Peer-to-Peer File Sharing System

Gabriel Lancaster-West

**Contents**

[ANALYSIS 3](#_Toc150326377)

[Background to the Problem 3](#_Toc150326378)

[The Current Solution 4](#_Toc150326379)

[Interview with the Client 4](#_Toc150326380)

[What I’ve learned from the interview 5](#_Toc150326381)

[Client Requirements 5](#_Toc150326382)

[Users & Project Limitations 5](#_Toc150326383)

[Objectives 6](#_Toc150326384)

[Possible P2P Solutions 6](#_Toc150326385)

[1. Centralised Index Server 6](#_Toc150326386)

[2. Kademlia 6](#_Toc150326387)

[3. Chord 7](#_Toc150326388)

[Chosen P2P Solution 8](#_Toc150326389)

[Possible Encryption Solutions 8](#_Toc150326390)

[1. Transport Layer Security (TLS) 8](#_Toc150326391)

[2. Advanced Encryption Standard (AES) 8](#_Toc150326392)

[3. Secure File Transfer Protocol (SFTP) 9](#_Toc150326393)

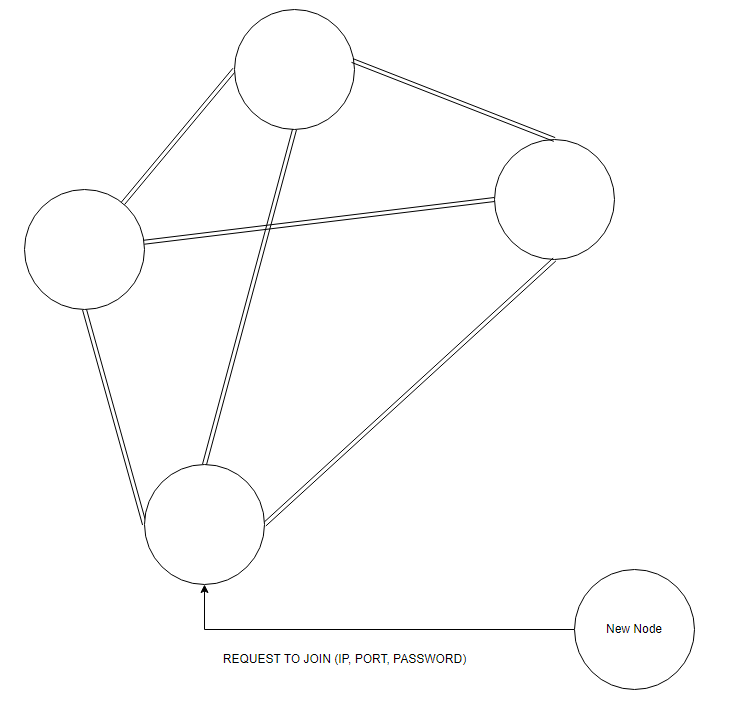
[Chosen Encryption Solution 9](#_Toc150326394)

[Data Dictionary 10](#_Toc150326395)

[Object Orientation Plan 10](#_Toc150326396)

[Entity Relationship Diagram 10](#_Toc150326397)

[Temp 11](#_Toc150326398)

[ 11](#_Toc150326399)

[A diagram of a computer process

Description automatically generated 12](#_Toc150326400)

# ANALYSIS

## Background to the Problem

My friend Dan wants to share files between multiple computers. Dan is very privacy minded and requires this system to be encrypted and secure. There should be capabilities for multiple (more than two) computers to share files, in case Dan adds more computers, or wants to share files with his friends.

## The Current Solution

Currently, my client Dan, is backing his folders up to Microsoft OneDrive, which stores a copy of folders on a central server. However, without paying, the standard user only has 500MB of storage, which isn’t enough to sync large videos, or basic apps.

A screenshot of a computer

Description automatically generated

*Above: Current system – Microsoft OneDrive.*

To share between different computers, Dan is having to sign into OneDrive on each device to access his profile on the central server, then he is manually downloading them from the website.

## Interview with the Client

This is a message chain I had with my client before beginning my project – when referencing the “current system”, we’re referring to Microsoft OneDrive at <https://onedrive.live.com/>:

Me: “What do you like about the current system?”

Dan: “it’s simple to add new devices, and its encrypted”

Me: “What do you not like about it?”

Dan: “It’s slow, it costs tons for storage, and its run by a massive corporation”

Me: “What could be improved with it?”

Dan: “Small or lighter files to run on computers (Not as important), and the storage should be cheaper or free”

Me: “What do you mean by small or lighter files?”

Dan: “Less computationally intense/more lightweight”

### What I’ve learned from the interview

* My client likes how easy it is to add new devices to the current system.
* My client wants to be able to run the current system on a computer with worse hardware.
* He both likes and dislikes the security of the current solution, because it is secured from hackers by encryption, but it is still vulnerable to Microsoft, as it is still stored in their servers.
* A financially cheap solution is very important for my client.

### Client Requirements

1. The program must have mostly the same base elements as the current system; this includes:

* Implementation of folders.
* UI/GUI to browse files and folders.
* Ability to download and open files from the UI/GUI.

1. The program must be easy to add devices to.
2. The overall system must be relatively lightweight.
3. The system must be secure and encrypted.
4. The data must not be stored in an area where there is a risk that the host can read your data.
5. Must cost as little as possible – so preferably not using a server to avoid electricity costs.

## Users & Project Limitations

Although I only have a single primary client, I would like to make this system capable to be used by many people to share data across the world on separate networks. My client is extremely proficient with technology; however, I want to make this sufficiently user friendly, so that it is reasonably intuitive as to how it works to a new user. Because of this, it is unlikely that a command line interface will be used.

The limitations to this project are the following:

* What I know: I am fluent in Python, and proficient in Rust. However, I’m not familiar with many other languages – such as Java, JavaScript, C, or C++.
* Time: I have until February 2024 to complete this project, giving me 5 months to complete this from start to finish.
* Python: Python can do almost anything another language can, with the help of external modules, which require pip to install most of the time. My organisation’s computers do not allow users to add executables to the windows PATH environment variable, which may cause pip to not work – therefore external modules may prove a challenge.
* Databases: I’m not fully comfortable with SQL, having knowledge of basic commands, but

## Objectives

What is needed in this project:

* Have a system which can send data between 2 devices using IP and port number.
* Use some method to determine which device has the requested data (DHT methods such as Pastry, Chord or Kademlia, or another method such as query flooding).
* Query devices asynchronously and recursively.
* GUI to coordinate the transfer process, either an app made in TKinter/customTkinter or a webapp made in Flask or React, like that used in many torrenting software.
* Must be scalable and lightweight.
* A database can be created by any device by querying devices, which indexes and displays all files on the network.
* Must be able to export this database.
* Must be able to search in this database.
* Should be aesthetically pleasing.
* Data sent must be secure.

## Possible P2P Solutions

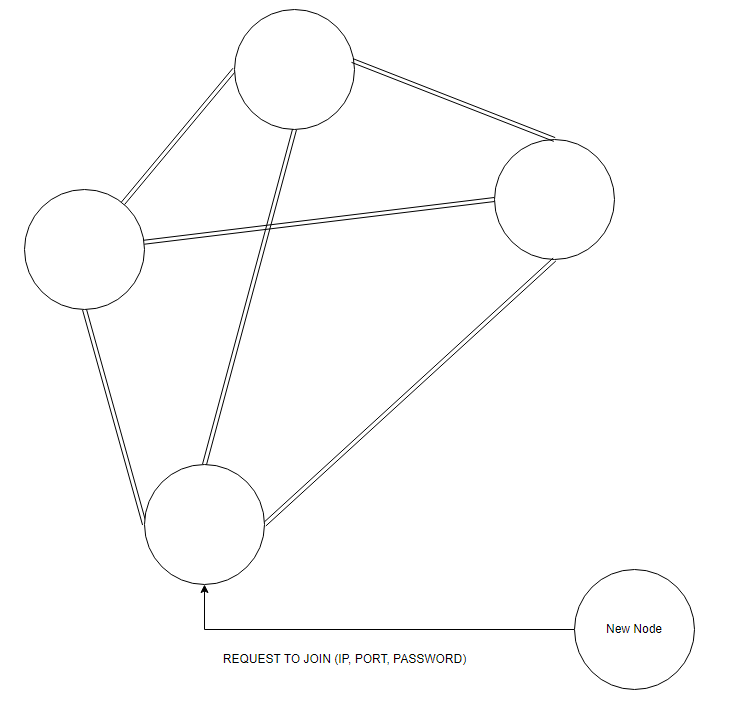
### Centralised Index Server

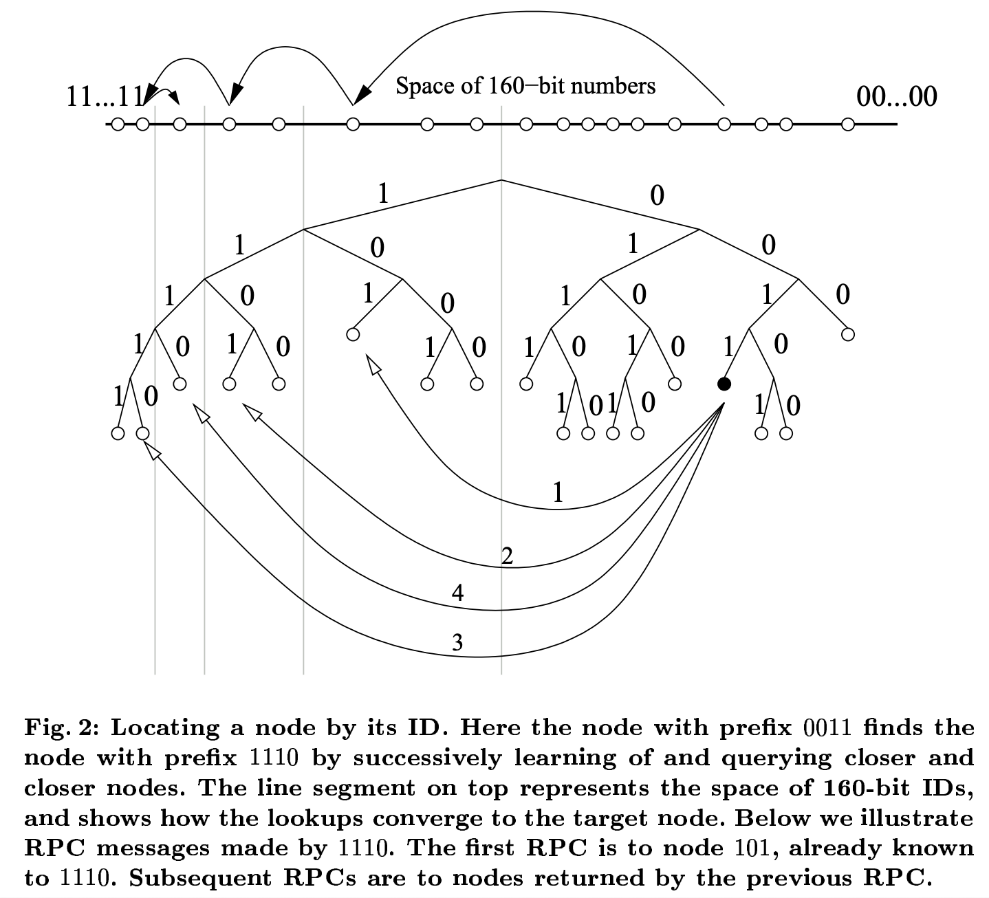
I could make a system which uses a central index server to coordinate data transfer between devices, like Gnutella. It should be noted here that this is still peer-to-peer, and the central server is used for device discovery. As there is a central trusted server, I could use this to coordinate HTTPS encryption between devices.

* If the server goes down, the whole network goes down.
* My client will have to rent a server, which costs money.
* It would be very easy to have a graphical interface of all devices and what they store, as that is all stored on the central server using something such as SQL.

### Kademlia

A system which uses the Kademlia Protocol could be created. Kademlia a very efficient peer-to-peer algorithm, to locate devices on an overlay network. Kademlia uses UDP to exchange data through ‘node lookups’ where each node is identified using a node ID (a 160-bit positive binary integer in the original specification), the Kademlia algorithm uses this ID to locate values – such as file hashes or keywords.

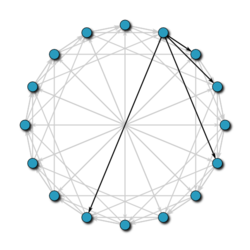




* This does not require a central server, so the network will always work on a device, unless all the devices “k-buckets” are offline, which is configured to practically never happen in a sufficiently large network.
* As there is no central server, HTTPS cannot be used. I believe that SFTP (SSH File Transfer Protocol) would work, as it relies on SSH credentials to symmetrically securely send data.
* A recursive node lookup would have to be used to show all devices and what they store, I may have to create an SQL database for user lookups by an individual device, but this may be impractical with large networks.

### Chord

I could create a system which uses Chord to locate devices on a network. It would then use a separate file transfer protocol, such as PGP (Pretty Good Privacy), which uses a mix of symmetric and asymmetric encryption to transfer data. In chord, each node has a ‘finger table’ to avoid a linear search for nodes. This table is such that the *ith* entry of node *n* will contain . This creates an order of , like Kademlia.

* This also does not require a central server and should never go offline if there are online nodes.
* In order to show all files available, a recursive node lookup would be required, and for easy use filtering I would use a list of dictionaries in Python, this would be impractical however for large networks, and if this is true this would be swapped to being stored in a JSON file.

*Above: A 16 node network. The ‘finger table’ for one of the nodes is in black.*

## Chosen P2P Solution

I will use the peer-to-peer system ‘Kademlia’. I shall use Kademlia primarily for its simplicity in handling offline devices when compared to Chord, as Kademlia’s load bearing is much more flexible than Chord, because it dynamically routes itself, instead of relying on a consistent hashing technique.

## Possible Encryption Solutions

### Transport Layer Security (TLS)

This is primarily known for its use in HTTPS; it uses an asymmetric cypher to begin communication with the server, where a symmetric cypher is then established. Typically, the server would provide a digital certificate to prove its legitimacy. However, there is no point for digital certificates in a peer-to-peer system – I considered replacing this with some kind of checksum for the data requested, but the recipient does not have a copy of the data, or the correct checksum to replace this with, so the recipient would have nothing to compare this to.

### Advanced Encryption Standard (AES)

AES is the first and only publicly accessible cipher approved by the U.S. National Security Agency (NSA) for top secret information transfer, having said this in a public statement:

‘The design and strength of all key lengths of the AES algorithm (i.e., 128, 192 and 256) are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 key lengths.’ - (Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce, 2003)

This fact is significant to me, as my client has explicitly said to me that they like the current system because it is encrypted. However, the standard is symmetric, therefore the recipient must know the key, which must be transferred using an asymmetric standard such as Rivest-Shamir-Adleman (RSA), RSA is infeasible for use for the file transfer itself due to its slow nature, but it is perfectly suitable for key transfer. Keys cannot be used for Kademlia lookups

### Secure File Transfer Protocol (SFTP)

SFTP is a network protocol for securely accessing, transferring large files and sensitive data, however this uses Secure Shell (SSH) that requires the client to be authenticated by the server. This is infeasible for my network due to its centralised nature, this could be used in conjunction with a Central Index Server (described earlier), but then it is far to dependant on the server, and not in my clients wishes, due to the risk of the server becoming compromised.

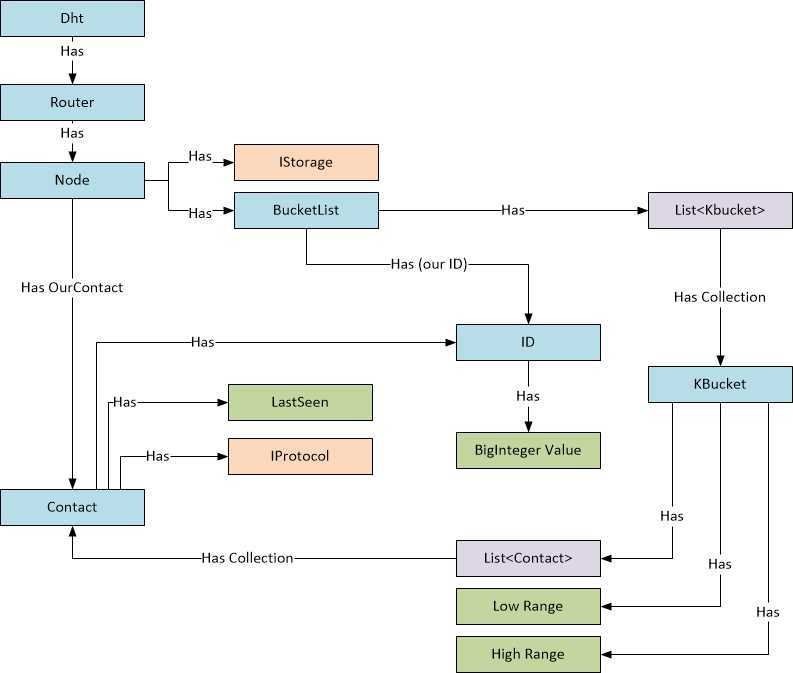
## Chosen Encryption Solution

I want to use AES/RSA for encryption, because SFTP and TLS are slightly too centralised, and fall out of the scope of my client's needs. I believe I could code a solution to RSA myself, but AES is extremely complex, so I will most likely use someone else's implementation for that, such as PyCryptoDome’s implementation: (<https://pycryptodome.readthedocs.io/en/latest/src/examples.html#encrypt-data-with-aes>)



I will use RSA to transfer the key for AES, which will be used for data transfer, I don’t think searches using Kademlia should be encrypted, as they are sent indiscriminately to devices on the network, and Kademlia uses UDP for its searches, therefore RSA cannot be used for key transfer, as it would drastically slow down the efficiency of the network. So another method would have to be used for encrypted Kademlia requests, like a password to join the network, which could then be used generate an identical AES key for all devices on the network, so that only devices on the network could view Kademlia traffic. This means there will be a global AES key on the network used for Kademlia, and a random key generated for file transfer, so that only the sender and recipient can view the data being transferred.

## Entity Relationship Diagram



This is taken from an explanation of the Kademlia Protocol (See “The Kademlia Protocol Succinctly” - Getting Started – A framework for the implementation, available at <https://www.syncfusion.com/succinctly-free-ebooks/kademlia-protocol-succinctly/getting-started>).

This shows the rough layout of the Kademlia protocol, which I am using in my project.

# DESIGN

What I want to make:  
  
1) device A has the option to use Kademlia to recursively search the network to create a database of files on the network

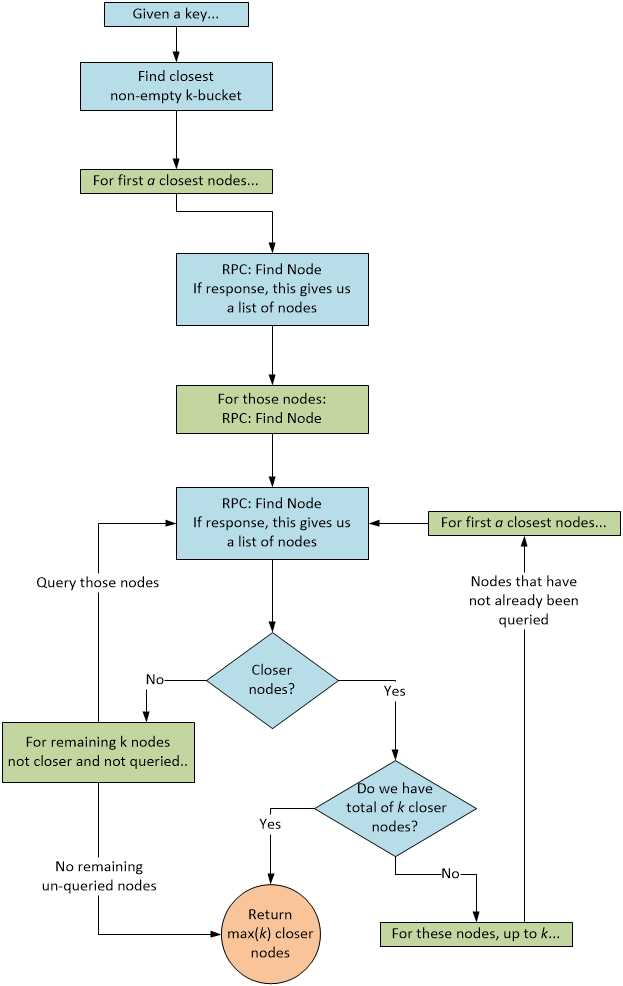
2) using the database (or directly using a file hash??) device A locates the device B, who has the file requested. This is performed using Kademlia.

4) They transfer public keys, so they can use RSA to transfer AES credentials.

5) AES is then used to transfer "file.zip" from A to B.

Advanced features:

## Node Lookup



From the flowchart above (<https://www.syncfusion.com/succinctly-free-ebooks/kademlia-protocol-succinctly/node-lookup>), a rough framework for how to look up nodes can be established.

A diagram of a computer process

Description automatically generated