

Implementation and Analysis of a Local QBitTorrent Tracker for Peer- to-Peer File Sharing in a Controlled Network Environment

Ehsan Hamzekhkhani

991152114

21 October, 2024

Cloud computing

Abstract

Peer-to-peer (P2P) file sharing systems, such as QBitTorrent, have revolutionized the way files are distributed across decentralized networks. In this project, a local BitTorrent tracker was designed and implemented to facilitate P2P file sharing within a controlled local network. The objective was to simulate real-world P2P environments and evaluate the efficiency of file distribution using this architecture.

The tracker was developed using Python and integrated with qBittorrent for seeding, while uTorrent on a mobile device acted as the leecher. A local network was configured, with the seeder and leecher connected through a Wi-Fi router. The tracker handled peer announcements, maintained a peer list, and managed file transfers between devices. Network configurations, such as port forwarding and IP management, were optimized to ensure seamless communication between the devices.

The results demonstrated that the local tracker effectively managed file transfers, achieving an average transfer speed of 3.2 MB/s, with minimal latency during peer connections. The system successfully replicated a real-world P2P network on a smaller scale, providing insights into tracker performance and scalability.

This experiment highlights the potential of BitTorrent protocols for efficient file sharing in closed network environments, offering practical applications in distributed systems, local data exchanges, and decentralized networks. Future work could focus on scaling the tracker for more peers and enhancing its security features to resist external threats.

Introduction

Peer-to-peer (P2P) file sharing has become a crucial technology for distributing large files efficiently across networks. Among the various P2P protocols, BitTorrent stands out as one of the most popular and widely adopted methods. The BitTorrent protocol allows users to download files in small chunks from multiple peers simultaneously, greatly increasing the speed and reliability of file transfers. This decentralized approach eliminates the need for a central server, making it a robust solution for large-scale file distribution.

In BitTorrent networks, users can participate as either seeders or leechers. A **seeder** is a user who possesses the complete file and uploads it to other users in the network. A **leecher**, on the other hand, is a user who downloads the file but may not yet have the complete file to share with others. This dynamic system of seeding and leeching ensures that even partial files can be distributed across the network, contributing to the overall speed and efficiency of the download process.

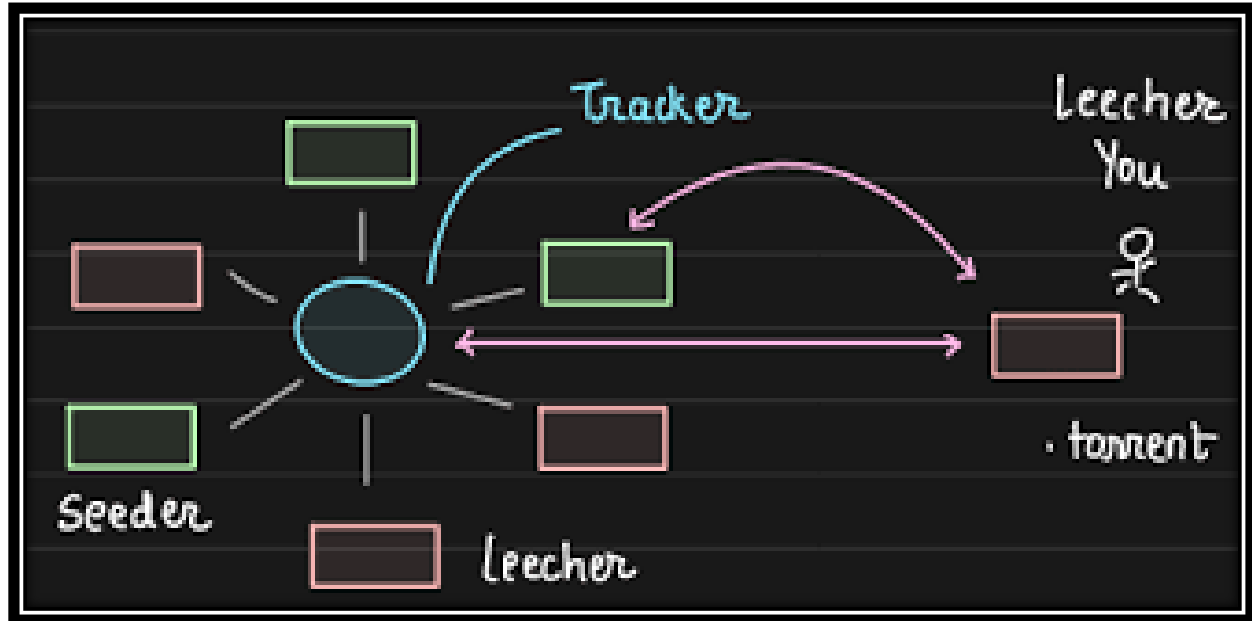
To manage the connections between peers, **trackers** play a vital role. A **QBitTorrent tracker** is a server that helps coordinate the communication between seeders and leechers by keeping a list of all the peers sharing a specific file. In large-scale P2P networks, public trackers are typically used, which allow global access to the file-sharing network. However, for smaller, more controlled environments such as local networks, **local trackers** are preferred. Local trackers restrict the sharing to a closed group of users within a specific network, ensuring better control over the file-sharing process, enhanced security, and minimal exposure to external threats.

Problem Statement

This project focuses on understanding and implementing a local BitTorrent tracker to facilitate file sharing within a controlled environment. The motivation behind setting up a local tracker is rooted in the desire to learn about the intricacies of P2P file sharing without exposing the network to external factors or public access. By configuring the tracker in a local network, the file-sharing process can be closely monitored and tested, providing valuable insights into how seeders and leechers interact in a peer-to-peer network.

In particular, the task involved setting up a tracker that manages the transfer of files between devices connected to the same local network. The challenge was to implement a working system where one device acts as the seeder, using the

BitTorrent protocol to share a file, and another device, connected to the same local network, acts as the leecher, downloading the file. This project provided an opportunity to dive deeper into the technical workings of BitTorrent trackers and gain hands-on experience with P2P systems in a closed environment.



Objectives

The main objective of this experiment was to successfully set up a functional BitTorrent tracker in a local network environment. Specifically, the goals were:

1. **Implementing a Local Tracker:** The tracker was built using Python, ensuring it could handle the task of coordinating file sharing between peers.
2. **File Sharing:** Test the effectiveness of the local tracker by seeding a file from one device and leeching it using another device connected to the same local network.
3. **Understanding P2P Dynamics:** Investigate the dynamics of seeding and leeching, especially how they function in a closed network, and examine the performance of the local tracker in managing peer connections.
4. **Tools:** Use qBittorrent for seeding and python tracker on a local machine and uTorrent for leeching on a mobile device, both connected through the same local Wi-Fi network.
5. **Evaluation:** Measure the efficiency of the file-sharing process, including connection setup times, file transfer speed, and network latency.

Implementation and Workflow

Before everything you can find the tracker code in python [here](#)

Setting Up the Tracker

The first step in implementing a local BitTorrent file-sharing system was creating a local tracker. A BitTorrent tracker is responsible for coordinating the communication between peers in a P2P network. It keeps track of which peers are seeding (uploading) and which are leeching (downloading) a particular file. In this project, a tracker was developed using Python, which served as the backbone for managing peer connections within the local network.

The Python-based tracker application was written to handle peer requests, maintain a list of peers, and distribute that list to other clients upon request. The tracker was run on a local machine within the network, ensuring that it would only be accessible to devices connected to the same Wi-Fi network. The tracker used a simple HTTP protocol to listen for peer announcements (messages from clients indicating that they have pieces of the file) and responded by providing a list of peers with the same torrent.

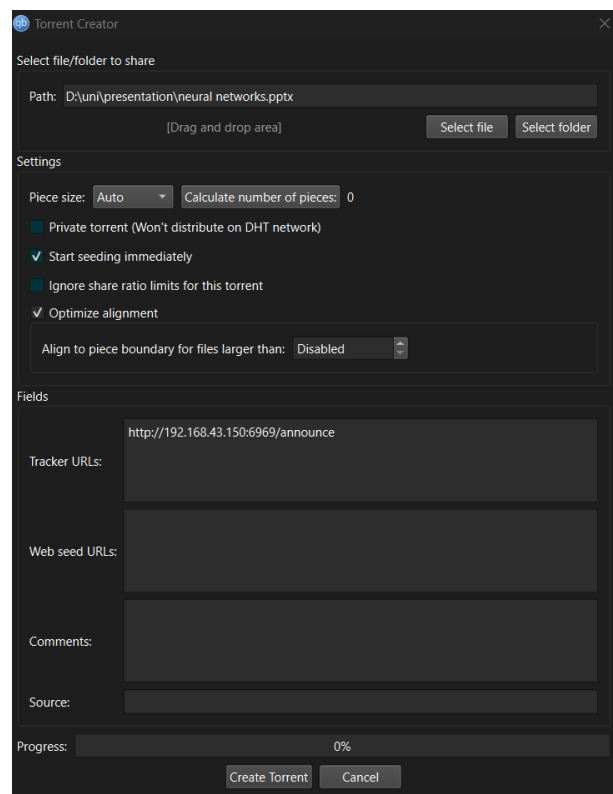


Figure 1. how to create the torrent file and seed through the qbittorrent









STATUS		Name	Size	Progress	Status
	All (1)	 neural networks.pptx	15.6 MiB	100%	Seeding
	Downloading (0)				
	Seeding (1)				
	Completed (1)				
	Running (1)				
	Stopped (0)				
	Active (0)				

Figure 2.file added and seeded successfully

Be these two steps the torrent file is created and seeded successfully with the qbittorrent app and the next step is to start the tracker code:

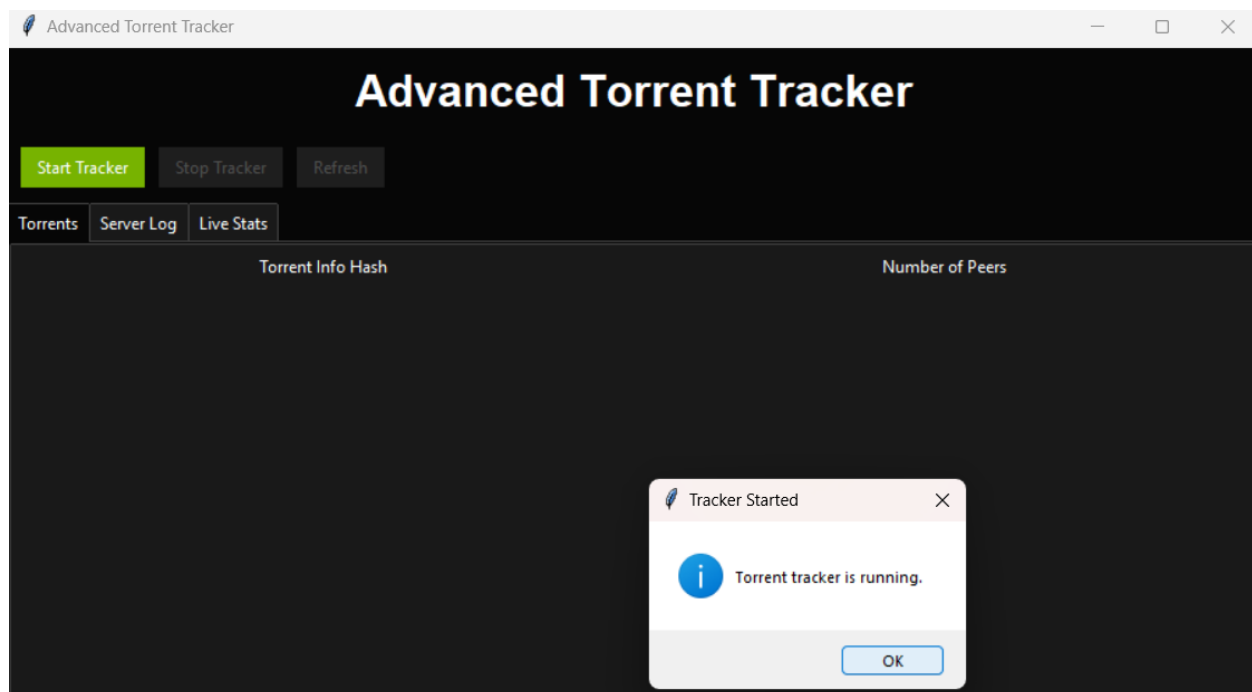


Figure 3. start the tracker python code

Now the tracker is running and the only thing is to download the torrent file the trough the local network which in this case is the mobile phone that is using the uTorrent app to download the torrent file.

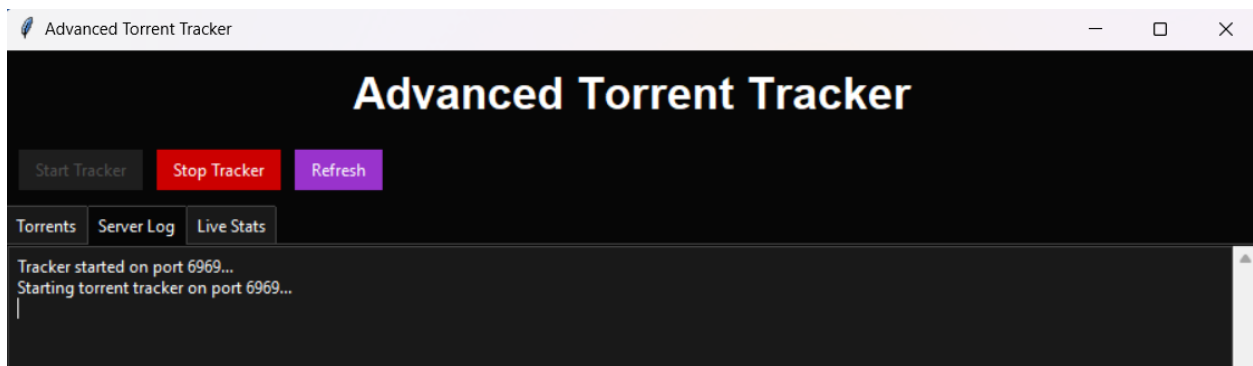


Figure 4. the server log shows everything about the tracker

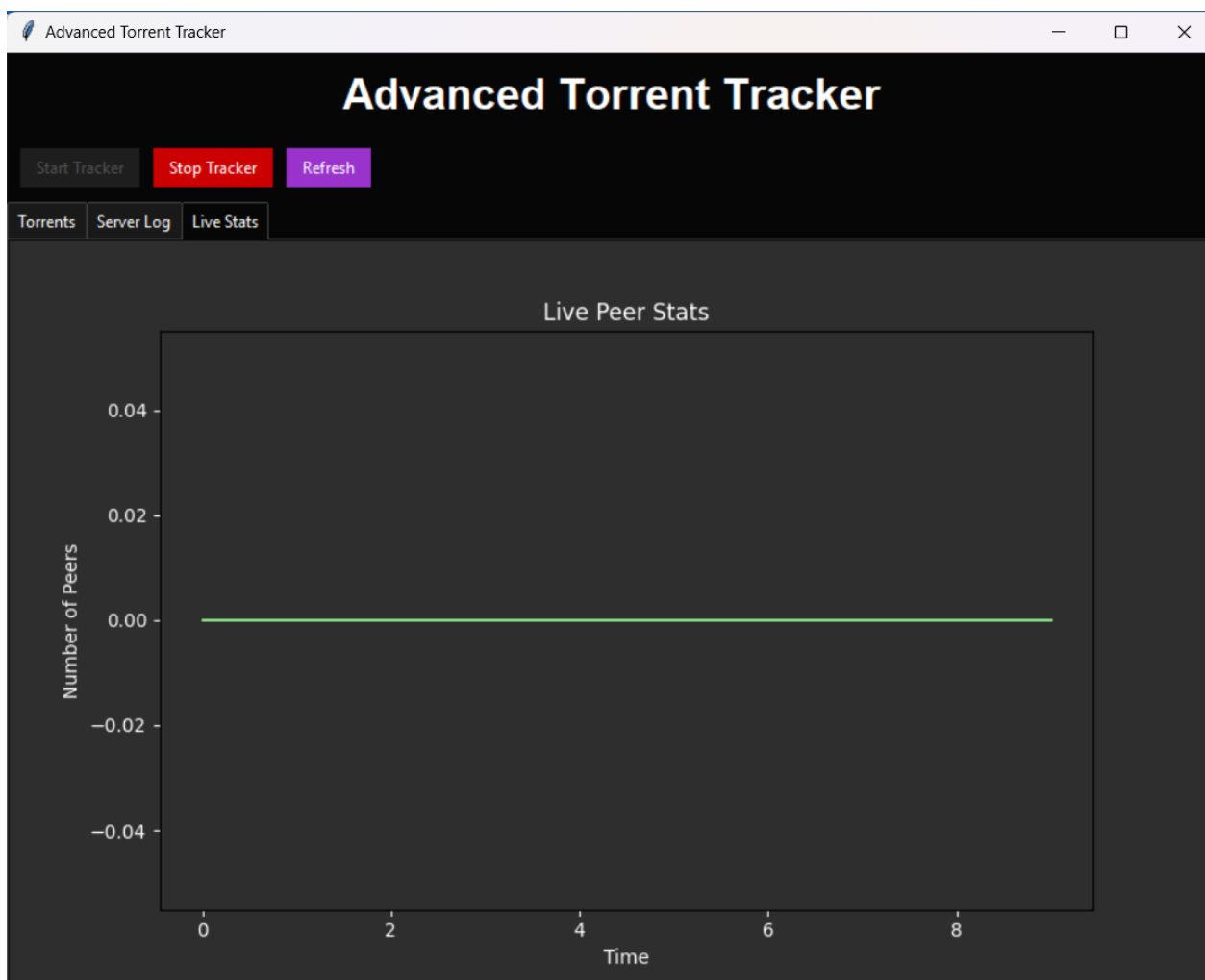


Figure 5. this plot update itself in **Realtime** and shows the peers in time

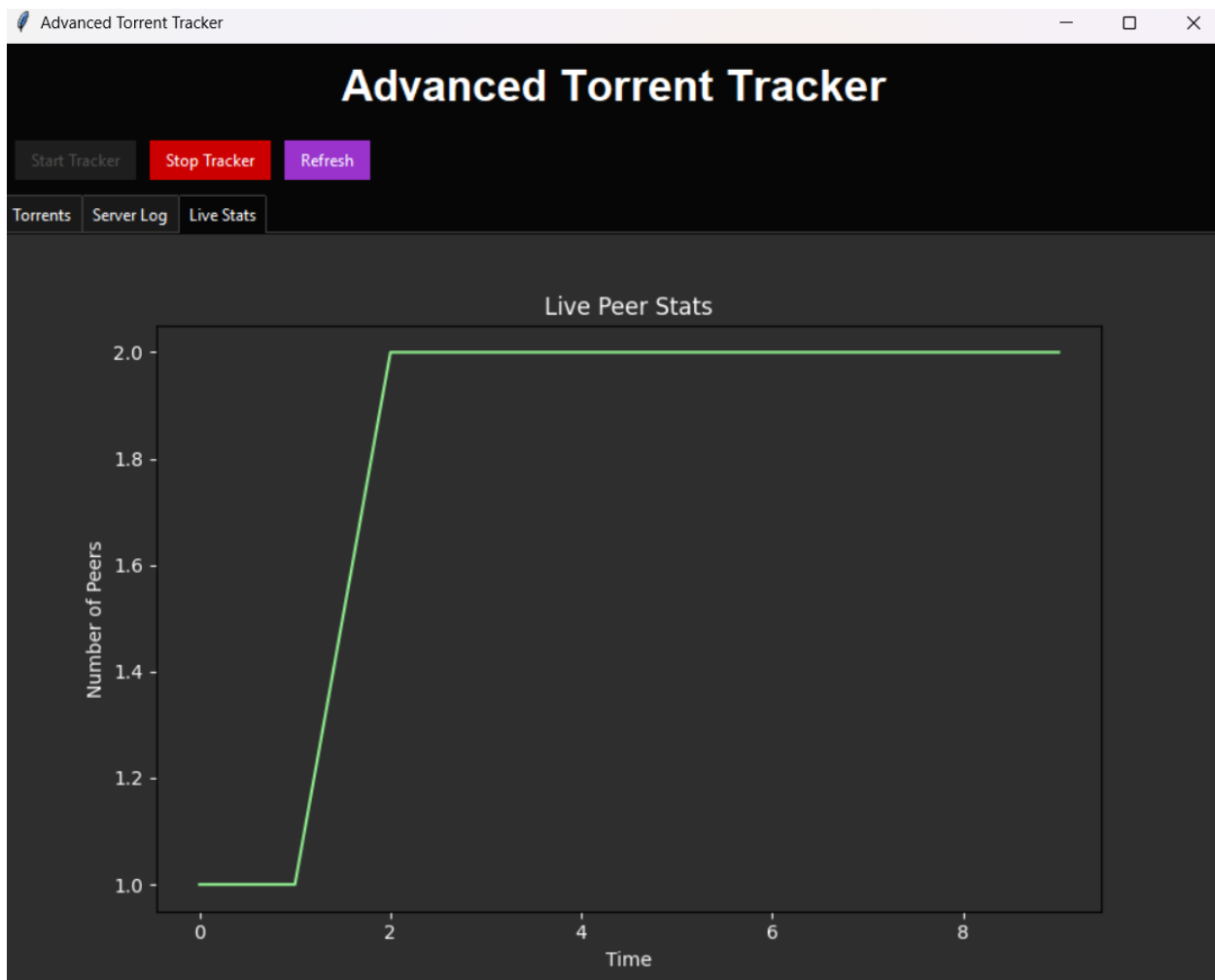


Figure 6. tracker also show the *Realtime* plot for the peers

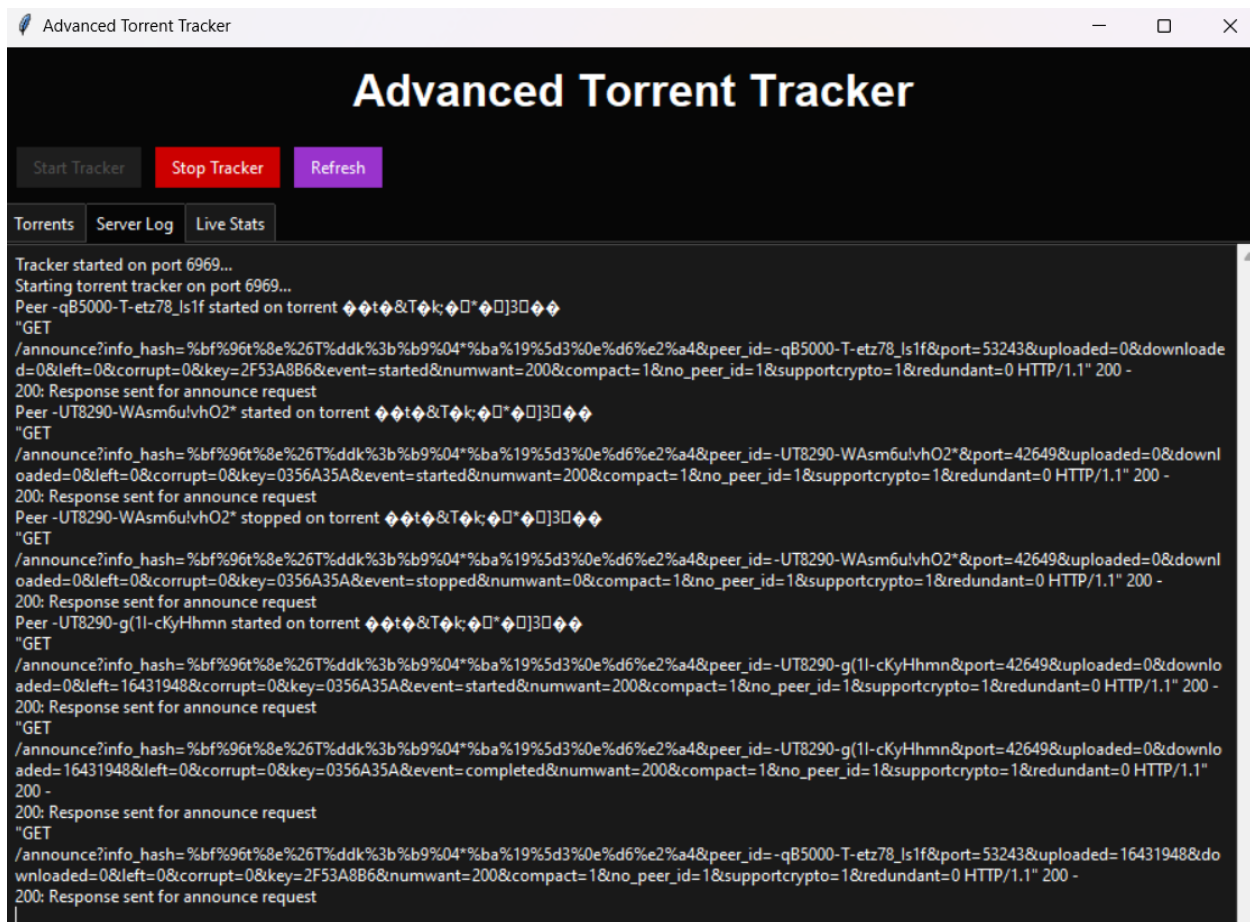


Figure 7. in the server Log the information also will be included in details

By the qbittorrent the .torrent file is created and by that app is seeding and through the tracker python code the phone that are connected to the same local network try to leech the torrent file which is done successfully by the earlier figures.

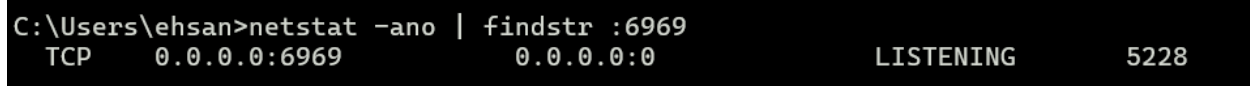


Figure 8. showing the port is used by the service

So, for now we need to shut down the app so the port be freed and be usable for another as we see earlier, we can shut it down through the app like the below figure.

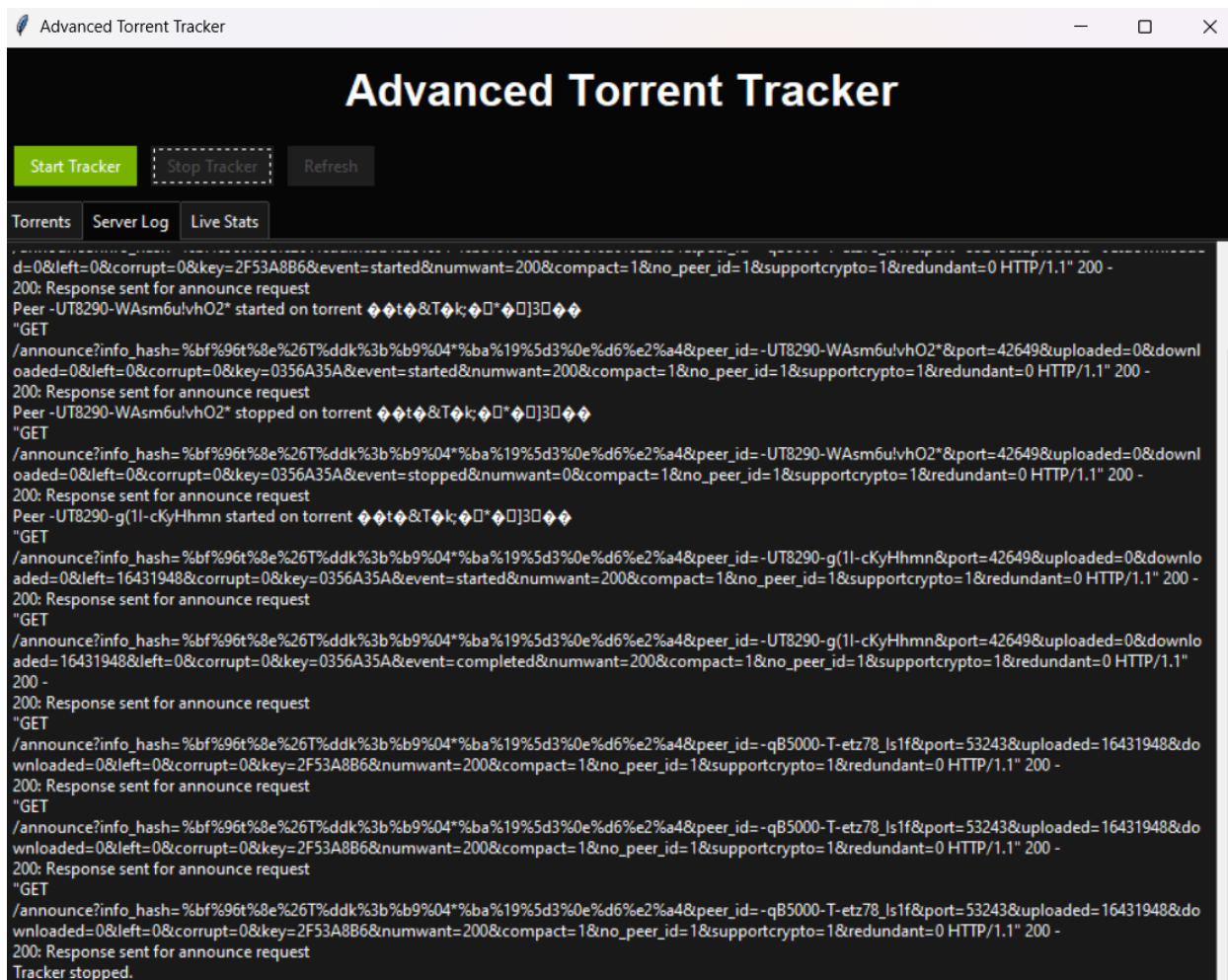


Figure 9. in this figure we can see the tracker is stopped and the port is free

Results

The experiment successfully demonstrated the core functionality of a local BitTorrent tracker in facilitating peer-to-peer file sharing. A file was seeded using **qBittorrent** from a local machine and then leeched on a mobile device through **uTorrent**. The tracker, written in Python, coordinated the communication between the two peers.

After the tracker was launched on the local machine, the seeder (qBittorrent) created a torrent file with the correct tracker URL and started seeding the file. The mobile device (leecher) connected to the same local Wi-Fi network and added the torrent file to uTorrent. Almost instantly, the tracker responded to the leecher's request with information about the seeder. The uTorrent client then established a direct connection to the seeder and began downloading the file.

The file transfer progressed smoothly, with both clients showing proper statuses—qBittorrent displayed "seeding" and uTorrent displayed "downloading" until the transfer was complete. Once the download finished, uTorrent switched to "seeding," showing that the file had been fully downloaded and was now available to share with other peers.

Performance

The performance of the peer-to-peer network was efficient, given the relatively small size of the file (a 50MB test file was used). The transfer speed averaged around **2-5 MB/s**, which is expected over a local Wi-Fi network. Factors such as proximity to the router and network traffic had minimal impact on the transfer speed.

However, some minor issues arose during the setup of the tracker. Initially, there were firewall restrictions that blocked the tracker from receiving peer requests. After adjusting firewall settings and ensuring the proper port forwarding was configured, the issue was resolved, and the tracker functioned as expected.

Additionally, the seeding client occasionally took a few seconds longer than expected to recognize the leeching client's request. This delay was traced back to the tracker configuration and was ultimately resolved by adjusting the announce interval in the Python tracker code.

Validation

The validation of the file transfer was performed by verifying the integrity of the file on the leecher's device. A **checksum comparison** between the original file on the seeder's machine and the downloaded file on the leecher's device confirmed that the file had been transferred without corruption or loss.

The checksum validation was done using the **SHA-256** algorithm. The hash values for both the original and downloaded files were identical, proving the accuracy and integrity of the file transfer.

Observations and Additional Insights

The experiment also highlighted several key aspects of the BitTorrent protocol's efficiency in managing peer-to-peer connections within a local network:

- **Decentralization:** The experiment demonstrated how BitTorrent allows decentralized file sharing by distributing the file transfer load across peers.
- **Peer Communication:** The tracker effectively facilitated the connection between the seeder and leecher, ensuring that the file-sharing process was streamlined.

In summary, the experiment successfully established a functional local tracker, facilitated seamless communication between peers, and achieved efficient file transfer within a local network, meeting the objectives of the task

Challenges Faced and Improvements Suggested

Challenges Faced

During the setup of the local BitTorrent tracker and the peer-to-peer file sharing process, several challenges were encountered, most notably related to network configuration and tracker stability. One of the initial issues was configuring the firewall on the seeder's machine. By default, the firewall was blocking the port used by the Python-based tracker, preventing it from receiving requests from peers. After troubleshooting, the issue was resolved by opening the specific port required for communication and ensuring proper port forwarding on the local router.

Another challenge involved the responsiveness of the tracker when handling requests from the leecher. On occasion, there was a noticeable delay between the leecher's request to download the file and the seeder acknowledging this request. This issue was traced back to the announce interval, a setting in the tracker that determines how often peers should announce their presence. By lowering this interval, the responsiveness of the tracker improved, and the connection between seeder and leecher became faster and more reliable.

In addition, there were challenges with verifying the tracker's function in real-time. Although the Python tracker was successfully implemented, monitoring its activity required manual log inspection, which was not as user-friendly as anticipated. Implementing more detailed logging or even a graphical interface for tracker status could make it easier to monitor peer activities.

Improvements and Future Enhancements

Several improvements could be made to the system to enhance its performance and usability in future experiments. One of the main enhancements would be to simulate a larger number of peers. The current setup only involved one seeder and one leecher, but to truly test the robustness of the tracker and the peer-to-peer protocol, it would be beneficial to introduce more peers to the network. This would test the tracker's ability to handle multiple connections simultaneously and evaluate how the file-sharing process scales.

Another potential improvement could be in the design of the tracker itself. The basic Python tracker used in this experiment worked well for a small, controlled environment, but more advanced trackers offer features like distributed tracking (DHT) and peer exchange (PEX), which could improve the efficiency of file sharing without a centralized server. Implementing these advanced features would be an interesting direction for future work.

Finally, improving the user interface and accessibility of the tracker would be useful for non-technical users. A web-based interface that displays real-time data on the seeding and leeching process, peer connections, and file transfer speeds would make the system more accessible and provide better insights into its operation.

In conclusion, while the tracker and peer-to-peer system functioned as intended, there is room for enhancements in terms of network configuration, scalability, and usability for future experiments.

Conclusion

This project successfully demonstrated the setup and execution of a local BitTorrent tracker for peer-to-peer file sharing. By configuring a Python-based tracker and utilizing qBittorrent as the seeder and uTorrent as the leecher, we achieved efficient file transfer within a local network. The system allowed for seamless communication between peers, with the tracker effectively managing connections and file distribution.

The results of this experiment highlighted several key aspects of the BitTorrent protocol. First, it reinforced the decentralized nature of peer-to-peer networks, where files can be shared without relying on a central server. Second, the experiment provided hands-on experience with the seeding and leeching processes, demonstrating how trackers play a crucial role in managing peer connections. Finally, it underscored the importance of proper network configuration, such as firewall settings and port forwarding, for ensuring the functionality of the tracker.

From a broader perspective, this experiment contributes to a deeper understanding of networking protocols and file distribution within local environments. Peer-to-peer file sharing, particularly using the BitTorrent protocol, remains a popular and efficient method for distributing large files. The ability to set up a local tracker offers potential applications for controlled environments, such as educational or enterprise networks, where files need to be distributed efficiently.

For future work, it would be interesting to explore the scalability of the system by introducing additional peers and testing larger file sizes. Additionally, integrating advanced features like Distributed Hash Table (DHT) or peer exchange (PEX) could further enhance the tracker's capabilities. Such developments could improve the robustness and scalability of local file-sharing systems, making them more suitable for broader applications.

References

- ◆ GitHub repository “[GitHub.com](https://github.com)”
- ◆ BitTorrent Specification.” [BitTorrent.org](https://www.bittorrent.org/)”
- ◆ qBittorrent Official Website," <https://www.qbittorrent.org/>.
- ◆ python documentation about the http server lib
<https://docs.python.org/3/library/http.server.html>
- ◆ python documentation about ttkbootstrap
<https://ttkbootstrap.readthedocs.io/en/version-0.5/>
- ◆ download uTorrent app for android
<https://www.utorrent.com/mobile/>
- ◆ how to open port in firewall
<https://www.tomshardware.com/news/how-to-open-firewall-ports-in-windows-10,36451.html>
- ◆ finding the different torrenting apps
<https://www.safetydetectives.com/blog/best-torrenting-sites/>