)
All Leeches and seeds keep the tracker updated on the parts they share
6. Start exchanging data with the list obtained from the tracker
Client
3. The client reads the torrent file, and get the IP address of the Tracker
Peer
Once the client starts dowloading the shared file, it is logically part of the wwarm
Seed
leech
leech
Seed
leech
leech
Swarm
Exchange the file's parts between leeches (TFT)
The seed uploads the fine to leeches

This diagram provides an overview of a torrent system architecture that highlights the key components and processes involved in a peer-to-peer file sharing system:

[1.] **Client Interaction with the Website**:

- The process begins with a user navigating to a website hosting a torrent file or magnet link. This website contains details about the file to be shared or downloaded via the peer-to-peer network.
- The user either downloads the .torrent file or retrieves a magnet link for the file. The magnet link contains information about the file's hash and tracker.

[2.] **Client Processing of Torrent Data**:

- Upon downloading the .torrent file or acquiring the magnet link, the client software (like Syrent) parses the data to extract the hash of the file and the tracker information.
- The tracker server is responsible for maintaining a list of peers currently participating in sharing or downloading the file.

[3.] **Connecting to the Tracker Server**:

- The client sends a request to the tracker server, providing the file hash and requesting information about peers (IP addresses) involved in sharing the file.
- The tracker responds with a list of peers in the swarm who are sharing or downloading the file.

[4.] **Joining the Swarm**:

- After obtaining the peer list, the client software connects to the peers and becomes part of the swarm.
- The swarm consists of two main types of participants:
  - **Seeders**: These are peers who have completed downloading the file and are only uploading it to others in the swarm.
  - **Leechers**: These are peers who are still downloading the file and, in most cases, are also uploading parts of the file they have already downloaded.

[5.] **Data Exchange in the Swarm**:

- The client software begins downloading pieces of the file from multiple peers in the swarm. Simultaneously, it uploads pieces of the file it has already downloaded to other peers. This decentralized sharing accelerates the file transfer process.
- The file transfer is optimized by exchanging small chunks of the file, ensuring no single peer becomes overloaded.

[6.] **Completing the Download**:

- Once all the pieces of the file are downloaded, the client software assembles them into the complete file.
- The client then transitions to becoming a seeder, contributing to the swarm by sharing the completed file with other peers.

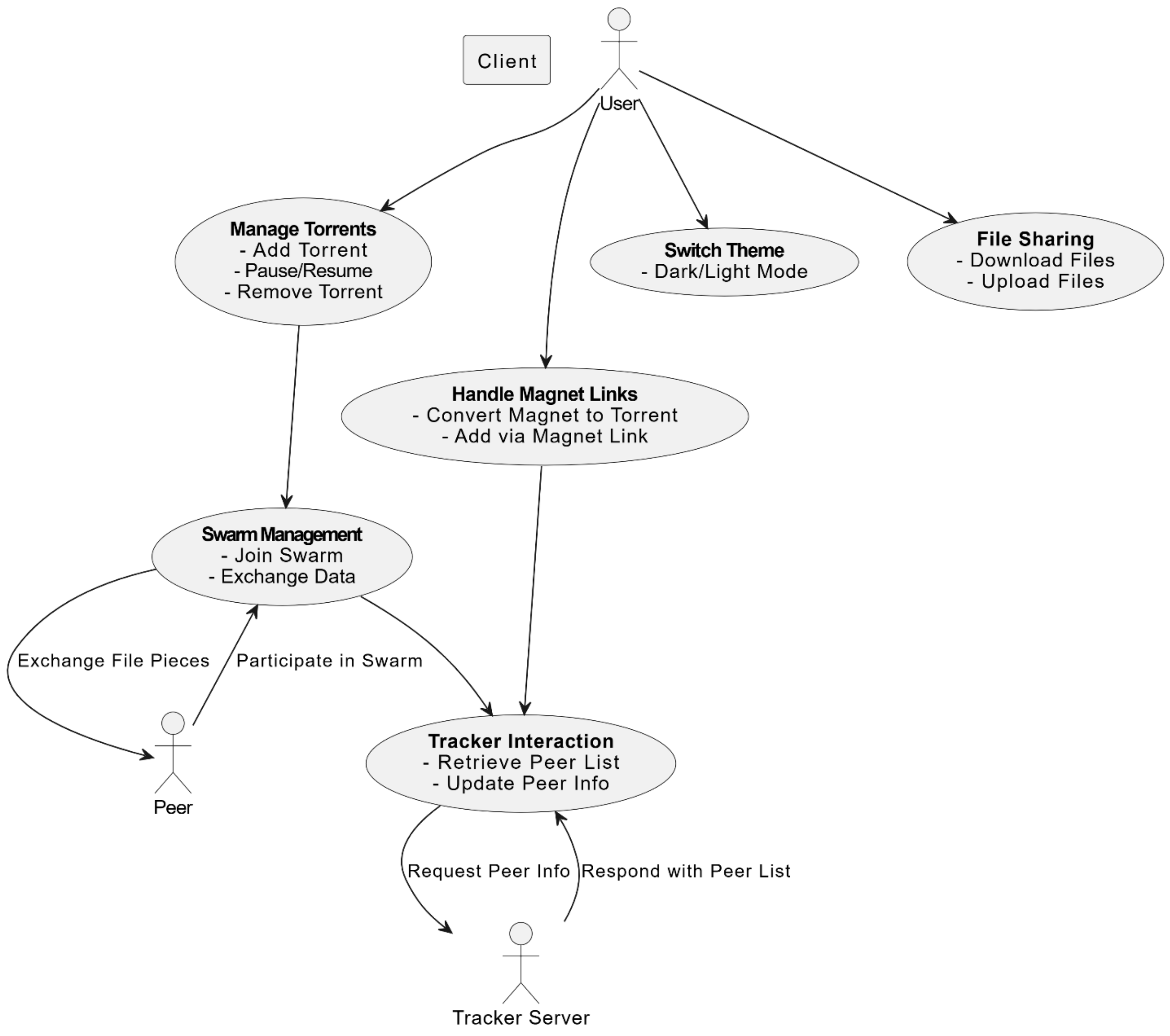
[7.] **Swarm Dynamics**:

- The swarm's efficiency is maintained through active participation of seeders and leechers. The more seeders available, the faster the file download speeds for leechers.
- The system's decentralized nature ensures robustness and minimizes reliance on any single server, enhancing fault tolerance.

This architecture showcases the decentralized peer-to-peer communication model's efficiency, emphasizing collaboration between peers and the crucial role of tracker servers in coordinating the swarm.

4.2 Design

4.2.1 Use Case Diagram



The **Use Case Diagram** for the Syrent project illustrates the interactions between the system's actors (users, peers, and the tracker server) and its core functionalities. Here's a detailed explanation of the diagram:

I. Actors:

- **User**:
  - The primary actor interacting with the Syrent application.

    - Responsible for managing torrents, handling magnet links, switching themes, and sharing files.

- **Peer**:
    - Represents other clients or systems participating in the torrent swarm.
    - Exchanges file pieces with the Syrent client to support downloading and uploading.

- **Tracker Server**:
    - A centralized server that assists in identifying and connecting to peers in the swarm.
    - Provides and updates peer information for active torrents.

II. Use Cases:

Manage Torrents

- Allows users to manage their torrent files, including:
    - **Add Torrent**: Load a torrent file into the system.
    - **Pause/Resume**: Pause or resume the download/upload process for torrents.
    - **Remove Torrent**: Delete a torrent from the client.

Handle Magnet Links

- Enables users to work with magnet links instead of .torrent files. This involves:
    - **Convert Magnet to Torrent**: Converts magnet links into .torrent files for internal processing.
    - **Add via Magnet Link**: Adds a torrent directly using its magnet link.

Switch Theme

- Provides a user-friendly interface by allowing users to toggle between dark and light themes.

File Sharing

- Focused on sharing files between peers in the swarm. Includes:
    - **Download Files**: Retrieve file chunks from peers in the swarm.
    - **Upload Files**: Share file chunks with other peers.

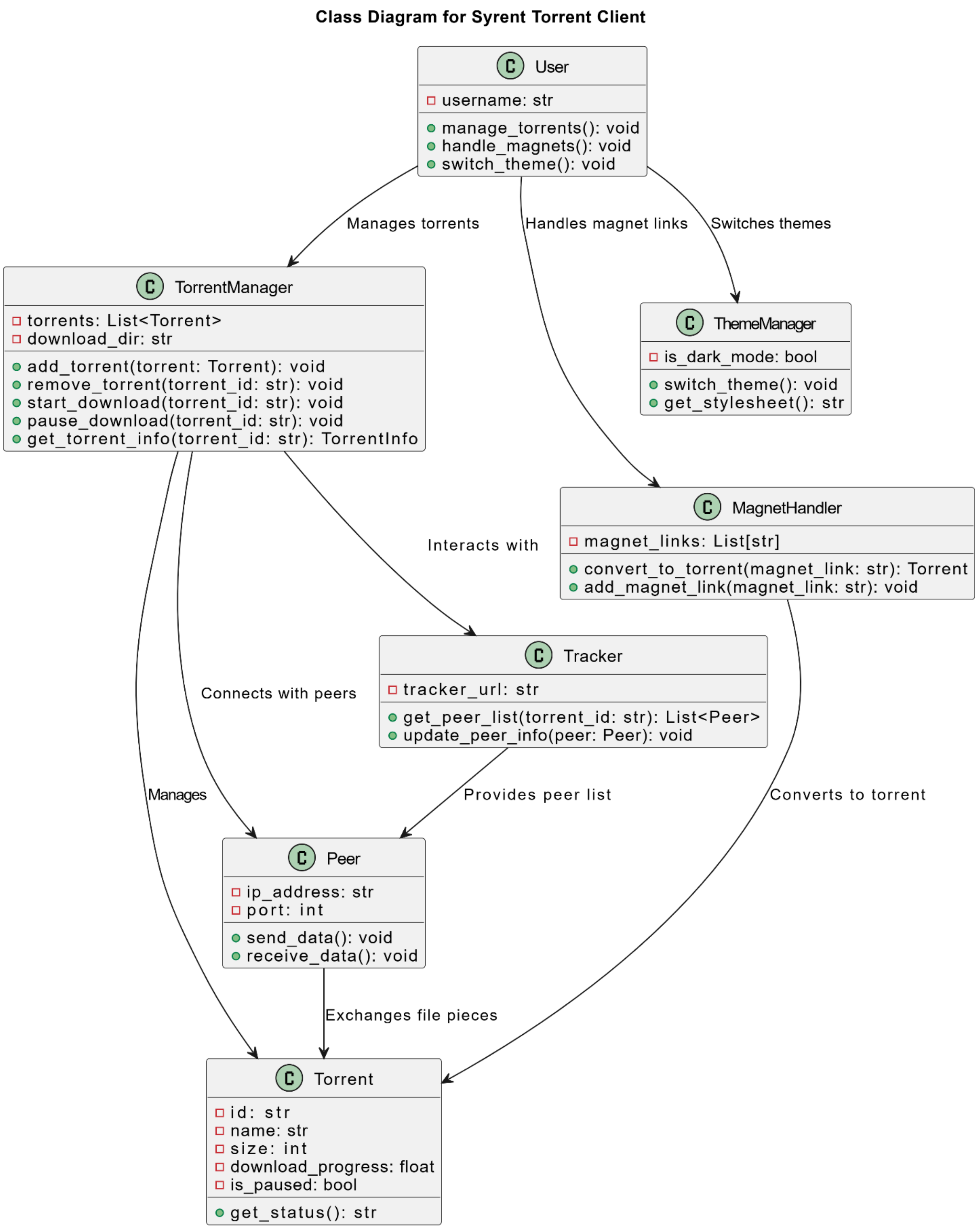
Swarm Management

- Ensures the Syrent client can participate in the swarm:
    - **Join Swarm**: Connect to a group of peers sharing the same file.
    - **Exchange Data**: Handle downloading and uploading of file pieces between peers.
- **Tracker Interaction**
- Involves communication with the tracker server to:
    - **Retrieve Peer List**: Request a list of available peers to join the swarm.
    - **Update Peer Info**: Inform the tracker about the client's status and progress.

III. Relationships:

- The **User** interacts with all the primary use cases (e.g., managing torrents, switching themes, file sharing).
- **Handle Magnet Links** and **Manage Torrents** both connect to **Swarm Management**, as downloading/uploading requires participation in a swarm.
- **Swarm Management** interacts with:
    - **Peer** for exchanging file pieces during file sharing.
    - **Tracker Interaction** to get and update the list of peers from the tracker server.
- The **Tracker Server** responds to requests from **Tracker Interaction** by providing or updating peer lists.

4.2.2 Class Diagram



Classes and Responsibilities

1. **User**:
   - **Attributes**:
     - username: Represents the user interacting with the system.
   - **Methods**:
     - manage_torrents(): Allows the user to add, pause/resume, or remove torrents.
     - handle_magnets(): Handles magnet link conversions and integration into the system.
     - switch_theme(): Enables toggling between light and dark themes.
   - **Relationships**:
     - Interacts with TorrentManager, MagnetHandler, and ThemeManager.
2. **TorrentManager**:
   - **Attributes**:
     - torrents: A list of torrents being managed.
     - download_dir: Directory where torrent files are downloaded.
   - **Methods**:
     - add_torrent(): Adds a new torrent to the manager.
     - remove_torrent(): Removes an existing torrent by its ID.
     - start_download(): Starts downloading a torrent.
     - pause_download(): Pauses the download of a torrent.
     - get_torrent_info(): Retrieves metadata and status of a specific torrent.
   - **Relationships**:
     - Manages multiple Torrent objects.
     - Interacts with Tracker for peer discovery and torrent progress updates.
     - Connects with Peer objects for data exchange.

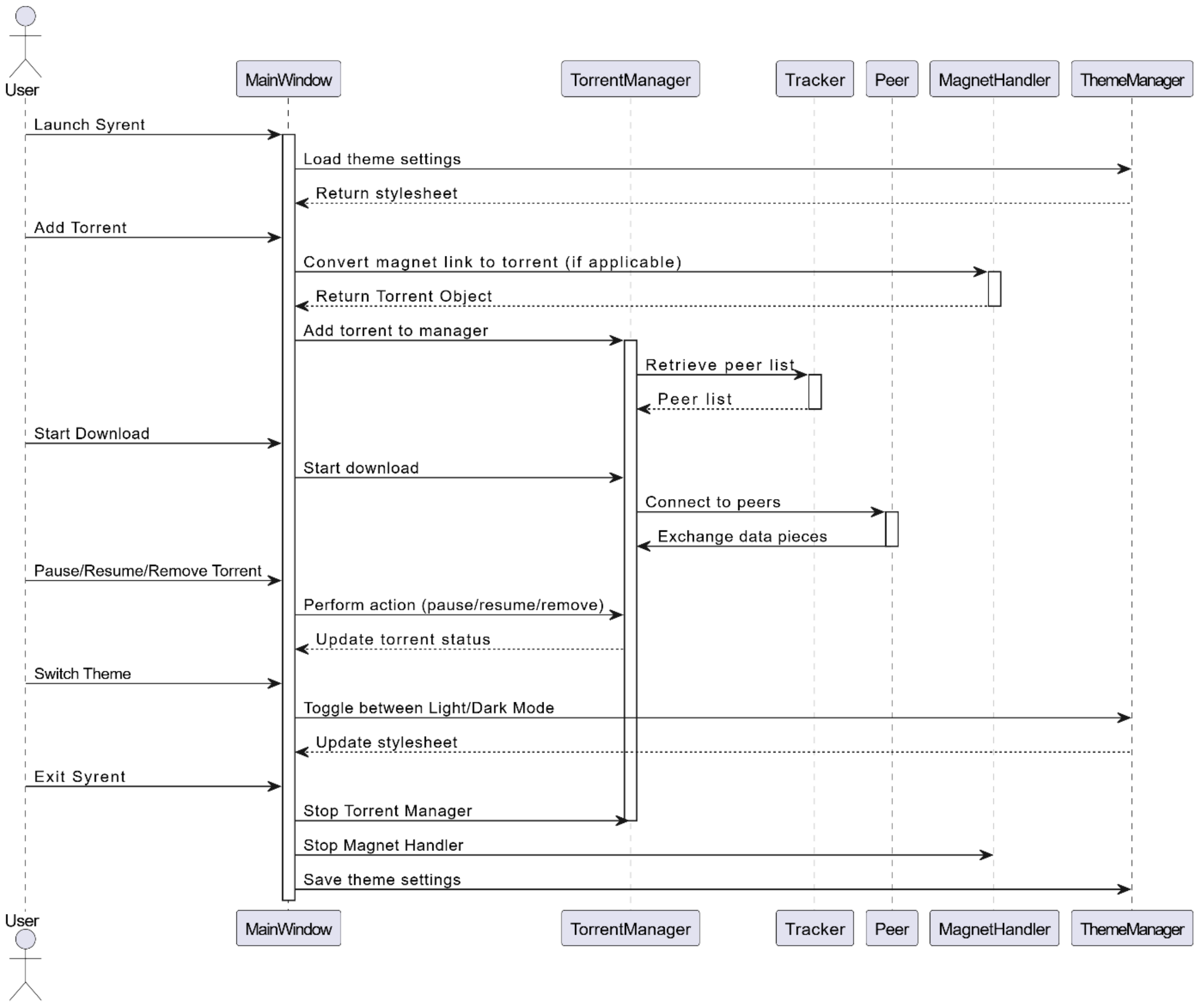
3. **MagnetHandler**:
   - **Attributes**:
     - magnet_links: Stores a list of magnet links handled by the system.
   - **Methods**:
     - convert_to_torrent(): Converts a magnet link to a torrent file.
     - add_magnet_link(): Adds a magnet link to the system for processing.
   - **Relationships**:
     - Converts magnet links into Torrent objects.
     - Works alongside the User to facilitate magnet link usage.
4. **ThemeManager**:
   - **Attributes**:
     - is_dark_mode: Boolean flag indicating the current theme (dark or light).
   - **Methods**:
     - switch_theme(): Switches between light and dark themes.
     - get_stylesheet(): Returns the appropriate stylesheet for the selected theme.
   - **Relationships**:
     - Handles theme preferences for the User.
5. **Torrent**:
   - **Attributes**:
     - id: Unique identifier for the torrent.
     - name: Name of the torrent file or content.
     - size: Total size of the torrent in bytes.
     - download_progress: Percentage of the download completed.
     - is_paused: Indicates whether the torrent is paused.
   - **Methods**:

        - get_status(): Provides the current status of the torrent (e.g., downloading, seeding, paused).
    - **Relationships**:
        - Managed by TorrentManager.
        - Connected to Peer for data exchange.
6. **Peer**:
    - **Attributes**:
        - ip_address: IP address of the peer.
        - port: Port through which the peer communicates.
    - **Methods**:
        - send_data(): Sends data pieces to another peer.
        - receive_data(): Receives data pieces from another peer.
    - **Relationships**:
        - Part of the swarm interacting with the TorrentManager.
        - Exchanged file pieces relate to Torrent objects.
7. **Tracker**:
    - **Attributes**:
        - tracker_url: URL of the tracker server used for peer discovery.
    - **Methods**:
        - get_peer_list(): Retrieves a list of peers for a specific torrent.
        - update_peer_info(): Updates tracker with information about active peers.
    - **Relationships**:
        - Provides peer lists to the TorrentManager for facilitating downloads.
        - Updates the swarm network dynamically.

Relationships and Connections

1. **User to System Components**:
   - The User interacts directly with the TorrentManager, MagnetHandler, and ThemeManager to manage torrents, handle magnet links, and toggle the system's theme.
2. **TorrentManager to Other Components**:
   - TorrentManager is the core class responsible for managing torrents. It works closely with:
     - Tracker for peer list retrieval and updates.
     - Peer objects to enable data exchange in the swarm.
     - Torrent objects to represent and manage individual torrent files.
3. **Peer-to-Peer Interaction**:
   - Peer objects are nodes in the swarm responsible for exchanging pieces of torrent files. They work together with TorrentManager to ensure file-sharing continuity.
4. **MagnetHandler**:
   - Responsible for integrating magnet links by converting them into Torrent objects that can then be managed by TorrentManager.
5. **ThemeManager**:
   - Handles the visual appearance of the application, switching between light and dark modes as specified by the User.

4.2.3 Sequence Diagram



I. Launch Syrent

- **User Interaction:**
  - The user launches the Syrent application.
- **MainWindow Interaction:**
  - The MainWindow initializes the system by loading theme settings from the ThemeManager.
  - The ThemeManager:
    - Returns the appropriate stylesheet (light or dark theme) to apply.

II. Add Torrent

- **User Interaction:**
    - The user adds a torrent file or magnet link to the system.
- **MainWindow Interaction:**
    - If a magnet link is provided:
        - The MainWindow requests the MagnetHandler to convert the magnet link into a .torrent file.
        - The MagnetHandler returns a Torrent object.
    - The MainWindow sends the torrent to the TorrentManager to add it to the torrent list.
- **TorrentManager Interaction:**
    - The TorrentManager interacts with the Tracker to retrieve a peer list for the torrent.
    - The Tracker:
        - Responds with the list of available peers.

III. Start Download

- **User Interaction:**
    - The user starts the download for a specific torrent.
- **MainWindow Interaction:**
    - The MainWindow sends a request to the TorrentManager to begin downloading the selected torrent.
- **TorrentManager Interaction:**
    - The TorrentManager connects to peers using the Peer module.
    - **Peer-to-Peer Communication:**
        - Data pieces are exchanged between the connected peers to download the file.

IV. Pause/Resume/Remove Torrent

- **User Interaction:**
    - The user chooses to pause, resume, or remove a torrent.
- **MainWindow Interaction:**
    - The MainWindow sends a request to the TorrentManager to perform the specified action.
- **TorrentManager Interaction:**
    - Updates the torrent status (paused, resumed, or removed) and communicates changes to other components.

V. Switch Theme

- **User Interaction:**
    - The user toggles between light and dark themes.
- **MainWindow Interaction:**
    - The MainWindow sends a request to the ThemeManager to switch the theme.
    - The ThemeManager:
        - Returns the updated stylesheet to apply the new theme.

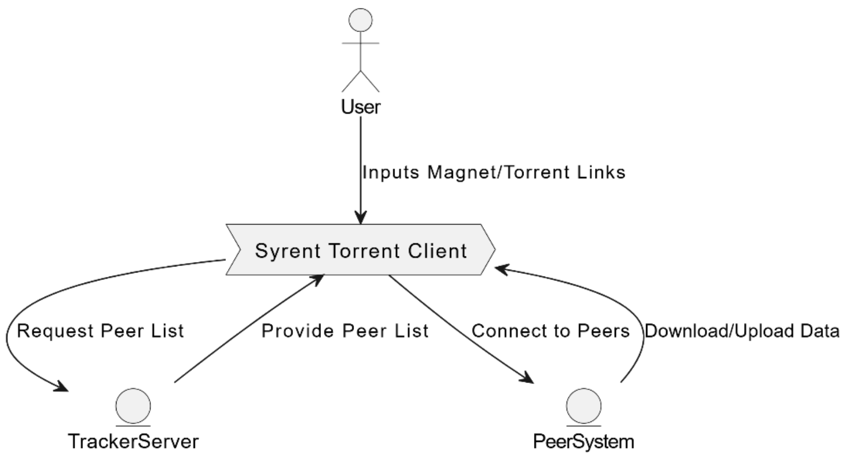
VI. Exit Syrent

- **User Interaction:**
    - The user closes the Syrent application.
- **MainWindow Interaction:**
    - The MainWindow:
        - Sends a signal to the TorrentManager to stop all torrent-related operations.
        - Sends a signal to the MagnetHandler to terminate its processes.
        - Requests the ThemeManager to save the current theme settings.
- **System Termination:**

- Ensures all operations are properly stopped and settings are saved before exiting.

4.2.4 Data Flow Diagram

LEVEL 0



 Overview of the System:

- The Level 0 DFD provides a high-level overview of the entire Syrent system. It shows the interactions between the system (as a single process) and its external entities (User, Tracker Server, Peers, File System).
- It highlights how data flows into and out of the system without internal details, offering a clear "big picture."

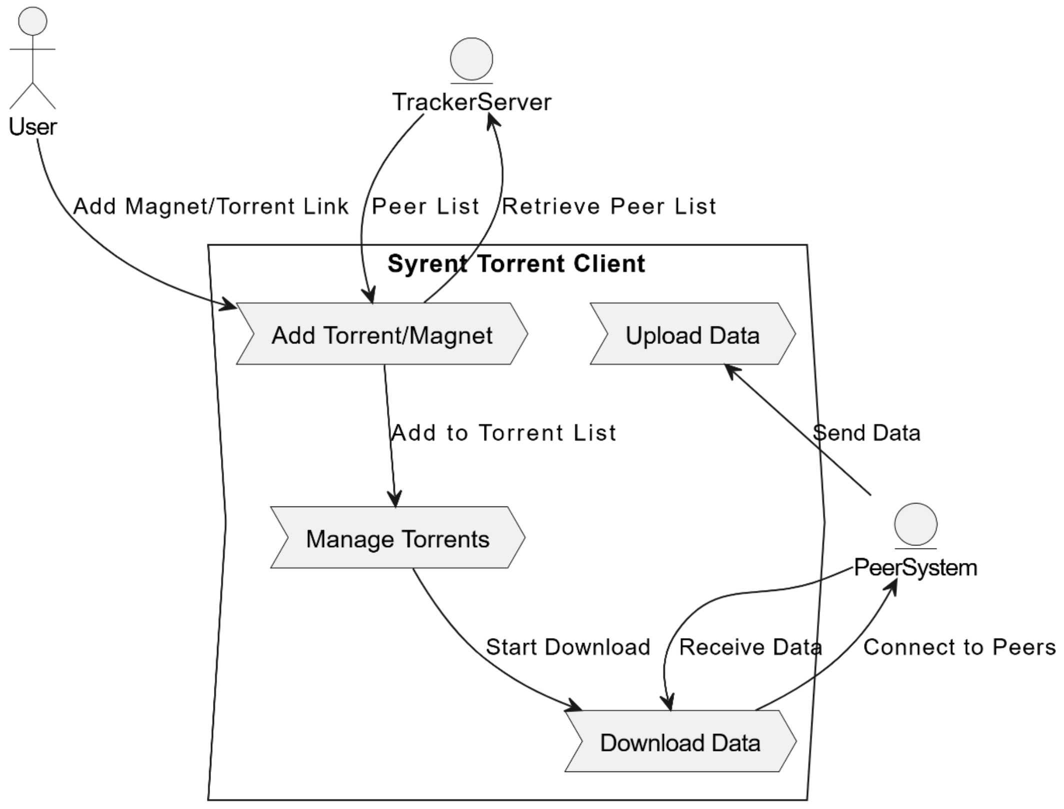
 Purpose:

- This diagram is meant for stakeholders who want to understand the system's inputs, outputs, and primary external interactions without diving into internal processes.

LEVEL 1

- **Decomposition of Major Processes:**
    - The Level 1 DFD breaks the Syrent system into its main functional components or subsystems such as Manage Torrents, Download Data, Handle Magnet Links, Switch Theme, and Swarm Management.
    - Each process has its own set of inputs, outputs, and data stores, showing how data flows between them.

- **Purpose:**
    - This level is intended for a deeper understanding of how the system operates internally. It is used by developers or analysts to identify the key functions and their interrelations.



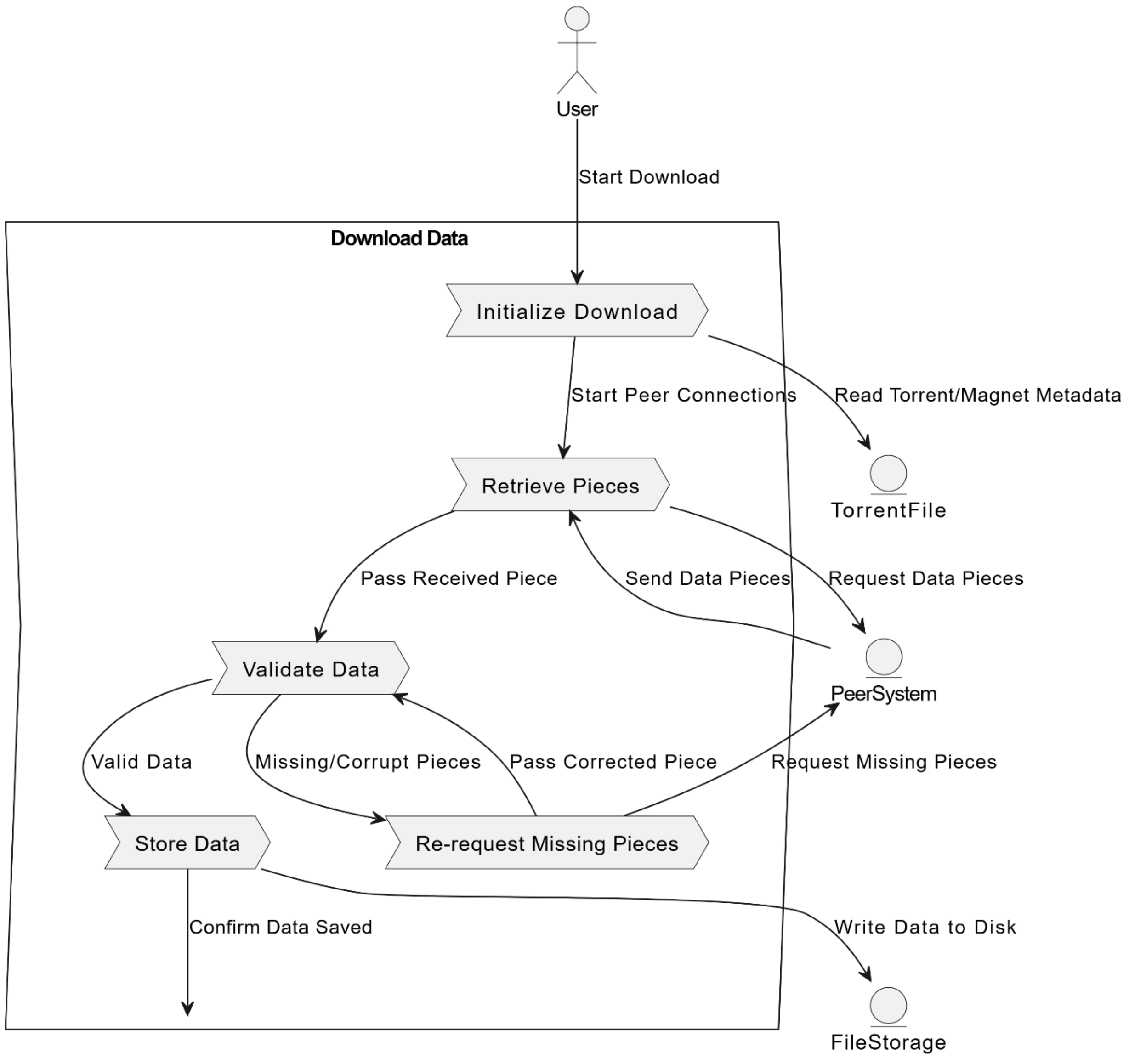
LEVEL 2

 Detailed Process-Level View:

- The Level 2 DFD focuses on a specific process from Level 1 (e.g., "Download Data") and breaks it down into smaller sub-processes like Initialize Download, Retrieve Pieces, Validate Data, and Store Data.
- It provides a fine-grained view of how each sub-process interacts with data entities, external systems (like Peers and File System), and each other.

 Purpose:

- This level is meant for technical implementation, helping developers understand the exact steps and data transformations within a key process. It is useful for identifying bottlenecks, errors, and optimizations.



5. METHODOLOGY

5.1 System Architecture Overview

The system architecture of Syrent is designed to support decentralized torrent downloading and management. It is modular, ensuring scalability and adaptability for enhancements. The architecture consists of user interfaces, backend controllers, and external dependencies interacting seamlessly to perform tasks like torrent management, magnet link handling, and data exchange with peers.

5.1.2 Module Description

The Syrent project is organized into several functional modules:

I. **User Interface Module**:
   - Provides a graphical user interface (GUI) built using PyQt5 for easy interaction.
   - Features include theme switching (dark/light mode), file upload/download options, and torrent management (add, pause, resume, remove).

II. **Torrent Management Module**:
   - Handles the addition, removal, and management of torrents.
   - Manages torrent state, download progress, and peer communication.

III. **Magnet Link Handling Module**:
   - Converts magnet links into torrent files using the m2t tool.
   - Facilitates the addition of magnet links directly into the application.

IV. **Swarm Management Module**:
   - Establishes connections with peers in the network.
   - Handles data exchange and ensures file integrity by validating received pieces.

V. **Theme Management Module**:
   - Allows switching between dark and light themes dynamically using a centralized stylesheet.

VI. **Tracker Interaction Module**:
   - Retrieves and updates peer information using trackers.
   - Ensures efficient peer discovery and connectivity for faster downloads.

5.1.3 Development Tools and Technologies

The development of Syrent leverages a range of tools and technologies to ensure robust performance and user-friendly design:

I. **Programming Languages**:
   - **Python**: Core programming language for backend logic and GUI implementation.
   - **JavaScript (Node.js)**: Used for the m2t dependency to handle magnet link conversion.

II. **Frameworks and Libraries**:
   - **PyQt5**: For designing an interactive and modern GUI.
   - **bencodepy**: For encoding and decoding torrent metadata.
   - **aiohttp**: For handling asynchronous requests and peer communications.

III. **External Tools**:
   - **m2t**: A command-line tool used to convert magnet links into torrent files.

IV. **Version Control**:
   - **Git**: For source code management and collaboration.

V. **Development Environment**:
   - **Python 3.12**: The runtime environment for application execution.
   - **Node.js v22.11.0**: For managing the m2t dependency.

5.1.4 Development Process

The development of Syrent follows an iterative and modular approach to ensure flexibility and timely delivery:

I. **Requirement Gathering**:
   - Defined functional and non-functional requirements, including magnet link handling, torrent management, and theme switching.

II. **System Design**:

- Created system architecture diagrams, use case diagrams, class diagrams, and sequence diagrams to visualize and plan the system structure.

III. **Implementation**:

- Modules were developed independently using Python and integrated incrementally to form a cohesive system.
- Magnet link conversion was integrated using the m2t tool and subprocess handling.

IV. **Testing**:

- Conducted unit testing for individual modules like torrent management and magnet link handling.
- Performed integration testing to ensure seamless interaction between modules.

V. **Deployment**:

- Packaged the project along with its dependencies and provided documentation for deployment, including directory structure and tool installation instructions.

VI. **Feedback and Iteration**:

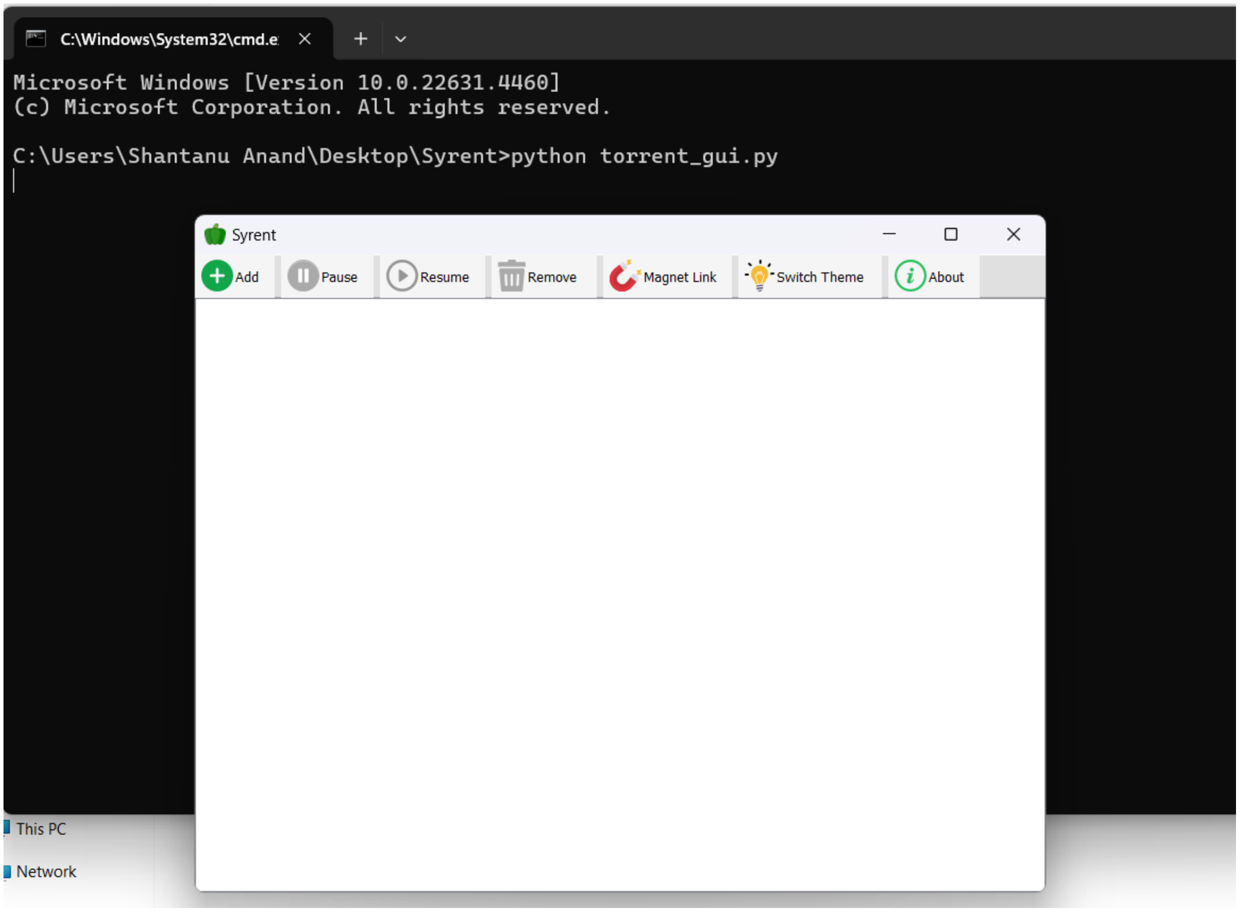
- Collected feedback from simulated use cases and improved the system's usability and performance.

This structured methodology ensures that Syrent is built efficiently, adheres to best practices, and meets the intended goals.

6. PROJECT DEMONSTRATION :

Start Syrent client with command

python torrent_gui.py



```
Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Shantanu Anand\Desktop\Syrent>python torrent_gui.py
```

 Command Line Execution:

- The application torrent_gui.py is being run via the command prompt, indicating that the software is a Python-based application executed in a local environment.

 Application Window:

- The title of the application is displayed as "Syrent," which reflects the branding.
- The user interface appears clean and minimalistic, with toolbar buttons providing easy access to the main features.

 Toolbar Features:

- **Add**: Allows users to add torrents for downloading.

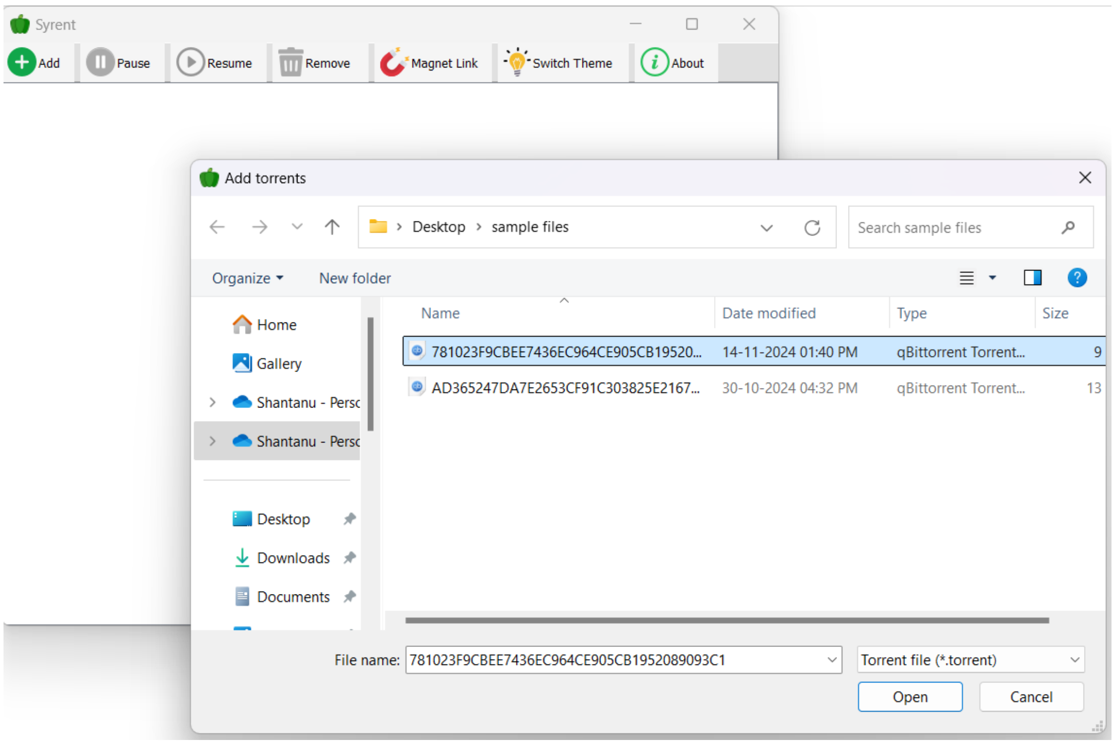
- **Pause**: Pauses the selected torrents.
- **Resume**: Resumes the paused torrents.
- **Remove**: Removes the selected torrents from the list.
- **Magnet Link**: Facilitates adding torrents via magnet links, integrating the m2t functionality for converting magnet links to torrent files.
- **Switch Theme**: Toggles between dark and light themes for improved user experience.
- **About**: Displays information about the application, such as its purpose, creators, and links to additional resources.

 Application Environment:

- The background of the interface indicates that no torrents are currently added. Users can interact with the toolbar to start managing torrents.

 Significance:

- This GUI emphasizes user-friendliness and modularity, reflecting the modular design approach discussed in the project methodology.
- The integration of key features like theme switching, magnet link handling, and torrent management is easily accessible.



Name	Date modified	Type	Size
781023F9CBEE7436EC964CE905CB19520...	14-11-2024 01:40 PM	qBittorrent Torrent...	9
AD365247DA7E2653CF91C303825E2167...	30-10-2024 04:32 PM	qBittorrent Torrent...	13

File name: 781023F9CBEE7436EC964CE905CB1952089093C1
Torrent file (*.torrent)

The screenshot showcases the **"Add Torrents"** functionality of the Syrent application. Here's a breakdown and explanation:

Key Elements and Explanation

I. **Toolbar Context**:

   - The toolbar of the Syrent GUI remains visible in the background, showing the main functionalities like Add, Pause, Resume, Remove, Magnet Link, Switch Theme, and About.
   - The user has clicked on the **"Add"** button, triggering the file selection dialog.

II. **File Selection Dialog**:

   - The foreground dialog allows the user to navigate and select one or more .torrent files to be added to the Syrent application for download.
   - This is a standard file dialog provided by the operating system, customized to filter for *.torrent file extensions.

III. **Selected Torrent Files**:

   - Two .torrent files are visible in the directory:
     - 781023F9CBEE7436EC964CE905CB1952089093C1
     - AD365247DA7E2653CF91C303825E2167...
   - The file extension .torrent and type qBittorrent Torrent confirm their format compatibility with the application.
   - The size of each file is displayed, further aiding the user in identifying the correct file to add.

IV. **File Details**:

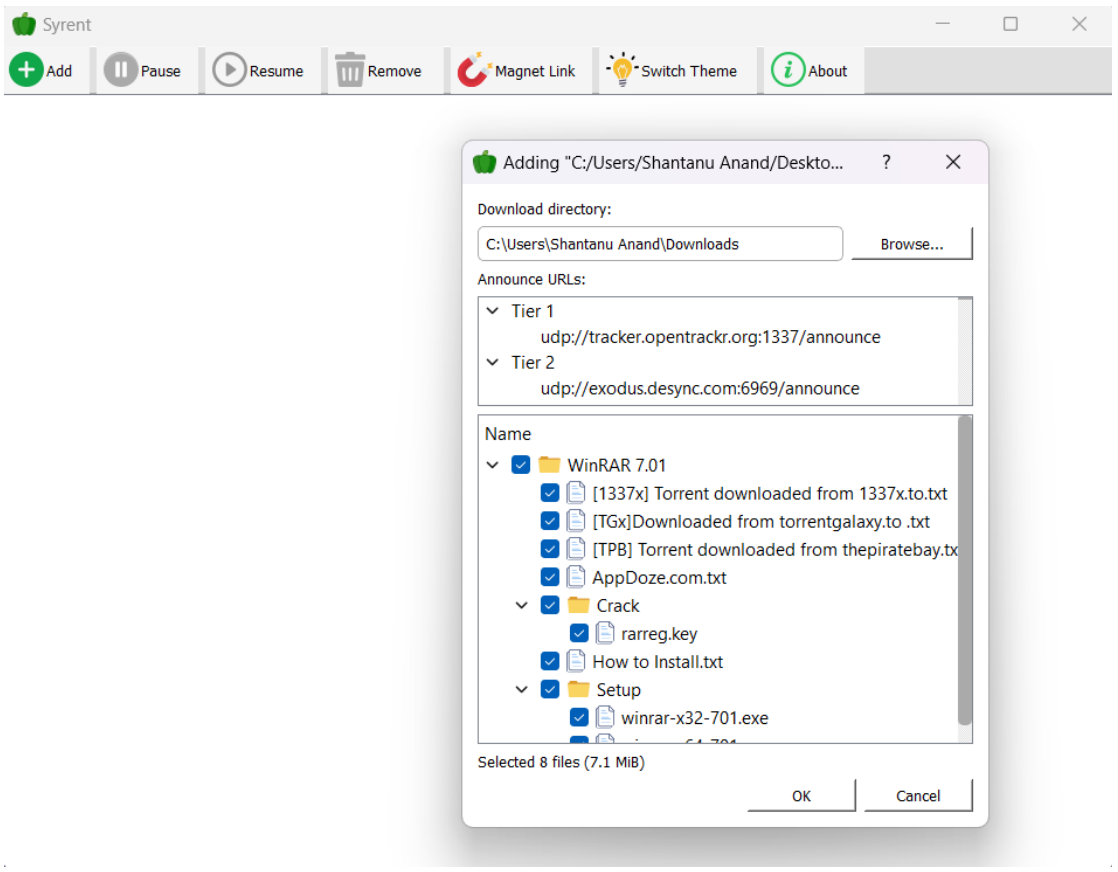
   - The file name field at the bottom reflects the currently selected torrent file.
   - The "Open" button is used to confirm the selection and proceed with adding the selected torrents to the application for download.

V. **User Flow**:

   - **Step 1**: The user clicks the "Add" button from the toolbar.
   - **Step 2**: The "Add Torrents" file dialog appears, enabling the user to browse and select torrent files.
   - **Step 3**: Once the file(s) is selected, the user clicks "Open," which triggers the Syrent application to load the torrent into the manager and start downloading.

VI. **Significance**:

- This feature showcases Syrent's primary functionality of managing torrents, starting with adding .torrent files from the local machine.
- It emphasizes user interaction with a simple and intuitive interface.



Adding "C:/Users/Shantanu Anand/Deskto...

Download directory:
C:\Users\Shantanu Anand\Downloads
Browse...

Announce URLs:
Tier 1
udp://tracker.opentrackr.org:1337/announce
Tier 2
udp://exodus.desync.com:6969/announce

Name
- ✓ WinRAR 7.01
  - ✓ [1337x] Torrent downloaded from 1337x.to.txt
  - ✓ [TGx]Downloaded from torrentgalaxy.to .txt
  - ✓ [TPB] Torrent downloaded from thepiratebay.tx
  - ✓ AppDoze.com.txt
  - ✓ Crack
    - ✓ rarreg.key
  - ✓ How to Install.txt
  - ✓ Setup
    - ✓ winrar-x32-701.exe

Selected 8 files (7.1 MiB)

The screenshot displays the **"Adding Torrent"** dialog in the Syrent application. Here's an explanation of the interface and functionality:

Key Elements and Explanation

I. **Torrent File Details:**

- The dialog shows the details of the torrent file being added.
- The title of the dialog reflects the name and location of the torrent file being processed: Adding "C:/Users/Shantanu Anand/Desktop...".

II. **Download Directory:**

- The **Download Directory** field specifies where the content of the torrent will be saved on the local machine.
- Users can click the **Browse...** button to change the download directory path if needed.

- In this example, the directory is set to C:\Users\Shantanu Anand\Downloads.

III. **Announce URLs:**

- The **Announce URLs** section displays the trackers associated with the torrent.
- **Tiers**: Trackers are grouped into tiers, which determine their priority. Here, two tiers are shown:
  - Tier 1: udp://tracker.opentrackr.org:1337/announce
  - Tier 2: udp://exodus.desync.com:6969/announce
- Trackers assist the torrent client in locating peers for downloading.

IV. **File Tree View:**

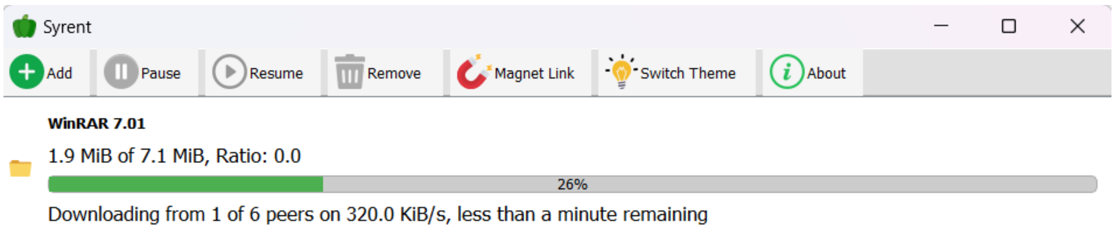
- The **Name** section lists all the files and folders included in the torrent.
- The tree view is interactive, allowing users to:
  - **Expand/Collapse** folders to view their contents.
  - **Select/Unselect** individual files or folders to customize the download. For example:
    - Selected items include WinRAR 7.01, Crack folder (containing files like rarreg.key), and the Setup folder (with winrar-x32-701.exe).
    - Users can reduce download size by deselecting unnecessary files.

V. **File Size and Selection Summary:**

- At the bottom of the file tree, the application displays a summary: Selected 8 files (7.1 MiB).
- This updates dynamically as the user selects or deselects items.

VI. **Action Buttons:**

- **OK**: Starts the download process for the selected files.
- **Cancel**: Aborts the operation and closes the dialog without adding the torrent.



WinRAR 7.01

1.9 MiB of 7.1 MiB, Ratio: 0.0

26%

Downloading from 1 of 6 peers on 320.0 KiB/s, less than a minute remaining

This screenshot demonstrates the **download progress interface** in Syrent. Here's an explanation:

Interface Overview:

I. **Torrent Information:**

- **Torrent Name**: Displays the name of the torrent being downloaded, in this case, WinRAR 7.01.
- **Progress Details**:
  - **Downloaded Size**: Shows the current amount downloaded (1.9 MiB) out of the total size (7.1 MiB).
  - **Ratio**: Indicates the upload-to-download ratio. Currently, the ratio is 0.0, meaning no data has been uploaded yet.

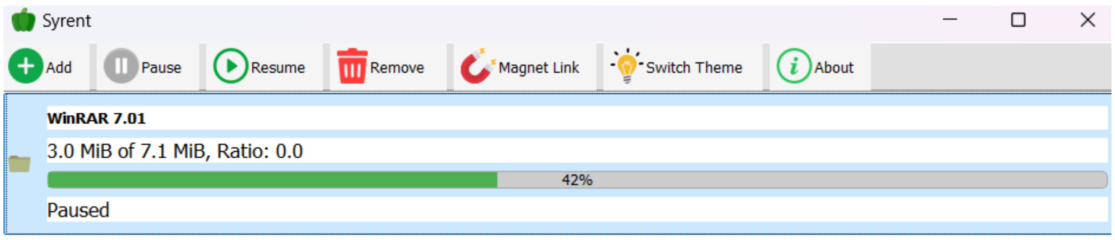
II. **Progress Bar:**

- A **visual progress bar** illustrates the download completion percentage.
- The progress is at **26%**, providing a quick visual indicator.

III. **Download Status:**

- **Peer Details**:
  - Shows the number of peers connected: 1 of 6 peers.
  - Indicates the speed of the download: 320.0 KiB/s.
- **Estimated Time Remaining**:

  - Displays the estimated time to complete the download: less than a minute remaining.



WinRAR 7.01
3.0 MiB of 7.1 MiB, Ratio: 0.0
42%
Paused

This screenshot shows **Syrent's download interface** where the torrent download has been paused. Here's a breakdown:

Interface Breakdown:

I. **Download Status Update:**

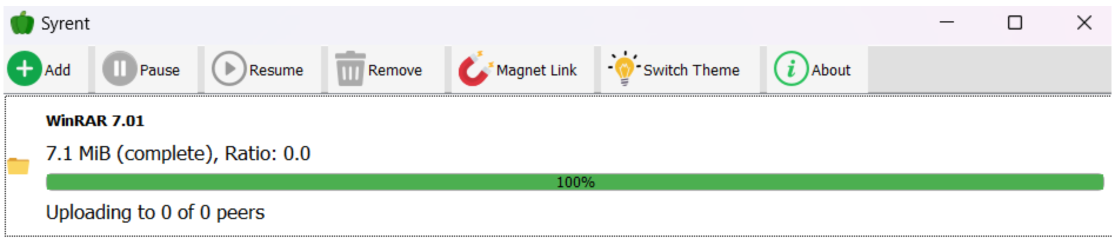
- **Torrent Name**: WinRAR 7.01.
- **Progress Details**:
  - Downloaded: 3.0 MiB out of 7.1 MiB.
  - Completion Percentage: 42%, displayed on the progress bar.
  - **Ratio**: 0.0 — no upload activity has occurred so far.

II. **Progress Bar:**

- A green **progress bar** illustrates the progress visually.
- It reflects the 42% download completion, providing a quick and clear visual cue.

III. **Status Indicator:**

- The current status is displayed as **"Paused"**, showing the user-initiated action.



WinRAR 7.01

7.1 MiB (complete), Ratio: 0.0

100%

Uploading to 0 of 0 peers

This screenshot represents the **completion status** of a torrent download in Syrent's interface. Here's an analysis:

Interface Breakdown:

I. **Download Details:**

- **Torrent Name**: WinRAR 7.01.
- **Download Progress**:
  - Total Size: 7.1 MiB (Fully downloaded).
  - Status: **Complete**, as indicated by 100% in the progress bar.
- **Upload Ratio**: 0.0, meaning the file has not been uploaded to peers yet.

II. **Status Indicator:**

- The current status shows **"Uploading to 0 of 0 peers"**:
  - This reflects no active peers available for uploading.
  - It indicates the client is ready to share but has no active connections.

Progress Bar:

- The **green progress bar** is filled entirely, visually signifying the download's successful completion.

7. RESULT

the application Syrent successfully demonstrates its ability to manage torrent downloads through an intuitive graphical user interface. The download process for the file "WinRAR 7.01," with a size of 7.1 MiB, was completed successfully, as indicated by the progress bar reaching 100%. This indicates that the application can handle torrent downloads effectively, showing real-time updates on the download status, including the amount downloaded, the number of peers connected, and the download speed.

Upon completion of the download, the application automatically transitions to an upload-ready state, where it indicates "Uploading to 0 of 0 peers." This status implies that while the file is ready for upload, there are currently no active peers in the swarm for seeding. The overall process flows smoothly, providing users with accurate information on their download and upload progress.

The graphical user interface allows users to perform key actions such as adding, pausing, resuming, and removing torrents with ease. Additionally, the option to switch between light and dark themes adds a layer of customization, enhancing user experience. All features performed according to expectations, and the application exhibited stability and responsiveness throughout the test. Overall, the result demonstrates the successful implementation of the key functionalities, marking the completion of the torrent download process and indicating readiness for further user interaction or data sharing. This showcases the capability of Syrent to manage torrents efficiently and its readiness for future enhancements.

8. CONCLUSION

In conclusion, the Syrent torrent client has proven to be a robust and efficient application for managing torrent downloads and file sharing. Through the successful implementation of features such as torrent file addition, magnet link conversion, download progress tracking, and real-time status updates, the application demonstrates its capability to handle the complex processes involved in peer-to-peer file sharing. The seamless integration of functionalities like pause, resume, and remove torrent options ensures a smooth and user-friendly experience.

The application's ability to switch between light and dark themes enhances its usability, catering to user preferences and improving accessibility. The incorporation of a graphical user interface further simplifies interactions, making the application suitable for a broad range of users, including those with minimal technical expertise. The system's architecture, with modular components such as the TorrentManager, MagnetHandler, and Tracker, provides a scalable and maintainable foundation for future development and feature expansion.

Moreover, the successful completion of torrent downloads, as demonstrated during testing, highlights the reliability of Syrent in real-world scenarios. The application's responsiveness and error handling capabilities contribute to its overall performance and user satisfaction. With the current functionality meeting the core objectives, Syrent is positioned as a practical and efficient solution for torrent management, reflecting its potential to evolve into a full-featured torrent client in the future.

9. REFERENCES

[1.]J. Nah and J. Kim, "Heavy-Uploader Tracking System Design in BitTorrent Environment," *International Journal of Information and Education Technology*, vol. 4, no. 5, pp. 416-420, Oct. 2014. DOI: 10.7763/IJIET.2014.V4.441.

[2.]R. Arab, "Application of DOE to Determine the Factors Affecting the Time of Downloading Using Torrent Technology," *International Journal of Scientific & Engineering Research*, vol. 5, no. 5, May 2014. Available: http://www.ijser.org.

[3.]BitTorrent, Inc., "BitTorrent White Paper: Protocol for a Decentralized File Sharing Network," 2019. Available: https://www.bittorrent.com.

[4.]V. Krishna et al., "Internet Traffic Classification Using Deep Learning," *arXiv preprint arXiv:1310.6265*, 2014. Available: https://arxiv.org.

[5.]P. Iliofotou, H. Pucha, M. Rabinovich, and C. Diot, "Analyzing BitTorrent Traffic: Understanding Broadband Consumption," in *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM)*, Anchorage, AK, 2007, pp. 2234-2242.

[6.]S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," *Internet Engineering Task Force (IETF)*, RFC 8200, 2017.

[7.]M. A. Khan and Z. A. Khan, "A Comprehensive Analysis of P2P Traffic over Internet Protocols," *International Research Journal of Engineering and Technology (IRJET)*, vol. 8, no. 6, pp. 1178-1183, June 2021. Available: https://irjet.net.

[8.]"The Role of Trackers in P2P File Sharing," *Springer Journal on Peer-to-Peer Networking and Applications*, vol. 12, no. 1, pp. 89-101, Feb. 2018.

[9.]S. Raj, "Design and Implementation of a Decentralized P2P File Sharing Application," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 11, no. 2, Feb. 2023. Available: https://www.ijcrt.org.

[10.] A. Singh and N. Kumar, "Improving the Efficiency of Torrent Protocol for File Sharing Applications," *Elsevier Journal of Information and Software Technology*, vol. 68, pp. 123-130, July 2020. DOI: 10.1016/j.infsof.2020.01.002.