6.5610 Final Project Report: Distributed Private File Downloads

Mario Mrowka
`mrowkam@mit.edu`

Patrick Whartenby
`pwartenb@mit.edu`

Andrew Zhao
`aazhao@mit.edu`

Abstract

We propose a distributed file download system that uses the power of Private Information Retrieval (PIR) to privatize peer-to-peer (P2P) file downloads. Users can privately download files from a large database privately, ensuring that peers can not determine what file they desire. Our system leverages ideas from BitTorrent and SimplePIR (Henzinger et al., 2022) to prevent peers in the network from gaining information about what other users are attempting to download. Though there are ethical concerns about these platforms, we feel that increased privacy in file sharing is a net positive for users on an insecure internet.

1 Introduction

P2P systems utilize distributed resources to execute a function such as file sharing in a decentralized manner. A notable example of a P2P system is BitTorrent, where users can upload and download files between users. Our project applies a PIR scheme to ensure that when peer A queries another peer B, peer A reveals nothing about what they are searching for (Kushilevitz and Ostrovsky, 1997). We combine these concepts to distribute some of the high computational and communication costs associated with PIR across a network of users. Our system ensures users can continue to query services without revealing what content they want to download.

The motivation for our approach comes from two papers. The first is "One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval," which describes SimplePIR, the fastest single-server private information retrieval scheme known to date (Henzinger et al., 2022). The second is "Can Peer to Peer (P2P) Replace Direct Download for Content Distribution," which explains how P2P networks can deliver paid digital content instead of direct-download models (Sherman et al., 2007). When combined with PIR, we achieve a distributed file download network that provides the same security of direct downloads and the privacy of a PIR scheme.

In our scheme, users share pieces of files within a peer-to-peer network. Users can make PIR queries to determine what files each peer has and make separate PIR queries to download the respective files.

2 Related Work and Background

2.1 BitTorrent

The inspiration for the file download system comes from the BitTorrent protocol initially released in 2001 (Cohen, 2001). The system attempts to replace the need for servers to facilitate large downloads by breaking files up into small pieces called chunks. Once a peer has received a chunk, they become a source for that portion of that file.

To upload files to the network, a source peer breaks the file into several identically sized chunks. The size of each chunk is arbitrary, but most implementations use something between 32 kB and 26 MB. The course peer then hashes each chunk and creates a .torrent file with the hashes. The hashes ensure that other users can verify the legitimacy of the data they receive. Along with the hash of the piece, the .torrent file also contains a URL to a tracker instance. Peers can query the tracker to get the location of other peers. ”Seeders” are peers that hold the entirety of a file and continue to participate in the network. As more users download pieces of content, they can also share the newly downloaded chunks with the rest of the network, allowing for a more efficient exchange of information (Cohen, 2001).

One of the most significant drawbacks of the system is the amount of information leaked to peers.

To connect to the network or download data, every other party in the network must know what .torrent file they are using, and thus learn what content they want to download. Privacy concerns are elevated when considering that these networks are frequently used to distribute copyrighted content or to avoid censorship.

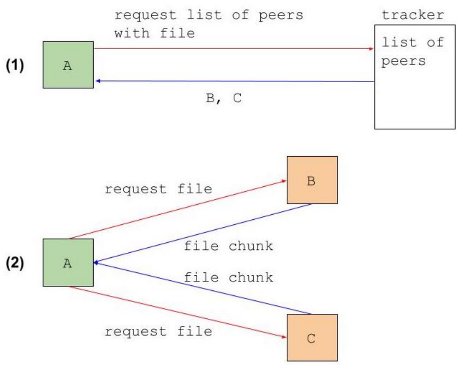

Figure 1: Example scenario illustrating our system. **(1)**: Peer *A* makes a PIR query to the tracker to request a list of peers with chunks of a file. The tracker returns peers *B* and *C*. **(2)**: Peer *A* requests Peer *B* and *C* for the file chunks. Peers *B* and *C* return the file chunk.

2.2 SimplePIR

We model our PIR scheme after SimplePIR, developed by Henzinger et al. (2022). Similar to the scheme described in class, this system stores a size N database in a $\sqrt{N} \times \sqrt{N}$ matrix, referred to from here forward as *db* (Kushilevitz and Ostrovsky, 1997). Each entry has a corresponding column i and row j. To access the database, the client builds a vector of all 0s and sets the index corresponding to the column of the desired entry to 1. To ensure the privacy of the query, the system relies on the LWE assumption under parameters n, q, and χ. Using the secret key version of the LWE encryption scheme provided by Regev (2009), the query vector is encrypted. The equation for the encryption of a query vector $\mathbf{u}$ is:

$$\text{Enc}(\mathbf{u}) = (\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e})$$

which is the the Regev encryption that is zero everywhere but a 1 at index i and 0 everywhere else for some LWE matrix $\mathbf{A} \in \mathbb{Z}^{\sqrt{N} \times \sqrt{N}}$, secret $\mathbf{s} \in \mathbb{Z}^{\sqrt{N}}$, and error $\mathbf{e} \leftarrow \chi^{\sqrt{N}}$ (Henzinger et al., 2022)

Once encrypted, the client sends the query to the server, which multiplies it by *db* and returns the resulting vector. Finally, to recover the answer to the query, the client parses the correct row of the query and decrypts it with a standard LWE decryption algorithm, shown below to recover $db{\cdot}\mathbf{u}$, which is the i-th column of the database.

3 System Design

Our scheme is designed as a way to add privacy to distributed file download platforms. It also provides a novel use case for PIR schemes. Thus far implementations of this idea remain held back by high communication and computation costs. We found that splitting up the queries costs across a P2P network decreases the computational burden on any one server, at the cost of higher communication bandwidth. While the communication cost of downloading these files is a concern, we believe that clients are willing to pay the price. One example of where this already exists in practice is the Tor network. Tor users accept that the connection will be slower in exchange for not having their IP address linked to certain sites. Our system follows a similar logic, as the belief is users will accept the longer time it takes to download files in exchange for privacy regarding what they download.

3.1 System Setup

The setup for our system begins when a source peer decides on a set of files they want to share. The source then divides files into chunks and hashes each of the chunks. After converting eh chunks into the database form described in section 3.3, the source creates a tracker URL, and uploads their endpoint to the site. To allow others to join the network they upload a .torrent file with the tracker URL and the hashes of chunks to the internet.

3.2 Peer Discovery and Initial Query

To first discover peers in the network, clients visit the public tracker URL. The URL is discoverable from the .torrent file new users can get from the internet. The tracker provides information regarding what peers are a part of the network. After querying the tracker, clients can query the peers to determine what files they hold. A peer will not connect to the tracker again, unless the vast majority of peers it is connected to have disconnected from the network. Following the initial connections to the tracker URL, new peers are discovered by talking to query other known peers in the network.

The initial query to a peer is done using SimplePIR. Each peer holds a database containing entries that show peers they are connected to, which chunks they hold as well as the data for those chunks. The exact structure of the database is explained further in the next section. The security property of SimplePIR ensures that a query for the list of chunks that the database holds is indistinguishable from a query for a chunk that contains file data and a query for the neighboring peers (Henzinger et al., 2022). This provides a unique "plausible deniability" property in that even the owner of the files can not determine if a peer is downloading files at all, just asking to see what files they hold or trying to discover new peers. The list of files also contains the location of each file in the database to ensure future queries are made to the correct place.

3.3 Database Structure and File Download

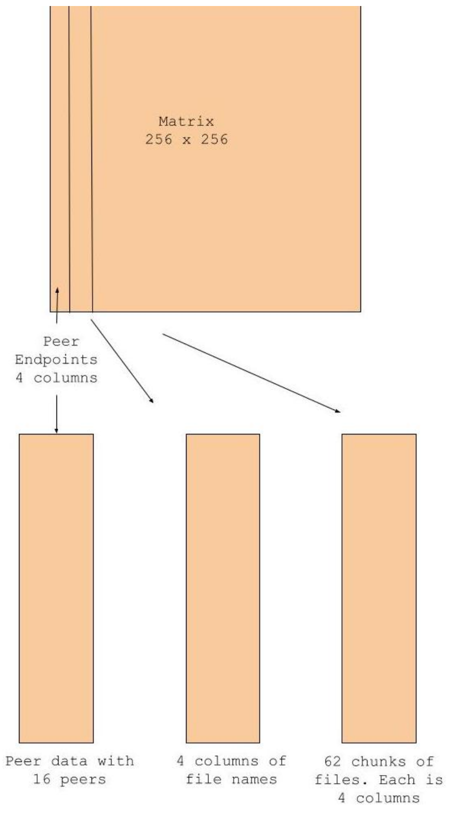

Figure 2: P2P Chunk Structure with Chunk Size of 4 columns

Our network's database structure works to limit the number of queries needed to determine what files a user has and to download said files. The primary observation this relies on is that when a PIR query is made an entire column is returned rather than a specific index in the matrix. To take advantage of this, we arrange the database, such that the files are stored in the columns. The database is grouped into "chunks," all of which are the same size. Each chunk contains multiple columns and multiple chunks make up the entire file. Figure 2 shows an example of our database structure. The first chunk holds a list of peers that the person knows about. The second chunk in the database stores information about what files the database holds and which chunks make up those files. Even though a user's first query to the system is likely to request the second chunk, neither an observer nor the owner of the database can tell because the Regev encryption scheme used in SimplePIR provides chosen-plaintext attack (CPA) security (Regev, 2009). This ensures that, without the secret key, no one can tell the difference between a query for the first chunk and a query for any other chunk. An adversary is unlikely to have the ability to make any assumptions besides this because future queries could be for the list of peers, list of files, or file data. All three of these are indistinguishable as mentioned above.

To download an entire chunk, a user makes multiple queries for consecutive columns, as SimplePIR only returns one column at a time. Once a peer completes their download they can decrypt the respective columns to recover the full chunk. They now have the option to either host the chunk on the network themselves or not. They are not required to host the file as this would take away from the privacy as someone hosting the file has downloaded it at some point. Additionally, by waiting for a peer to upload a new file, adversaries can see decryptions of PIR queries opening up the possibility of chosen-ciphertext attacks, which SimplePIR does not protect against. To reduce this vulnerability when peers choose to host files they downloaded from other peers, we recommend switching the secret key used for PIR after each query.

4 Analysis

4.1 Correctness

By using an existing PIR scheme, we rely on the correctness of SimplePIR to ensure that the reconstructed value is equal to the desired value from the database (Henzinger et al., 2022). The SimplePIR paper, linked in the references section, provides a detailed proof. We rely on the BitTorrent protocol, specifically the hashing of chunks, to guarantee the reliable reconstruction of files.

4.2 Security

The system is *secure* if no parties can discover the file another peer desires to download. We define a file to be a subset of chunks uploaded by the original seed peer. We also assume an honest but curious tracker system, and non-collusion between peers. We define a "desired file" to be the file the peer originally wanted, not any chunks/files that are downloaded to help with seeding.

We also note that our system has inherent limitations that we considered while creating this definition. For example, our system does not guarantee a peer to see every chunk of the original set of files before finishing its download. This means that we need the non-collusion principle, since an adversary that has control over the entire network can know the total set of chunks a peer has seen. They can then determine what chunks are missing from that set, and narrow down the range of possible files the user downloaded. By continuing to stay on the network after finishing a download, a peer is able to reduce the ability of an adversary to narrow down the possible files, but we do not force users to follow this.

Below is a list of possible attacks we considered when finetuning our system:

1. A peer asking the tracker for more peers after not finding every chunk it desired means that the first set of peers did not have the entirety of the file it desires. This gives information that the tracker can use to narrow down which file is attempting to be downloaded. This is why we only ask the tracker for peers once.

2. The chunks a peer hosts cannot be influenced by the file it desires. Doing this would cause any peer that queries it to know which file it desires. Our mitigation to this is to have a peer only host random chunks that it finds, and not the chunks it desires.

3. A peer not updating their database reveals that they just queried for a chunk that they wanted. This is because if a peer does not host the desired file, any query to that peer is for a chunk they want to host or their list of peers. Both of these cause a database update. Thus the only time a database update does not occur is when a user queries for a chunk they desire. Our mitigation for this is to have peers constantly update their personally hosted database with the rarest chunks in their local network. This provides two benefits, first it allows peers to make queries for chunks owned by another peer. These queries do not cause the database to update. Secondly it improves movement of chunks through the network, as rare chunks are more likely to be propagated.

Our system fulfills the definition above. We note that each individual query does not leak information about which chunk a peer is attempting to download based on the LWE assumption inherent in SimplePIR. Because of the non collusion principle, and that peers should be churning through chunks they own information cannot be gained about whether a peer does or does not desire the chunks of another peer, since they may be asking for a list of known peers or for which chunks the peer owns.

5 Implementation

The implementation of our system leveraged the Go programming language. We chose to use Go because it provides easy to use, intuitive rpc calls, uses static typing, and is a compiled language. The rpc calls made incorporating P2P communication significantly easier. Static typing helps prevent bugs, while compiling makes the program easier to distribute to users. All of the PIR code was written from scratch, including functions to multiply matrices, add matrices, encrypt column vectors, decrypt column vectors and other necessary functions to implement different PIR schemes. The PIR scheme used in the implementation is more rudimentary than SimplePIR as it does not use hints or pseudo random matrices. We made this choice to decrease the complexity of the implementation. The code for this project can be found in the following GitHub Repository.

We made use of the crypto/rand library in Go wherever possible to generate the best possible randomness. We did this as the normal random library in Go is not cryptographically secure, and since Regev encryption relies on randomness to encrypt we opted for the more secure version (Regev, 2009).

Our current implementation uses a seed URL that is hosted locally on one of our laptops. In future updates to the implementation, we hope to have an MIT hosted URL that users are able to query for information about peers in the network. When a peer queries the seed URL, it returns a list of endpoints, IP addresses and ports, that are the current peers in the network. The implementation also adds support for users to make themselves a peer on the network by opening up a random high-port for other peers to communicate with. Our current implementation allows for very rudimentary file sharing.

6 Applications

Applications of our system are similar to the normal BitTorrent system. It can be used as a simple P2P file sharing system, but is more commonly used to distribute copyrighted or other types of restricted content. Section X includes further discussion about the ethics of such a system.

A few other applications we considered involved distributed databases. Some existing applications of BitTorrent include open source video games, Linux distributions, video sharing, and file syncing. All of these use cases apply to our system with the added benefit of hosts being unable to track who downloads what from their server.

7 Future Work

Due to limits with time and resources, we believe that there is a lot of room for improvement in our current system design and implementation. One area for future work we found particularly interesting is the possibility of fully homomorphic path finding. In our system there is no guarantee that the local network of a peer will have all of the chunks of the original file, this means that they need to discover new peers that may have the chunks they want. Our current approach simply involves discovering random peers and checking if they have the desired chunk. We believe it would be possible to ask a peer for who in their local network has a certain chunk, without revealing what that chunk is through fully homomorphic pathfinding. Though we could not successfully implement the idea, we think it poses an interesting question for future research.

8 Ethical Concerns

One of the issues many people have with the current form of BitTorrent centers around the idea that many people use it to download pirated or other forms of illegal content. By building upon initial BitTorrent implementations to add private queries, our system could make it more difficult to track the distribution of illegal content. As we considered the possible implications of such a design, we considered the arguments made in "Keys Under Doormats." Abelson et al. (2015) argue that the fundamental insecurities of today's internet make it such that we should be wary of sacrificing privacy for security for the potential benefits to law enforcement.

Our system is in no way intended to make cybercrime easier. We believe that all users of the internet have a fundamental right to privacy and ought to expand their abilities to keep sensitive information private.

9 Work Division

Our team attempted to divide the work as much as possible. The ideation phase was a collaborative effort as we sought to mold our initial idea of a distributed PIR scheme into its final form. Regarding the implementation, Mario Mrowka wrote much of the PIR code. Patrick Whartenby and Mario developed much of the networking code together, while Andrew Zhao was responsible for building an extensive testing suite. All three contributed to the paper and the presentation, with each handling the areas they felt most comfortable in.

10 Conclusion

Our scheme shows the possibility of privatizing P2P file sharing. The novel use of PIR further extends its possible use cases and helps to push the internet towards a place where users have a greater degree of privacy. While there still exists a lot of room for improving the efficiency of finding peers and finding files, we successfully implemented the system for small text files.

11 Acknowledgements

We would like to acknowledge the help we received from course staff and others in the completion of our final project. We want to thank our TA, Katarina Cheng for her guidance during the ideation and development phases of our project. We want to thank Prof. Corrigan-Gibbs and Prof. Kalai, not just for their excellent lecturing during the semester, but for their passion for the subject as it motivated and inspired us.

References

Harold Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael A. Specter, and Daniel J. Weitzner. 2015. Keys under doormats: mandating insecurity by requiring government access to all data and communications ‡. *Journal of Cybersecurity*, 1(1):69–79.

Bram Cohen. 2001. The bittorrent protocol specification. `https://www.bittorrent.org/beps/bep_0003.html`. [Accessed 14-05-2024].

Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2022. One server for the price of two: Simple and fast single-server private information retrieval. Cryptology ePrint Archive, Paper 2022/949. `https://eprint.iacr.org/2022/949`.

E. Kushilevitz and R. Ostrovsky. 1997. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 364–373.

Oded Regev. 2009. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):1–40.

Alex Sherman, Angelos Stavrou, Jason Nieh, Clifford S. Stein, and Angelos D. Keromytis. 2007. Can p2p replace direct download for content distribution.