Advanced Network Programming

P2P Chord like Network

George Kozakjian

Table of Contents

[**1.** **Introduction** 2](#_Toc513664798)

[**2.** **Overview of P2P Architecture** 3](#_Toc513664799)

[**3.** **The Chord Protocol architecture & Model** 3](#_Toc513664800)

[**3.1.** **Chord protocol architecture** 3](#_Toc513664801)

[**3.2.** **Chord protocol model** 4](#_Toc513664802)

[**4.** **Implementation & code** 5](#_Toc513664803)

[**4.1.** **Use of Sockets** 5](#_Toc513664804)

[**4.2.** **Data representation** 5](#_Toc513664805)

[**4.3.** **Generating keys with Sha1 Algorithm** 6](#_Toc513664806)

[**4.4.** **Join requests to chord** 6](#_Toc513664807)

[**4.5.** **Leave request:** 7](#_Toc513664808)

[**4.6.** **Chord Dynamic timeout optimization (DTO)** 8](#_Toc513664809)

[**4.7.** **Verbose output for diagnostics** 9](#_Toc513664810)

[**5.** **Results** 9](#_Toc513664811)

[**6.** **Limitations** 11](#_Toc513664812)

[**6.1.** **Bidirectional communication between peers** 11](#_Toc513664813)

[**6.2.** **Not very reliable code** 11](#_Toc513664814)

[**6.3.** **Security** 11](#_Toc513664815)

[**7.** **Improving the protocol** 11](#_Toc513664816)

# **Introduction**

The client server architecture model was the standard for the last 50 years, and worked very well serving needs of humanities. However, this model hit the melting point with the increase number of devices and users today. Hence a new architecture needed to emerge, where huge number of devices could be supported without any issues. This resulted in creating the peer to peer architecture model, where nodes in a network called peers are responsible for providing services in addition of consuming services. This model holds true because today end devices are capable of processing and providing services to others.

The P2P architecture model came into existence mainly to address scalability problems in the client/server architecture; the server at some point hit melting point and DDOSd from client requests and the heavy load on itself.

The P2P architecture model also uses the TCP/IP stack and the underlying IP network as a base layer to build an overlay network on, and uses the routing and forwarding services provided by the underlying IP layer to interface and communicate with other peers.

This new architecture, however, introduced its own design and implementation difficulties as something that has not existed before. Some of this difficulties include new lookup, and routing technologies; new technologies needed to be designed for lookup so that peers can find each other without sending lookup requests to a central server like DNS, new technologies needed so that the peer knows how to forward and find some peer either directly or route it through other peers which is addressed usually by DHTs – distributed hash tables.

# **Overview of P2P Architecture**

The P2P architecture model introduces an overlay network on top of the existing IP infrastructure, which facilitates the process of creating a P2P network without worrying about connectivity between peers. The architecture mainly include one type of node instead of two. This node is called a peer and acts as client who consume services provided by other peers, and as a server that provides services to other peers in the created overlay p2p network.

Under the curtains, and behind closed doors, the architecture operates by running its own software client at application layer of the protocol stack. It is mainly not so different from any other application layer protocol like http, ftp, or ssh. The main difference is that in P2P there is no client and server software, but one entity that is the peer who includes the code or application for both the client and the server.

It is true that this architecture overcame the client/server central model, however, there has been allot of applications and services who used a hybrid approach to create something very scalable and solid using the two models.

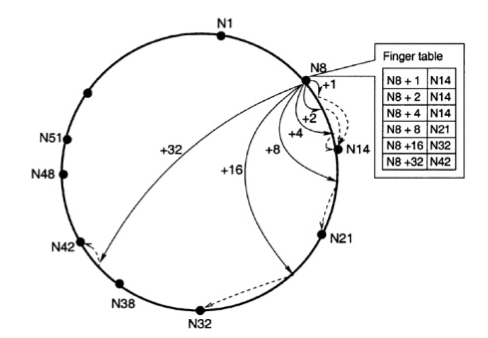
# **The P2P custom Protocol architecture & Model**

## **protocol architecture**

Chord is a protocol that is also an algorithm for P2P DHT network. A DHT is a lookup table consists of key-value pairs. Peers and resources are assigned keys, which are stored by different peers or nodes in the network. Different group of peers are responsible for storing keys for some nodes or peers. Chord provides hence a lookup service for any peer or resource in the P2P network by seeing the key it is assigned, which can be done by querying the peer responsible for the corresponding key.

Keys are mainly used as unique identifiers for peers and resources. These keys are usually Hash values obtained by using a hashing algorithm on the peers IP address and/or port number used to join the network. This key uniquely identifies and assures that the node indeed joined the chord network. Then it is stored by some nodes in the p2p network.

After the key is stored and distributed in the chord network, some peer can use chord lookup by generating the key to find the IP:Port of other peers or find resources assigned to them and start communicating with them.



## **Chord protocol model**

In this implementation a chord like overlay p2p network will be created. It will include a peer that who has different threads for client and server services running in the background. Each peer stores a chord peer table, which include all other chord peers in the network. Also, a peer stores successor and predecessor which are the peers that is closes and farthest from its own key. These peers can be used for lookup and forwarding services.

Component and subcomponent of peer and their responsibilities:

* Peer: sends a join request to chord, and starts the server/client threads
  + Server:
    - Listens on peer generated port for join requests, messages, and newjoin broadcast messages that come from other peers and indicate that other peer accepted new peer join request.
    - Server-DTP: transfers the data on command from the PI. Usually will be in passive state listening, waiting client to establish the data channel through control channel passive command (indicate to server it will be using passive technique to establish the data channel)
  + Client:
    - Acts like a CLI and provides interface to peer to send messages to the chord network and other peer
    - Takes some message and verifies other peer in the chord, and sends the message to the peer

In this implementation, the protocol interpreter (control channel) and DTP (data channel) will be using TCP. Also, these channels use different sockets and different TCP sessions and channels.

# **Implementation & code**

## **Use of Sockets**

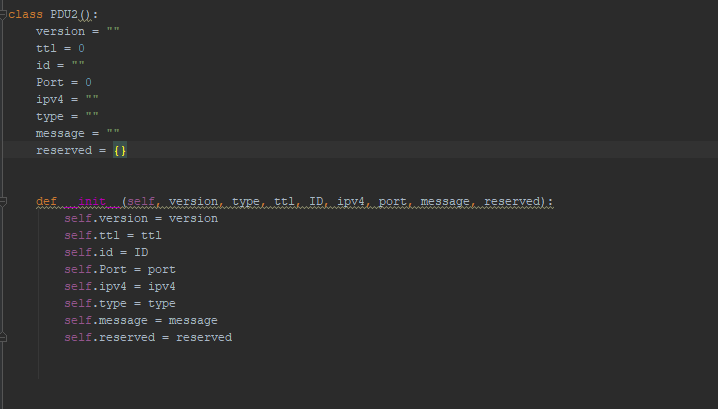
This implementation tries to mimic the real chord protocol by using python TCP sockets between a two peers. Use of socket is helpful in simulating and testing real-life client and server that are trying to communicate end-to-end over a packet-switched shared network infrastructure. A socket is a high level API that is used to create transport layer end-to-end communication channel, which can be utilized to create application layer protocols and services. It is comprised of an IP address and a port number, which uniquely identifies it on the internet or intranet.

The implementation starts with the peer creating its own TCP socket by using bind on a randomly generated port number outside the well-known TCP port range. Next, the peer sends a “join” request to the other specified peer, allowing the peer to obtain the peer’s remote IP address and Port number, the peer then tries to reply to the “joinack” request, by using the port and ip of the received TCP datagram from the peer. [Figure 1]

## **Data representation**

In this implementation, a real life chord is simulated that has its own PDU for data exchange of all types. This type scan be join, message, joinack, joinackid, etc. hence, to make it read-world like, a pdu class is used to act as chord pdu, and then serialization library is used, so that the class can be serialized and represented in heterogeneous data type, and sent to other peers that may have different underlying architecture.

In P2P, there are many data representations that can be used when transferring over the network. However, to make data representation heterogeneous, python picle is used in this implementation. This library serializes the pdu packets before sending it out through the network to the other peer.



## **Generating keys with Sha1 Algorithm**

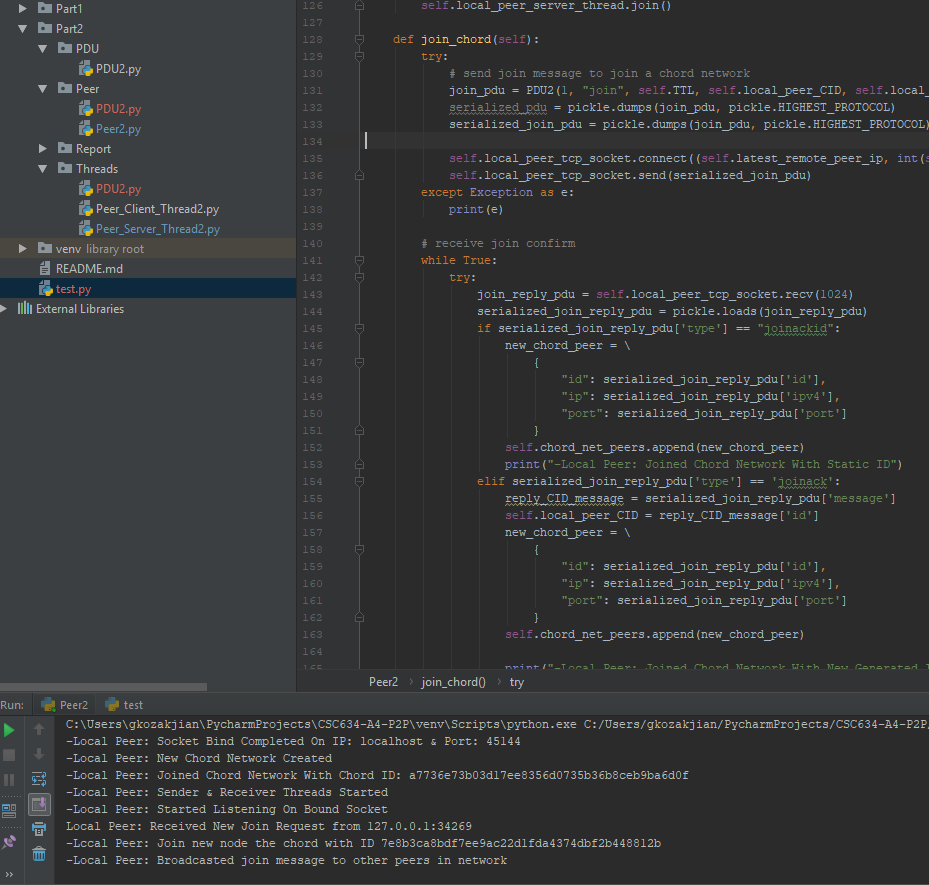
This implementation of chord uses IDs to uniquely identify each peer, and this IDs are generated with Sha1 hashing algorithm.

The python *hashlib* library is used to generate and compare MD5 hashes.

## **Join requests to chord**

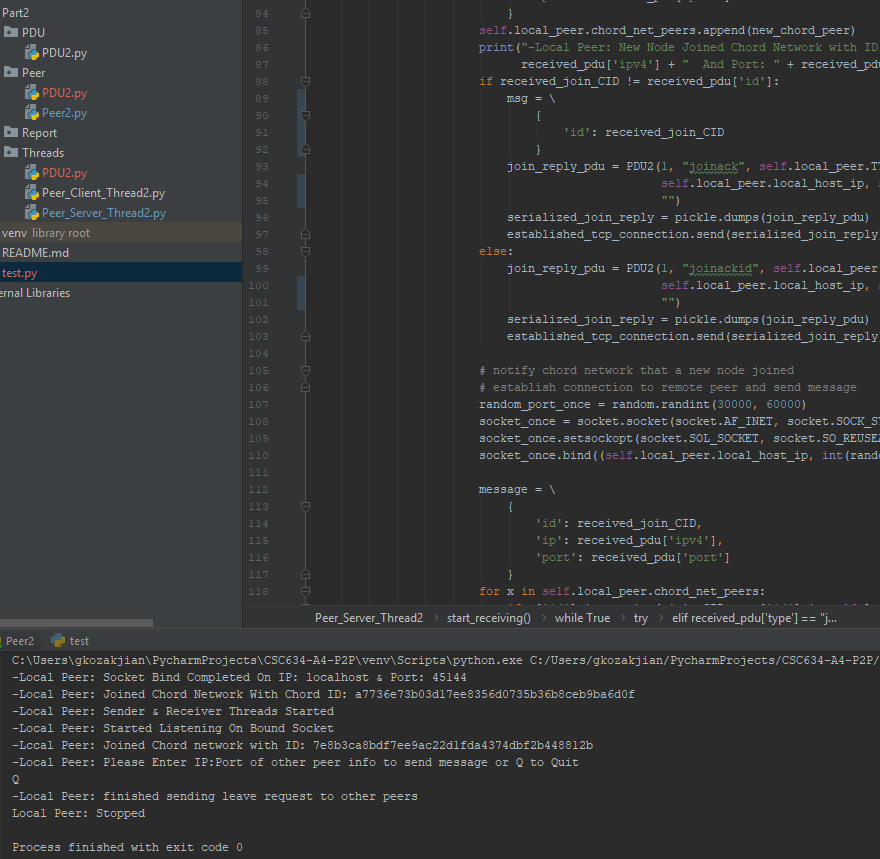
In the chord network, for a peer to join, it has to send a join request to some peer that is in the chord network. Hence, when a peer starts, it formulates a chord PDU with type of “join” and sends it to specified peer IP:Port in the run parameters. This peer processes the join request, checking for duplicate IDs if static is specified, or generating a new ID for the requesting peer from its IP:Port and replying a chord PDU with joinack type. Then this peer starts to notify other peers that a new node has joined the chord network, and stores it in its chord array.

Then each peer calculates the successor and predecessor and sets them according to the hash which Is the ID of the new node.



## **Leave request:**

when a chord peer leaves the p2p network, it sends a leave request to other peers to notify that it no longer exist in the chord, so that they can remove its resources and ID and reassign and readjust the chord network accordingly.



Hence, when the processes of the peer are closed, the peer sends a leave request in a loop to others that it knows in the chord network.

## **Chord Dynamic timeout optimization (DTO)**

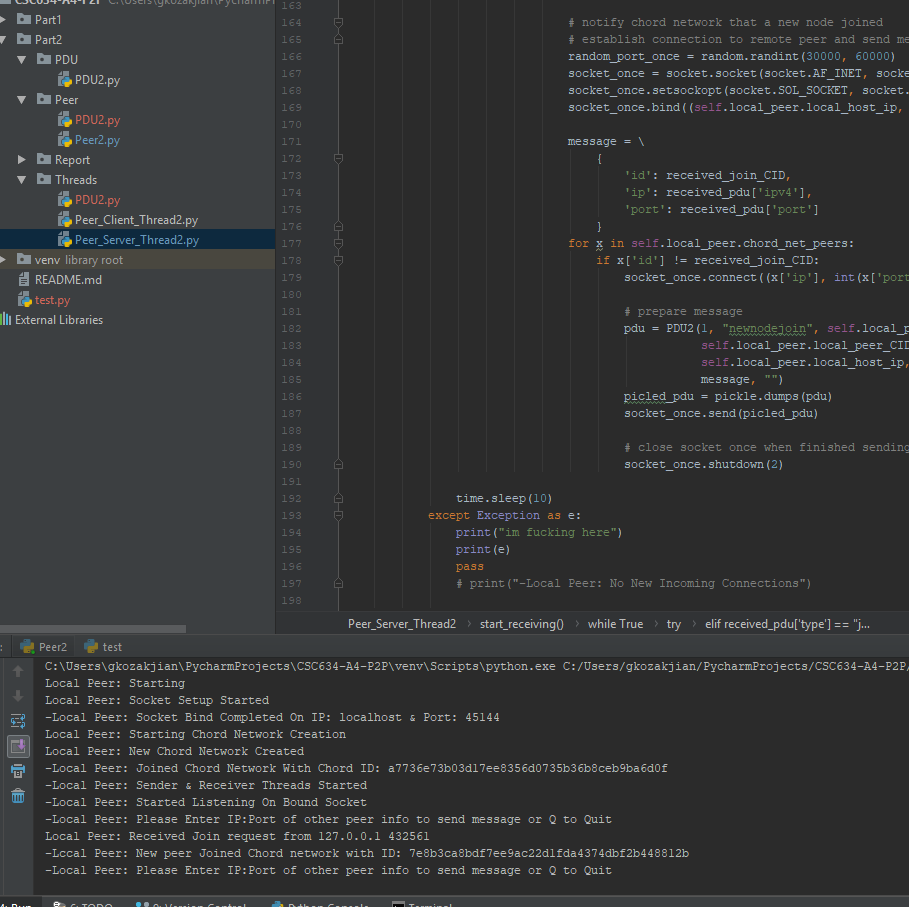
In this implementation, when the packet times out, we increase the timeout-value to next one. This delay will stay, even if the channel goes back to normal, and the latency of the channel decreases. Hence, this implementation adds an automatic channel optimization feature.

The idea is to keep track of the datagrams that are being received in time, and if the number of datagrams are more than some defined threshold value, then dynamic optimization will revert the latency/timeout back to a previous lesser value.

The dynamic optimization is off by default, and starts when first timeout is occurs. Upon optimizing the channel, the DTO goes back to being off, and starts again when a timeout occur.

## **Verbose output for diagnostics**

This implementation provides the ability to specify a verbose flag when starting the client and server, which triggers the verbose feature, allowing to see a detailed output.

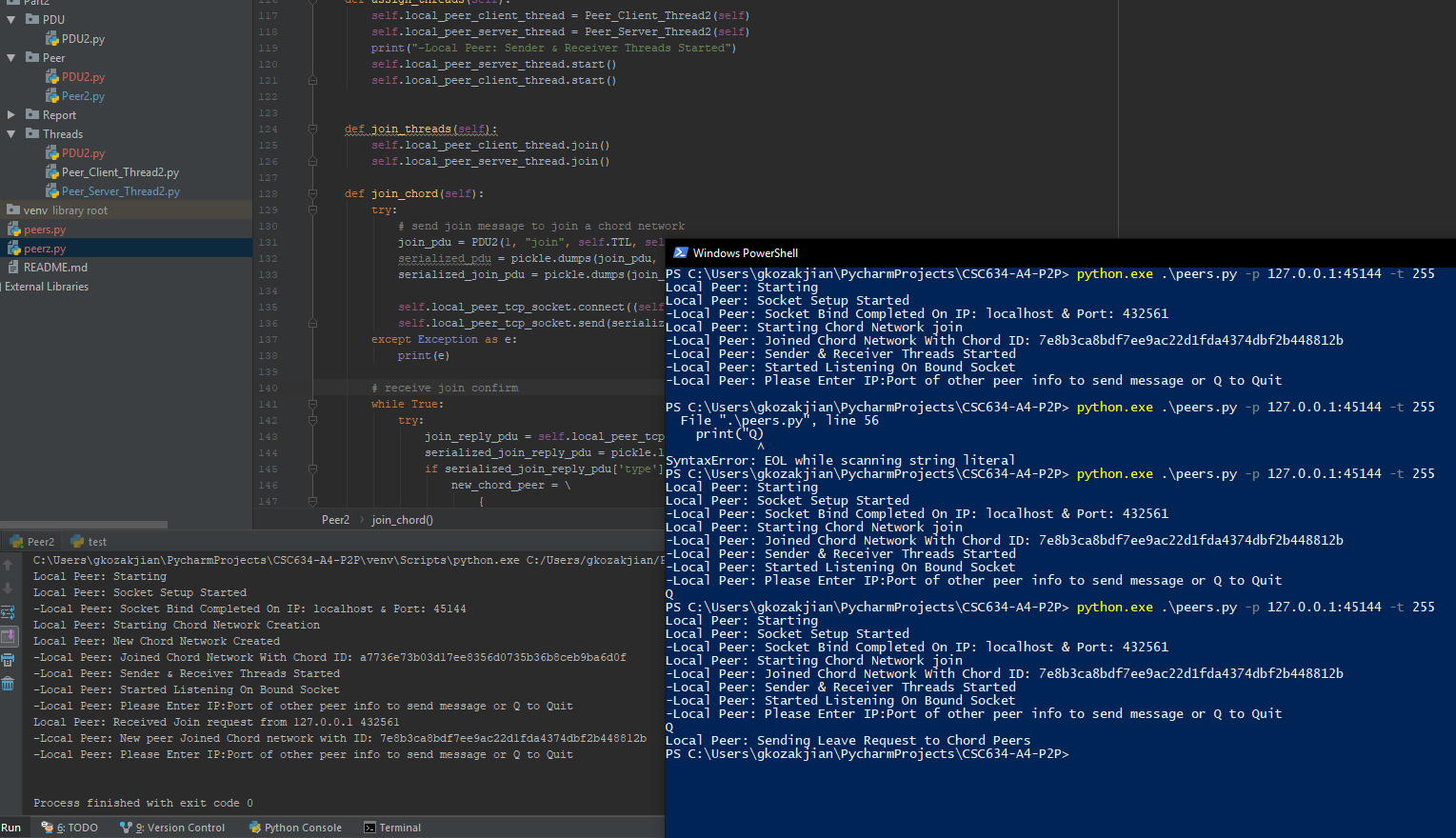


# **Results**

The implementation transfers any type of file very efficiently and reliably. It has been tested so far for small number of nodes on a not so powerful machine s. This implementation is able to create a small chord p2p network of peers and nodes and oh boy this days p2p is hard ...

In the picture below, we can see transfer of an image file. Since we are using blocks of size 1024 bytes, we can see that file transfer was quick and efficient.

In this third test case, we have an error rate of 10 % of total transmissions. So every 9th transmission will be timeout, and client will repeat the request. We can see in the output that the client timeout is being

optimized and increased to 4 rather than 10 at end of each cycle, since the dynamic timeout optimization algorithm is running and reducing timeout to 1 every 4 cycles of good transmissions which will occur, which is keeping the transmission running. We can see transmission was successful and verify file integrity .

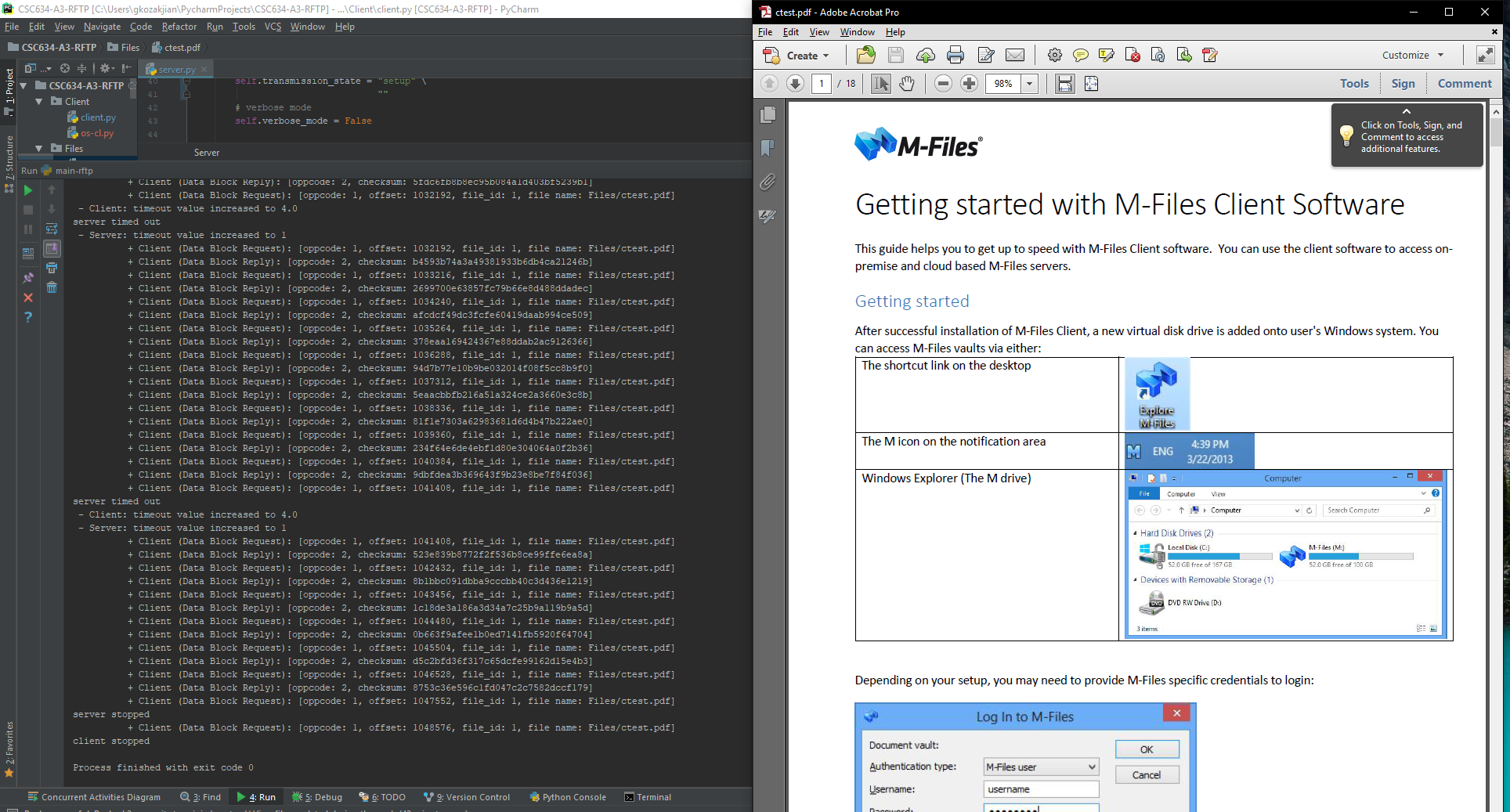
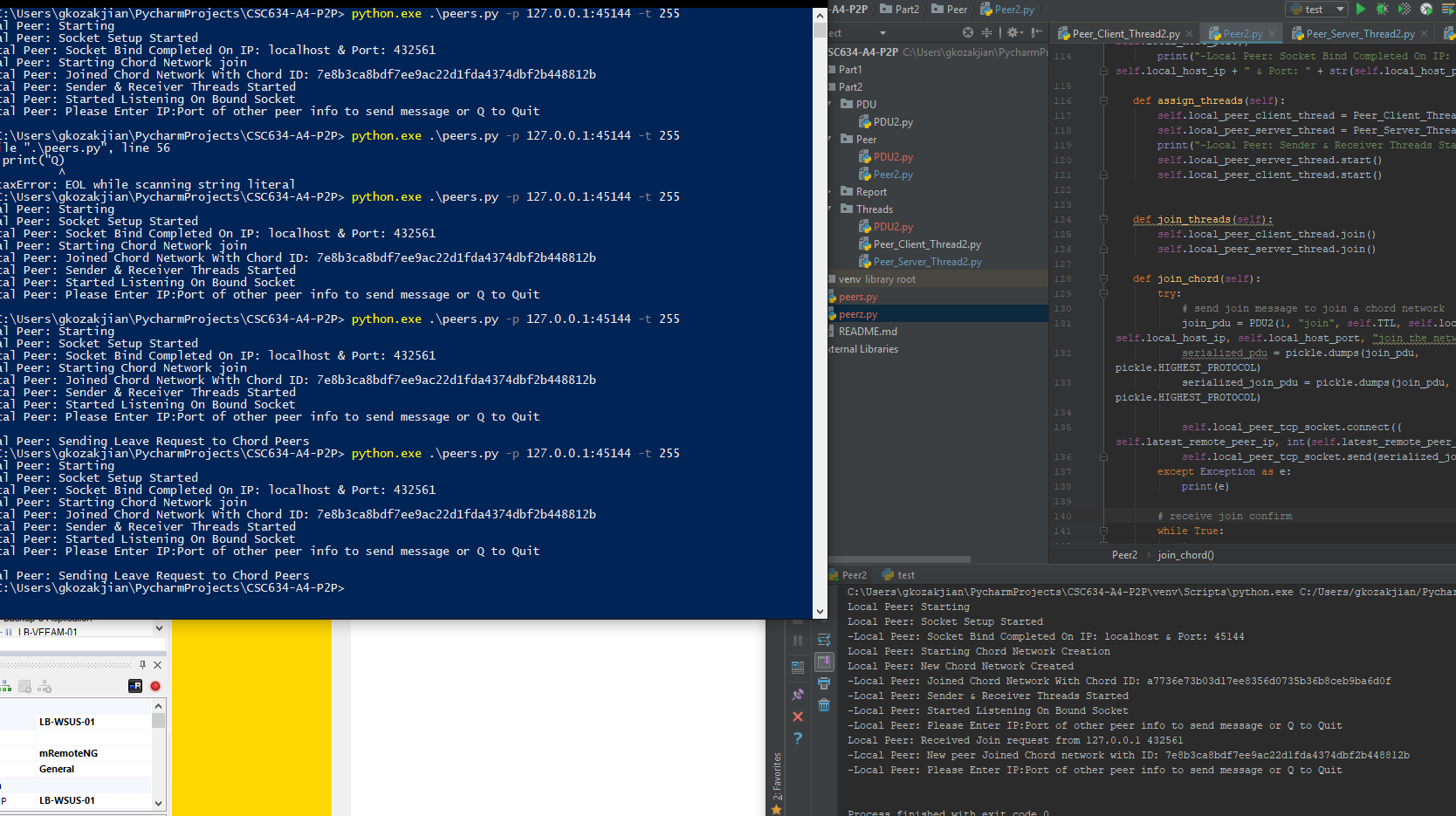


Figure 7: Error rate of 10%

In this final example, we can see that the error rate is more than 30%, so the client timeout value will be as high as 10, and it will stop, causing the transmission to stop as well.



# **Limitations**

## **Bidirectional communication between peers**

In this implementation, when a peer wants to send a message, it goes into blocking mode, not allowing it to receive as requested in the problem. This might be an issue for applications that require it like file transfer, etc.

## **Not very reliable code**

The implementation did not use exception handling well, hence allot of problems may emerged when testing it in real-life or development environments. Also, custom exceptions are not used here, which may make it easier to understand errors at occurrence.

## **Security**

This implementation of chord does not address security in any form. Hence, the transmitted data is in clear text, and can be sniffed and re-constructed by some intruder.

To address the security, encryption of data can be used with SSL sockets. to address authentication, password can be used or certificates by peers.

# **Improving the protocol**

This implementation can be improved by extensive testing in real-life environments. Areas of improvement might also include a better graphical interface, more multithreading for each peer connection, dividing the route chord into 8 or 16 divisiions so that the time to join or forward some packet goes down from O(n) to O(llnn).