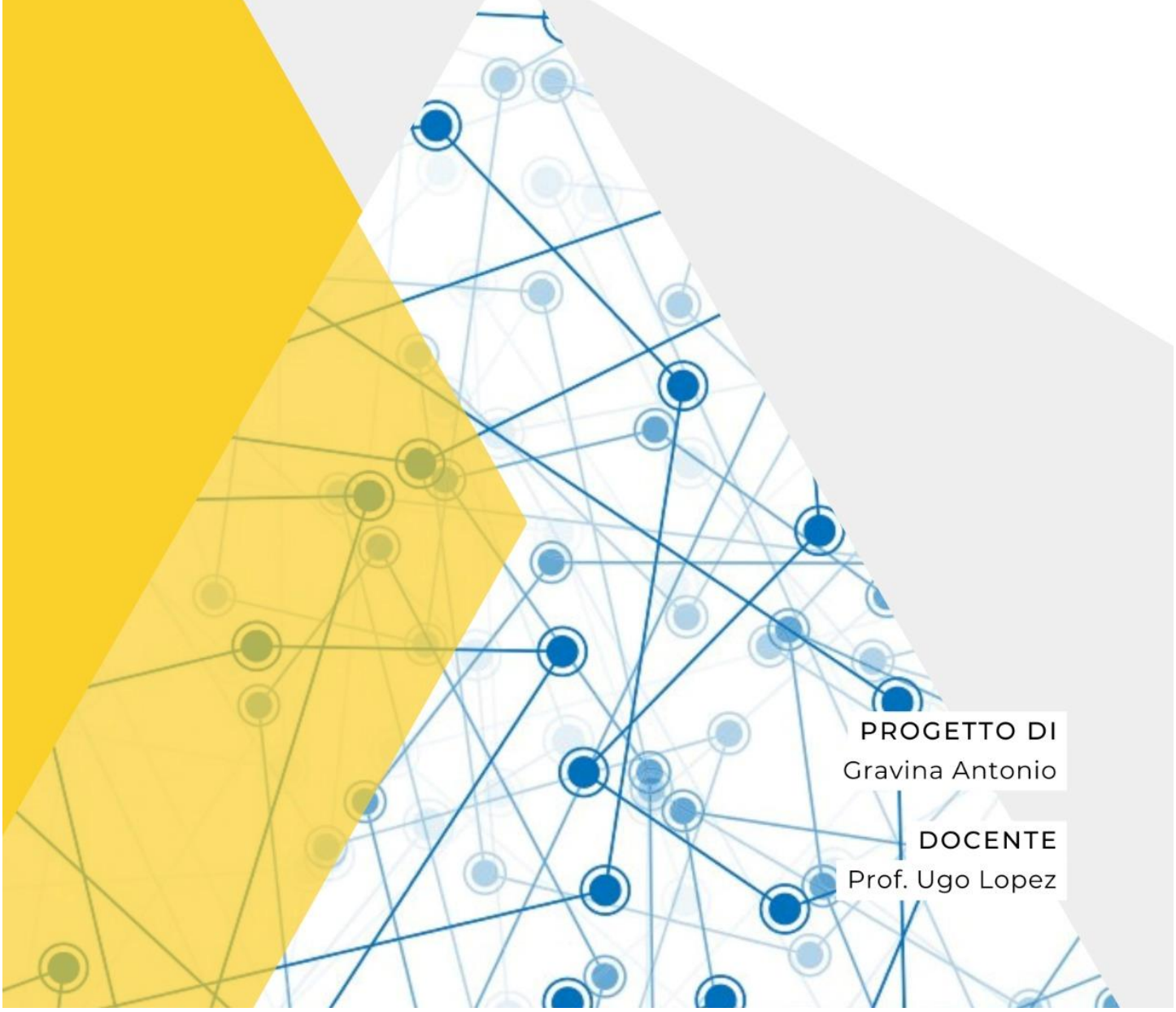
PROGETTO RETI DI CALCOLATORI

A.A 2022/2023



PROGETTO DI

Gravina Antonio

DOCENTE

Prof. Ugo Lopez

UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Dipartimento di Informatica

Corso di Laurea in

Informatica e Comunicazione Digitale

RETI DI CALCOLATORI E COMUNICAZIONE DIGITALE

Studente: Antonio Gravina

Matricola: 735584

Docente: Prof. Ugo Lopez

A.A: 2022/2023

Sommario

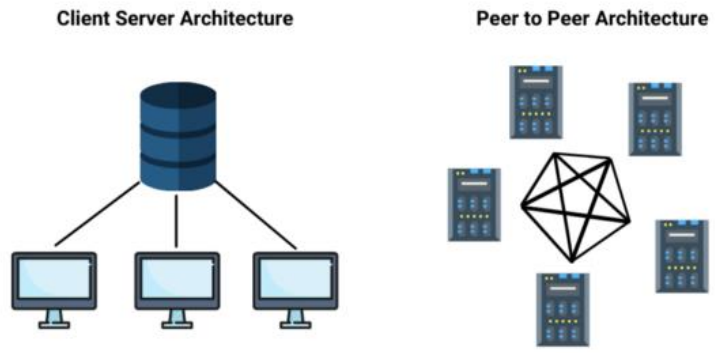
Per visualizzare la presentazione interattiva clicca gui.

1. Introduzione

Nel corso degli ultimi decenni la società è cambiata profondamente, principalmente a causa della diffusione di nuovi mezzi di comunicazione capaci di connettersi ad Internet e che consentono la trasmissione di informazioni in modo rapido ed efficace. Un insieme di dispositivi interconnessi fra loro capaci di scambiare dati e risorse prende il nome di **rete**.
Tra le architetture di rete più diffuse vi è sicuramente la *client-server*, caratterizzata da dispositivi chiamati client che effettuano delle richieste ai server, i quali hanno il compito di elaborare queste richieste e di inviare i risultati ai client. Questo modello di rete è alla base di molti servizi e web application, come Facebook, Instagram o YouTube.
È necessario, tuttavia, citare un modello di rete meno diffuso ma altrettanto importante: il **peer-to-peer** (P2P).

Le reti P2P sono caratterizzate dalla presenza di nodi (**peers**) interconnessi tra loro, in grado di condividere risorse senza un server che gestisce le richieste, a differenza delle reti client-server. Il modello P2P è quindi decentralizzato, perché consente una comunicazione diretta tra i nodi.
Ogni peer ha quindi un duplice comportamento:

- *Come client* quando richiede a un altro nodo una risorsa;
- *Come server* quando riceve una richiesta da un altro nodo ed invia la risorsa.

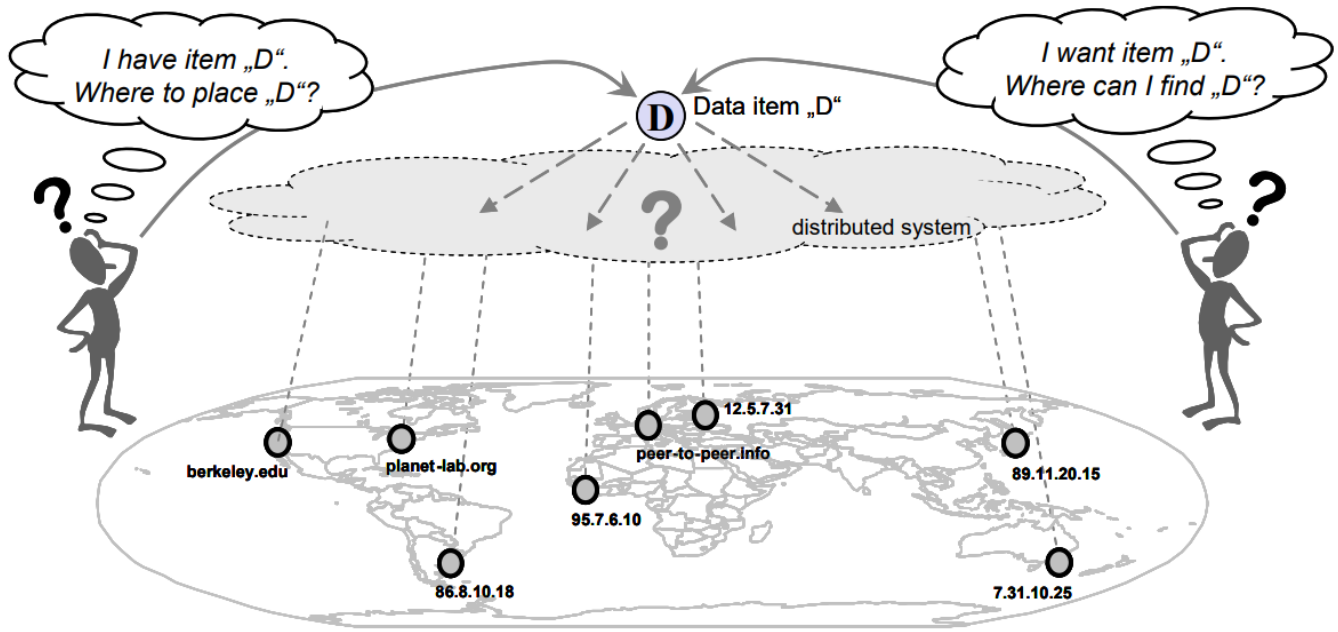


Di seguito si elencano i vantaggi e gli svantaggi principali nell'utilizzo del modello P2P rispetto al client-server:

Vantaggi	Svantaggi
Nelle P2P un numero maggiore di nodi connessi alla rete non causa il cedimento della stessa, mentre nelle reti client-server i server potrebbero non riuscire a gestire tutte le richieste dei client, se numerosi.	Ridotta efficienza a causa dell'operazione di discovery.
La mancata centralizzazione della rete permette una maggiore sicurezza e privacy degli utenti, in quanto non vi è nessun ente centrale che monitora le loro attività.	Bassa velocità di download delle risorse nel caso in cui ci siano pochi peer connessi alla rete.
Per l'azienda che fornisce servizi basati sul P2P i costi di gestione e manutenzione sono ridotti.	I servizi di P2P filesharing potrebbero essere utilizzati impropriamente, ad esempio per la diffusione di virus o di materiale protetto da copyright.

2. Tipologie di reti P2P

Per poter funzionare, una rete P2P prevede un meccanismo che consente ai nodi di vedersi tra loro e di potersi connettere quando vogliono. Questa fase prende il nome di *discovery*, in cui un nodo non è ancora connesso a nessun altro.



In base al meccanismo di discovery utilizzato, definiamo varie tipologie di rete P2P:

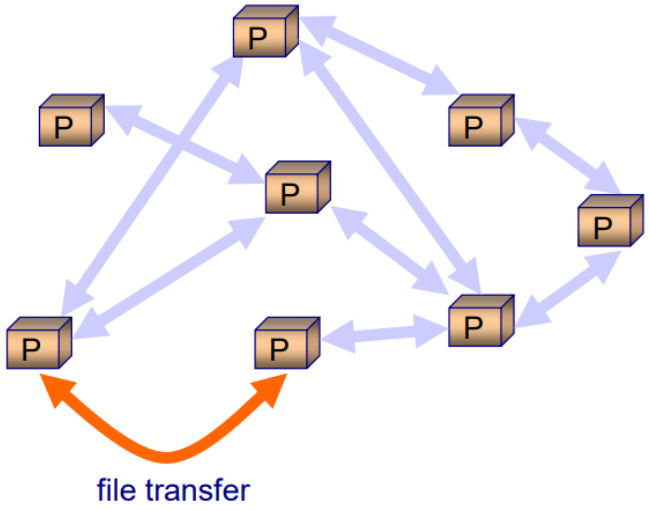
- **Reti P2P non strutturate**, in cui le connessioni tra nodi sono generate casualmente;
- **Reti P2P strutturate**, in cui i nodi riescono ad interagire attraverso un'architettura ben strutturata, ad esempio mediante l'utilizzo di tabelle hash;
- **Reti P2P ibride**, caratterizzate dalla presenza di un discovery server che consente ai nodi di vedersi all'interno della rete.

2.1. Reti non strutturate

Le reti P2P non strutturate sono quelle più decentralizzate, in cui ogni nodo è allo stesso livello degli altri. Questo modello è quindi considerato il "P2P puro". Infatti, non esiste una struttura gerarchica o una suddivisione di ruoli tra nodi.
Il limite principale di quest'architettura è il peer discovery: poiché non ci sono server o nodi dedicati, ogni peer che entra in rete deve necessariamente avere un *elenco* contenente la locazione degli altri peer della rete (non necessariamente tutti). In questo modo, quando il nodo entra in rete, chiede agli altri nodi presenti nell'elenco di condividere informazioni sugli altri peer presenti nei loro elenchi: avviene quindi uno scambio mutuale di dati, in modo che alla fine tutti i nodi della rete possano vedersi e condividere risorse.

Un altro metodo, applicabile solo in teoria ma totalmente inefficiente, è l'*IP scanning*: un nodo appena entrato in rete invia richieste ad altri possibili nodi tentando di stabilire connessioni con tutti gli indirizzi IP presenti entro

un certo intervallo. Se si riceve una risposta vuol dire che si è stabilita una connessione con un certo nodo (anch'esso appartenente alla rete), con il quale saranno condivise informazioni sugli altri nodi conosciuti.
Il problema principale di questo metodo è, come già affermato in precedenza, l'inefficienza. Inoltre, alcuni nodi potrebbero essere protetti da firewall che impediscono di rispondere agli altri nodi. Infine, l'utilizzo di questa tecnica può generare una quantità eccessiva di traffico di rete che potrebbe comportare ad un blocco da parte dell'ISP (nel caso in cui ci sia un overlay network sulla rete Internet).



Di seguito si elencano i vantaggi e gli svantaggi principali nell'utilizzo del modello P2P non strutturato:

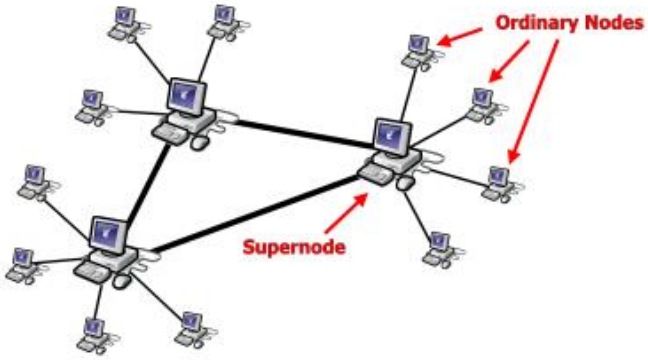
Vantaggi	Svantaggi
Mancanza di un single point of failure, perché il funzionamento della rete non dipende dalla presenza di particolari nodi o di server.	Il tempo di discovery è notevolmente maggiore rispetto ad altri modelli di rete P2P.
Maggiore flessibilità, in quanto gli utenti possono collegarsi e scollegarsi quando vogliono.	Maggiore difficoltà per l'ottenimento di risorse specifiche.
Minori costi, perché non ci sono server da mantenere.	Difficoltà di implementazione di algoritmi di discovery.
Maggiore privacy, perché senza un server è difficile tracciare tutti i nodi presenti in rete e le loro attività.	- - -

Fra i **software** più popolari che adottano questo modello troviamo GNUnet, una rete open-source utilizzata per comunicazioni anonime e sicure, e Freenet.

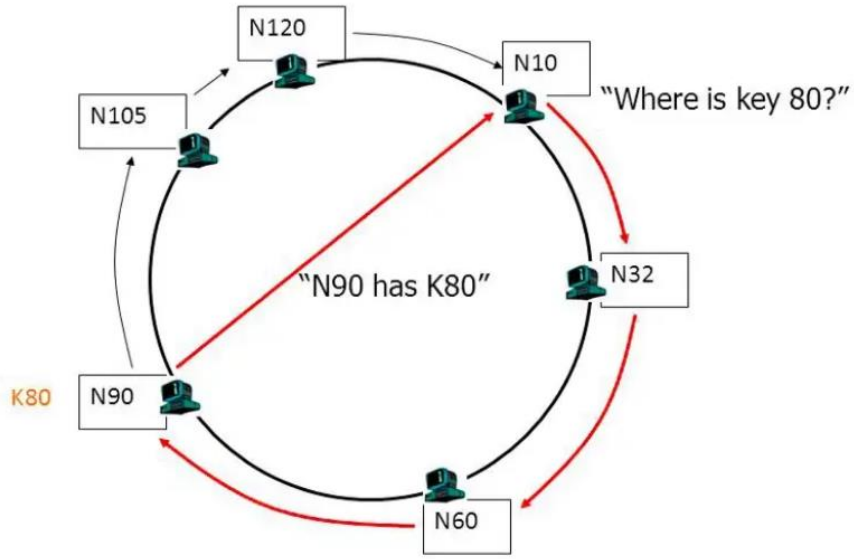
2.2. Reti strutturate

Nelle reti P2P strutturate, i nodi seguono una certa gerarchia: alcuni di essi sono detti "nodi di routing" o "supernodi" e hanno ruoli speciali. Ad esempio, quando un nodo si connette a un supernodo, quest'ultimo invia informazioni su come trovare altri nodi, fornendo un elenco di tutti gli altri peer presenti in rete. L'indirizzo IP e la porta dei supernodi sono già definiti nei file di configurazione del software.

Si ricorda che i supernodi sono, di fatto, dei nodi della rete con ruoli speciali, per cui sono anch'essi in grado di condividere file e di inviare richieste ad altri peer.



Molte delle reti P2P strutturate hanno un meccanismo basato sulle tabelle hash per consentire ai nodi di vedere più velocemente i peer che possono condividere la risorsa voluta. Questo tipo di rete si chiama **P2P con DHT** (Distributed Hash Table).
In quest'architettura, ogni nodo ha una copia della tabella hash, in cui sono presenti informazioni riguardo i file condivisi sulla rete e dove essi si trovano (ossia quali sono i peer a cui chiedere la risorsa desiderata). Per cui, quando un nodo vuole trovare un certo file all'interno della rete, deve calcolare l'hash utilizzando l'algoritmo comune per tutti i peer della rete, per poi ricercare questo file nella tabella: se esso è presente, vuol dire che il nodo ha già quella risorsa, oppure sa dove trovarla; se invece l'hash non è presente in tabella, il nodo chiede ad altri peer in rete se hanno il valore nella loro tabella. In caso affermativo, può ottenere il file da loro.



In sistemi come BitTorrent, ogni nodo possiede una copia della DHT, a sua volta caratterizzata (come tutte le tabelle hash) da coppie chiave-valore. In particolare, la chiave identifica un contenuto, mentre i valori associati sono gli indirizzi IP dei nodi che stanno condividendo quel file. In reti P2P DHT non esistono dei veri e propri supernodi, ma è comunque presente un certo livello di organizzazione fra i nodi.
Grazie alla DHT, infatti, i nodi riescono a scoprirsi rapidamente, senza aver bisogno di server trackers.

Di seguito si elencano i vantaggi e gli svantaggi principali nell'utilizzo del modello P2P strutturato:

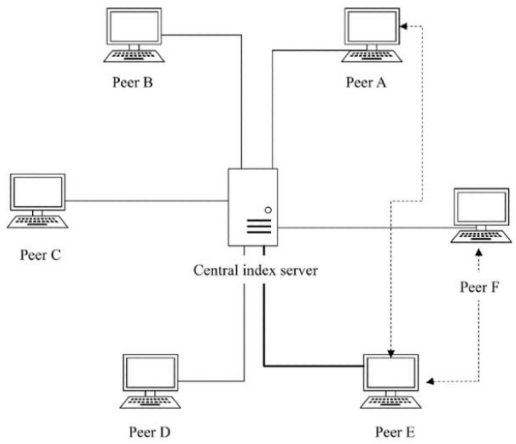
Vantaggi	Svantaggi
Minor tempo di discovery rispetto alle reti P2P non strutturate.	Non è un modello P2P "puro" a causa della presenza dei supernodi.
Maggiore efficienza rispetto alle reti P2P non strutturate.	I supernodi sono dei point of failure.

Fra i **software** che implementano questo modello troviamo Skype (fino al 2014), in cui ogni utente con una larghezza di banda sufficientemente ampia poteva svolgere il ruolo di supernodo. In realtà, Skype presentava anche dei server, ma dedicati unicamente al login degli utenti.

2.3. Reti ibride

Nelle reti P2P ibride, oltre ai nodi è presente anche un server che consente il discovery dei peer chiamato, appunto, **discovery server** (il cui IP + porta sono memorizzati nei file di configurazione del software). Questi server si occupano soltanto di aiutare i nodi a connettersi, per cui, pur essendo un sistema più centralizzato rispetto alle reti P2P strutturate e non strutturate, esso può comunque ritenersi di tipo peer-to-peer.
Appena un nodo si connette alla rete, esso richiede al discovery server l'elenco di tutti gli altri peer connessi. Inoltre, il discovery server ha il compito di aggiornare questo elenco ogni qualvolta che un nuovo nodo si connette, oppure quando avviene una disconnessione.

Per questo tipo di rete può bastare un solo server, perché non deve gestire un numero eccessivo di richieste.
Esiste anche il **lookup discovery server**, che condivide anche informazioni sulle risorse che i nodi possono condividere all'interno della rete.
Questo tipo di architettura è ottimale per applicazioni in cui è previsto un sistema di account, in quanto il discovery server può collegarsi a un database in cui sono memorizzate le credenziali degli utenti.
Attualmente, i modelli ibridi hanno performance migliori rispetto agli altri due modelli, perché presentano sia i vantaggi tipici delle reti P2P, sia quelli delle reti client-server.



Di seguito si elencano i vantaggi e gli svantaggi principali nell'utilizzo del modello P2P ibrido:

Vantaggi	Svantaggi
Maggiore efficienza di discovery rispetto a sistemi strutturati e non.	Non è un modello P2P "puro" a causa della presenza del discovery server.
Maggiore efficienza di file-sharing grazie al lookup discovery.	Se c'è solo un discovery server, esso è un single point of failure.
Possibilità di integrazione di account management system.	Maggiori costi di manutenzione del server.
Migliore organizzazione della rete.	- - -

Fra i **software** più popolari che utilizzano questo modello troviamo Spotify (fino al 2014) ed eMule.

3. Applicazioni e problematiche del modello P2P

Di seguito una descrizione delle applicazioni del modello P2P in contesti reali e delle problematiche riscontrabili.

3.1. Contesti d'uso

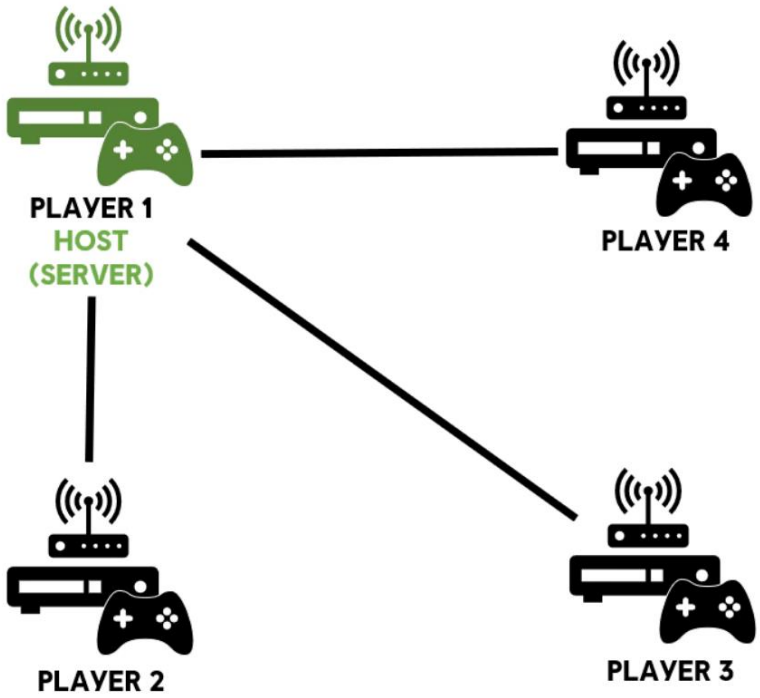
Il modello P2P è usato prevalentemente in software di messaggistica e di file sharing. Poiché non esiste un server che memorizza le attività degli utenti, questi software possono risultare molto utili per eludere la censura in paesi in cui vige un regime autoritario. Infatti, in alcuni paesi, siti come YouTube e Wikipedia non sono accessibili e le attività degli utenti sono costantemente monitorate (per via di accordi presi tra il governo e l'ISP) per impedire la condivisione di contenuti che potrebbero destabilizzare il regime.
Infatti, essendoci un'architettura decentralizzata, non vi è nemmeno un singolo punto di controllo utilizzabile dalle autorità. Inoltre, tra i nodi potrebbe vigere una criptazione end-to-end dei contenuti condivisi, rendendo ancora più difficile il tracciamento dei dati.

Fra i software più utilizzati da coloro che vogliono eludere la censura vi è I2P (Invisible Internet Project), che garantisce anonimato durante la navigazione in Internet, oppure Briar, un'app di messaggistica istantanea caratterizzata da un'architettura decentralizzata, criptazione dei contenuti e altre misure anticensura. Briar è diversa dalle classiche applicazioni di messaggistica, in quanto può collegarsi alla rete Tor e può funzionare anche offline affidandosi al Bluetooth per collegarsi con altri utenti in prossimità.
Questo sistema è utilizzato anche da Apple con 'Trova il mio iPhone': a partire da iOS15, se il telefono smarrito è spento o offline, viene instaurata una rete P2P tra iPhone nelle vicinanze, comunicanti tra loro tramite Bluetooth. Così, quando l'iPhone smarrito è in prossimità di questa rete, si aggancia ad essa e comunica la propria posizione agli altri nodi, che inoltrano quest'informazione al proprietario.

Il protocollo P2P è usato anche per l'implementazione di criptovalute, Bitcoin fra tutte. Infatti, nel momento in cui avviene una transazione, i dati relativi a quest'operazione sono condivisi tra i nodi della rete, che devono quindi validare la transazione per poi aggiungerla alla blockchain. È fondamentale il concetto di **consenso**: affinché un nuovo blocco (cioè i dati della transazione) possa essere aggiunto alla catena, la metà più uno dei nodi della rete deve averlo validato.
La decentralizzazione delle valute digitali porta con sé diversi vantaggi, come una maggiore trasparenza (in quanto tutte le transazioni sono visibili sulla blockchain da chiunque); tuttavia, le criptovalute sono ad oggi

considerate solo come fonte di guadagno per investitori, e non come una tecnologia realmente utile per i cittadini, anche se in futuro tecnologie del genere saranno sempre più diffuse.

Il P2P può anche essere utilizzato per i videogiochi online: nei giochi che adottano questo modello, ogni dispositivo assume il duplice ruolo di client/server, eliminando la necessità di server centralizzati. In questo modo si favorisce una maggiore robustezza, in quanto non c'è la possibilità di server intasati, ma anche minori costi per gli sviluppatori perché non è richiesta la manutenzione dei server. Inoltre, vi è spesso una minore latenza, in quanto vi è una diretta comunicazione tra i giocatori. Infine, le aziende chiudono i server per videogiochi molto vecchi, impedendo ai (pochi) giocatori rimasti di poter usufruire del multiplayer: ciò non accade per giochi che adottano il P2P.
Fra i giochi più popolari vi è Warframe, in cui un membro della lobby è eletto come host, oppure Fifa, in cui è possibile giocare in P2P con altri utenti (anche se in molte modalità di gioco si fa affidamento ai server EA).



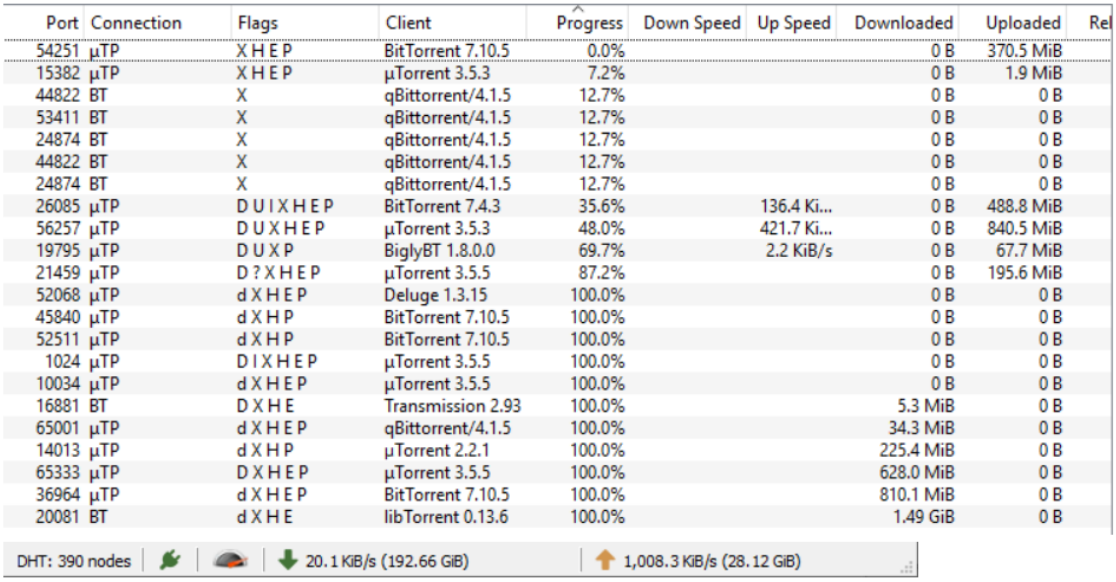
3.2. Implicazioni legali

L'architettura di rete P2P è utilizzata per applicazioni di file sharing. Nel 2005 nacque il protocollo BitTorrent, di tipo P2P con discovery server detti **trackers**: con BitTorrent è possibile agganciarsi a una rete di nodi che condividono un determinato file; ogni nodo invia un segmento del file completo agli altri, in quanto il file originale è frammentato in tanti piccoli pezzi. È fondamentale scaricare il corrispondente file con estensione .torrent che descrive la suddivisione del file nei vari pacchetti e inoltre contiene l'indirizzo IP di uno o più server chiamati tracker, utili per il discovery e per la gestione delle attività dei vari nodi che stanno condividendo la risorsa in quel momento.

Nell'ambito di BitTorrent è fondamentale definire due tipi di nodi:

- *Seeders*, cioè utenti che hanno terminato il download del file, ma continuano l'upload per permettere agli altri utenti della rete di scaricare il file più velocemente;
- *Leechers*, ossia coloro che stanno ancora scaricando il file, ma al tempo stesso eseguono l'upload dei pacchetti ottenuti fino a quel momento. Precedentemente al protocollo BitTorrent, con "leecher" si definiva un utente che scarica il file ma non lo condivide, ma con BitTorrent è costretto all'upload.

Durante la condivisione di un file è possibile ottenere informazioni di altri peer come l'indirizzo IP, la porta, il software utilizzato, la velocità di upload/download e la quantità di dati caricati/scaricati.



Port	Connection	Flags	Client	Progress	Down Speed	Up Speed	Downloaded	Uploaded	Rel
54251	µTP	X H E P	BitTorrent 7.10.5	0.0%			0 B	370.5 MiB	
15382	µTP	X H E P	µTorrent 3.5.3	7.2%			0 B	1.9 MiB	
44822	BT	X	qBittorrent/4.1.5	12.7%			0 B	0 B	
53411	BT	X	qBittorrent/4.1.5	12.7%			0 B	0 B	
24874	BT	X	qBittorrent/4.1.5	12.7%			0 B	0 B	
44822	BT	X	qBittorrent/4.1.5	12.7%			0 B	0 B	
24874	BT	X	qBittorrent/4.1.5	12.7%			0 B	0 B	
26085	µTP	D U I X H E P	BitTorrent 7.4.3	35.6%		136.4 Ki...	0 B	488.8 MiB	
56257	µTP	D U X H E P	µTorrent 3.5.3	48.0%		421.7 Ki...	0 B	840.5 MiB	
19795	µTP	D U X P	BiglyBT 1.8.0.0	69.7%		2.2 KiB/s	0 B	67.7 MiB	
21459	µTP	D ? X H E P	µTorrent 3.5.5	87.2%			0 B	195.6 MiB	
52068	µTP	d X H E P	Deluge 1.3.15	100.0%			0 B	0 B	
45840	µTP	d X H P	BitTorrent 7.10.5	100.0%			0 B	0 B	
52511	µTP	d X H P	BitTorrent 7.10.5	100.0%			0 B	0 B	
1024	µTP	D I X H E P	µTorrent 3.5.5	100.0%			0 B	0 B	
10034	µTP	d X H E P	µTorrent 3.5.5	100.0%			0 B	0 B	
16881	BT	D X H E	Transmission 2.93	100.0%			5.3 MiB	0 B	
65001	µTP	d X H E P	qBittorrent/4.1.5	100.0%			34.3 MiB	0 B	
14013	µTP	d X H P	µTorrent 2.2.1	100.0%			225.4 MiB	0 B	
65333	µTP	D X H E P	µTorrent 3.5.5	100.0%			628.0 MiB	0 B	
36964	µTP	d X H E P	BitTorrent 7.10.5	100.0%			810.1 MiB	0 B	
20081	BT	d X H E	libTorrent 0.13.6	100.0%			1.49 GiB	0 B	

DHT: 390 nodes | 20.1 KiB/s (192.66 GiB) | 1,008.3 KiB/s (28.12 GiB)

Seppur l'ideologia che sta alla base di sistemi come BitTorrent sia nobile (perché si vuole raggiungere un web libero, senza monopoli), spesso le applicazioni che utilizzano questo protocollo sono utilizzate per la condivisione di materiale coperto da copyright. Quest'attività è illegale, ma a causa della privacy garantita dai sistemi P2P è difficile per le autorità risalire ai primi nodi che hanno iniziato a diffondere il file.
Di solito, ad essere legalmente perseguibili non sono i nodi della rete che stanno condividendo il file, ma sono coloro che diffondono in primis il file .torrent che permette agli utenti di agganciarsi alla rete stessa. È questo il caso di **TNTVillage**, un forum (ormai chiuso) in cui gli utenti potevano condividere i file .torrent per scaricare musica, film e videogiochi.

TNTVillage nacque nei primi anni duemila, quando il modello P2P iniziò a diffondersi. Il concetto cardine era lo "scambio etico" del materiale, come affermato dal fondatore della piattaforma, Luigi Di Liberto. Egli propose una riforma del concetto di copyright, sostenendo per il decadimento della protezione di un contenuto dopo un anno esatto dalla sua pubblicazione.
TNTVillage non ha mai hostato sui suoi server file protetti da copyright, ma era piuttosto un motore di ricerca, unico nel suo genere (anche in ambito internazionale). Si stima un totale di 300 mila opere presenti sulla piattaforma, prima della sua chiusura avvenuta nel 2019. Infine, nel settembre 2022, il Tribunale di Milano ha disposto la rimozione di tutti i contenuti del sito.

Un altro utilizzo improprio di reti P2P riguarda la distribuzione di file governativi: un esempio è quello di **Chelsea Manning**, ex militare degli Stati Uniti, condannata a 35 anni di carcere per aver violato l'Espionage Act nel 2013. La donna fu scarcerata dopo sette anni per grazia di Obama. Manning aveva caricato documenti governativi segreti sulla piattaforma WikiLeaks e su piattaforme di file-sharing P2P che avrebbero dovuto garantire l'anonimità.
Oltre ai file governativi, spesso sono distribuiti anche file illegali attraverso software come **Tor** (The Onion Router) che permettono agli utenti di navigare in anonimato.
Utilizzando Tor per connettersi ad Internet, il traffico è criptato ed è indirizzato attraverso una serie di nodi (grazie a volontari da tutto il mondo). Ogni nodo conosce la posizione di quello precedente e di quello successivo della catena.
La garanzia di anonimato è il principale motivo per cui molti criminali si affidano a questi strumenti per poter svolgere le loro attività illecite sulla rete.

3.3. Problemi di sicurezza

Le reti P2P, come già detto nei capitoli precedenti, garantiscono un certo livello di privacy per i peer che ne fanno parte. Tuttavia, bisogna anche valutare eventuali problemi di sicurezza che potrebbero verificarsi.
All'interno della rete potrebbero esserci dei nodi malevoli che distribuiscono file dannosi che, se scaricati, potrebbero compromettere il dispositivo della vittima. Le principali minacce in questo ambito sono:

- **Ransomware**, ossia dei malware che criptano tutti i dati del dispositivo, per cui la vittima è costretta a pagare un riscatto (spesso in criptovalute) per poter riavere i propri dati;
- **Trojan**, che permettono all'attaccante di utilizzare in remoto il dispositivo della vittima (reverse shell)
- **DOS** (Denial Of Service): questo attacco consiste nell'invio di migliaia di richieste a un nodo della rete in modo da mandare il dispositivo bersaglio in crash.

Per risolvere queste problematiche è fondamentale che i nodi nella rete abbiano antivirus e firewall attivi. In generale, è opportuno che la rete sia strutturata in modo da mitigare possibili minacce.
La soluzione ideale sarebbe l'implementazione di un sistema di ban per gli utenti che rappresentano una minaccia per la sicurezza della rete, e quindi anche un sistema di accounting che permetta registrazione e login dei nodi.
Per attuare questa soluzione è opportuna la presenza di nodi "speciali" che si occupino di gestire gli accessi degli utenti alla rete, quindi bisogna utilizzare il modello P2P strutturato. Tuttavia, i supernodi dovrebbero necessariamente consultarsi ogni qual volta avviene un nuovo login/registrazione/ban, in modo da aggiornare i propri database. Per cui è più efficiente utilizzare un discovery server (quindi modello P2P ibrido) che si occupi non solo della scoperta dei nodi, ma anche di gestire il sistema di accounting implementato.

Nel capitolo successivo si descrive la realizzazione di un sistema di livechat e file-sharing in P2P ibrido con discovery server con gestione del sistema di accounting.

4. Realizzazione di un software di messaggistica e file-sharing in P2P

Il sistema realizzato è un software di messaggistica e file sharing tra nodi utilizzando il **modello P2P ibrido**, in cui è presente un discovery server. È stato scelto questo modello perché è più vantaggioso rispetto al P2P strutturato e non strutturato. Infatti, il discovery server si occuperà non solo di permettere ai nodi di vedersi, ma anche di gestire un sistema di accounting. Infatti, ogni utente dovrà registrarsi per poter utilizzare il servizio, per cui la presenza del discovery server è utile per la creazione di nuovi account e per l'autenticazione.

Le credenziali sono salvate in un database (il DBMS utilizzato è PostgreSQL) e la password è criptata (mediante Bcrypt). Il server gestisce anche un sistema di report di utenti, in modo da poter "bannare" un profilo nel caso in cui abbia ricevuto delle segnalazioni da parte di altri utenti.
La comunicazione tra i nodi avviene in P2P.

Per la realizzazione di un software di messaggistica e file sharing P2P sono stati utilizzati i seguenti strumenti.

- Linguaggio di programmazione: Python versione 3.11.1;
- IDE: Visual Studio Code versione 1.74;
- DBMS: PostgreSQL 14;
- Piattaforma di Version Control: GitHub;
- Sistema Operativo: Windows 10.

Prima di procedere con l'analisi del progetto è opportuno definire le motivazioni per cui è stato scelto Python come linguaggio utilizzato:

- Semplicità di utilizzo, in quanto linguaggio ad alto livello;
- Presenza di numerosi moduli standard necessari per poter sviluppare il software;
- È uno dei linguaggi più versatili, in quanto adoperato in numerosi campi della Computer Science (dalla data science al socket programming).

Il progetto è costituito da quattro file .py, di cui tre sono utilizzati dal server:

- *account.py* per il login e la registrazione degli account;
- *connDB.py* per la comunicazione tra il server e il database, quindi per le varie operazioni che devono essere eseguite sul database;
- *discoveryServer.py*, cioè lo script per avviare o stoppare il server, gestire la connessione/disconnessione dei nodi, nonché per inviare loro l'elenco di tutti gli altri nodi presenti in rete.

Per gli utenti è invece richiesta l'esecuzione di un solo script di nome *connNode.py* usato per connettersi al discovery server per poi collegarsi agli altri nodi della rete ed inviare loro messaggi/file.

4.1. Socket programming in Python

Per poter creare un oggetto Socket in Python (fondamentale per l'invio/ricezione di file) bisogna innanzitutto importare il modulo socket, per poi inizializzare una variabile in questo modo: nodoA = socket.socket(family, type) dove family indica il tipo di indirizzo IP e può essere AF_INET (per IPv4) oppure AF_INET6 (per IPv6), mentre type definisce il protocollo di comunicazione utilizzato, che può essere SOCK_STREAM (per TCP) oppure SOCK_DGRAM (per UDP).

Una volta creato l'oggetto socket chiamato 'nodoA', possiamo legare il socket a una coppia IP + porta mediante la funzione socket.bind((host, port)). In questo modo abbiamo associato il socket a un nodo (che chiamiamo per semplicità 'nodo A'), che quindi può aspettare che altri nodi si collegano ad esso mediante la funzione socket.listen().
Quando un nodo B tenta di connettersi al nodo A, quest'ultimo può accettare il collegamento con la funzione socket.accept(), che restituisce due valori:

- *conn*, cioè un 'connection object', ossia un oggetto che rappresenta la connessione tra nodo A e nodo B;
- *addr*, cioè una tupla formata da indirizzo IP + porta del nodo B.

Per potersi connettere al nodo A, anche il nodo B deve essere associato a un socket che chiamiamo nodoB, utilizzando il proprio IP + porta specificata. Dopodiché, il nodo B usa la funzione socket.connect((host, port)) dove host e port sono l'IP e la porta del nodo A.
Ricapitolando ci sono:

- Nodo A, che aspetta che il nodo B si connetta e, nel momento in cui si instaura la connessione, riceve un connection object e la combinazione IP + porta del Nodo B;
- Nodo B, che si collega al nodo A (deve conoscere la coppia IP + porta di A), ma non riceve nessun valore di ritorno. Il nodo B, infatti, non ha bisogno né della combinazione IP + porta del nodo A (visto che già la conosce), e nemmeno del connection object (perché per gestire la comunicazione con il nodo A basta usare l'oggetto socket).

Mediante conn.send(data) e socket.send(data) è possibile inviare data, mentre con conn.recv(dimension) e socket.recv(dimension) è possibile ricevere dati. Dimension definisce la dimensione (in byte) dei singoli pacchetti in cui il file originale dev'essere scomposto.
Infine, mediante conn.close() e socket.close() si termina la connessione tra i due dispositivi.

È quindi evidente che il nodo A che chiama la funzione socket.listen() può ottenere più connection object (in base al numero di nodi che si connettono ad esso), per cui è in grado di gestire più connessioni con altri nodi. Invece, il nodo B può gestire la connessione solo con il nodo A, in quanto è dotato di un solo oggetto socket. Per poter connettersi ad altri dispositivi, il nodo B deve creare un altro socket utilizzando una porta diversa da quella già usata.
Il nodo A può quindi comportarsi come server, mentre il nodo B come client. Ciò è alla base del socket programming in linguaggio Python.

Ora che è stata definita la modalità di utilizzo dei socket, si può procedere con l'analisi dei file di progetto.

4.2. discoveryServer.py

In discoveryServer.py sono dichiarate le seguenti variabili globali costanti:

- PORT;
- SERVER ossia l'indirizzo IP del server;
- ADDR ossia la tupla IP + porta del server;
- FORMAT ossia il formato di criptazione dei dati;
- DISCONN_MESSAGE, cioè il messaggio che i nodi devono inviare per disconnettersi dal server;
- REPORT_MESSAGE per segnalare un utente;
- MAX_REPORTS che definisce il numero massimo di segnalazioni prima che un utente sia bannato;
- SHUT_COUNTDOWN che stabilisce qual è il tempo di attesa del server. Vuol dire che se al server è connesso alcun nodo entro il tempo stabilito, il server va automaticamente in shutdown;
- SEGMENT_LENGTH, cioè la dimensione in byte dei pacchetti.

Queste sono le funzioni presenti:

- newNode() pone il server in ascolto per nuove connessioni con dei nodi. Una volta accettata la connessione con un nuovo nodo, viene gestita l'autenticazione o la registrazione dell'account. Nel caso in cui ci sia la registrazione, viene creato un nuovo account (che viene poi memorizzato nel DB); nel caso di login, invece, si controllano che le credenziali inserite dall'utente coincidano con quelle presenti nel DB.
  Ogni nodo collegatosi al server è memorizzato in un set (struttura dati simile a un dizionario) per il tracking. Mediante i dizionari *connObjs* e *users* si associa la tupla (IP + porta) di ogni client collegato rispettivamente al corrispondente connection object e al corrispondente username.
  Per ogni nuovo utente autenticato si apre un nuovo thread per gestire la comunicazione client-server.
- nodeDisconnection(node, addr, username) serve per gestire la disconnessione di un nodo dal discovery server, il quale invia agli altri nodi collegati una lista aggiornata degli utenti connessi alla rete.
- userBan(userReported) è usata per notificare agli altri utenti il ban di un nodo, a seguito di segnalazioni ripetute.
- nodeHandling(node, addr, username) serve per l'effettiva gestione della comunicazione client-server. Per ogni nodo connesso al discovery server si apre un nuovo thread di esecuzione di questa funzione. Poiché ogni client può mandare messaggi al server, quest'ultimo esamina il messaggio ricevuto e risponde di conseguenza. Il server può ricevere quattro tipi di messaggi:
  - Messaggi non validi;
  - Messaggi di disconnessione da parte di un nodo;
  - Messaggi di report di un utente;
  - Messaggio di avvio di una conversazione tra due utenti. Per avviare una conversazione, è necessario inviare al server l'username dell'utente con cui si vuole comunicare.

  Nel caso in cui il server riceva un messaggio di tipo 4, invia al nodo che vuole iniziare la comunicazione e a quello con cui esso vuole entrare in contatto degli appositi 'messaggi flag'. Quindi, i due nodi si disconnettono dal server e avviene una connessione P2P tra di loro. Il server non svolge alcun ruolo mentre i due nodi stanno comunicando, per cui non viene memorizzato alcun dato relativo a questa comunicazione.

- serverShutdown() è una funzione avviata in parallelo con newNode(). Consiste in u ciclo while in cui si controlla il numero di nodi connessi al server. Se esso è 0 si aspettano x secondi (tempo definito da SHUT_COUNTDOWN) si verifica di nuovo il numero di client collegati. Se in questo periodo di tempo non si è connesso nessuno, il server si spegne da solo.
- main() che avvia i thread paralleli newNode() e serverShutdown().

4.3. connDB.py

Le variabili globali presenti in questo file sono:

- DB_HOST;
- DB_NAME;
- DB_USER;
- DB_PASS;
- conn per collegare Python a PostgreSQL mediante il modulo psycopg2;
- cur, cioè il cursore che punta alle righe della tabella (usato per le query);
- usersList;

In questo file è definita la classe User, che ha come attributi i campi della corrispondente tabella nel DB. I metodi presenti sono invece i getters (per ottenere gli attributi di un oggetto di classe User) e il setReports per modificare il numero di report di un utente.

Le funzioni in questo file sono:

- loadUsers() per caricare in una lista gli utenti memorizzati nel DB. Restituisce la suddetta lista e il numero di elementi che la compongono.
- insertUsers(username, password) crea un nuovo oggetto User e inserisce i corrispondenti valori nel DB. Inoltre, chiama loadUsers() per aggiornare la usersList.
- newReport(userReported) per aggiornare il campo 'reports' di una tabella nel momento in cui avviene una nuova segnalazione di un utente.
- closeConn() per interrompere il collegamento tra il DB e il server.

4.4. account.py

In questo file le variabili globali sono:

- FORMAT;
- MAX_REPORTS.

Questo script si occupa della gestione del login e della registrazione:

- login(username, password) verifica che le credenziali inserite dall'utente siano corrette. Inoltre, verifica che l'utente non sia stato bannato dal sistema (nel caso in cui abbia superato MAX_REPORTS).

- signup(username, password) che prima verifica che l'username non sia già stato utilizzato, e poi chiama la funzione insertUser di connDB per l'inserimento di un nuovo utente.

4.5. connNode.py

In connNode.py le variabili globali utilizzate sono:

- node, ossia il socket del nodo;
- DISCOV_SERVER, cioè l'IP del server;
- DISCOV_PORT;
- DISCOV_ADDR cioè la tupla (IP + porta) del server;
- FORMAT;
- DISCONN_MESSAGE;
- REPORT_MESSAGE;
- SEND_FILE_MESSAGE, usato tra i nodi per segnalare l'invio di un file;
- SEGMENT_LENGTH;
- serverMessages, un array contenente dei possibili messaggi inviati dal server;
- variabili booleane per il coordinamento dei thread.

Le funzioni sono:

- getFile(sock) per ricevere un file dal peer con cui si è in comunicazione, inserendo il percorso in cui salvare il file.
- sendFile(sock, msg) per mandare un file il cui percorso è specificato in msg.
- receiveFromPeer(sock, addr) è utile per ricevere un messaggio testuale dal peer. Ci sono quattro tipi di messaggi:
    - Messaggio vuoto, che causa disconnessione;
    - SEND_FILE_MESSAGE per iniziare l'invio di un file (il percorso dev'essere specificato) chiamando la funzione sendFile per chi invia e getFile per chi riceve.
    - DISCONN_MESSAGE per disconnettersi dal peer;
    - Qualsiasi altro messaggio testuale.
- sendToPeer(sock, addr) per mandare un messaggio testuale al peer. È eseguito in thread parallelo con receiveFromPeer.
- startHosting(guestAddr) utilizzato dal nodo che vuole instaurare un collegamento con un peer di cui conosce l'indirizzo (in quanto lo ha mandato il discovery server). Con questa funzione, il nodo si mette in ascolto per nuovi collegamenti. Se il peer che si connette ha lo stesso indirizzo di quello passato come parametro a startHosting, vuol dire che è quello giusto. Dopo l'instaurazione della connessione, partono in parallelo i thread receiveFromPeer e sendToPeer.
- connectedToHost(hostAddr) serve per collegarsi al peer in ascolto (l'indirizzo è quello passato come parametro). Dopo l'instaurazione della connessione, partono in parallelo i thread receiveFromPeer e sendToPeer.
- getMsg() è utilizzato per ottenere messaggi dal discovery server. Ci sono tre tipi di messaggi che possono ricevere:

    - Messaggi di aggiornamento della lista dei nodi connessi alla rete. Appena un nodo si connette alla rete ottiene l'elenco di username di tutti gli altri utenti online;
    - Messaggio di disconnessione dal server;
    - Messaggio di invito da parte di un altro utente per iniziare il collegamento P2P;
    - Messaggio di ban.
- setConn() serve per mandare messaggi al discovery server. Inserendo l'username di un utente connesso alla rete, il server inoltra a quell'utente il messaggio di invito. Conseguentemente, i due utenti (ossia i due nodi) si disconnettono dal server e iniziano a comunicare privatamente.
  Questa funzione è eseguita in parallelo con getMsg, perché ogni nodo deve gestire contemporaneamente sia l'invio che la ricezione di messaggi.
- main(), in cui si chiede all'utente il proprio indirizzo IP privato e la porta da utilizzare (se si inserisce 0 come valore, Python cercherà di ottenere l'IP in automatico, ma dipende dalla configurazione del dispositivo). Poi, si chiede se si vuole effettuare il login oppure la registrazione di un nuovo account, con conseguente richiesta di credenziali. Una volta fatto ciò, partiranno due thread paralleli per getMsg e setConn. Nel caso in cui ci si disconnetta dal discovery server, si esegue un controllo sulla causa della disconnessione: se l'utente ha inviato al server il DISCONN_MESSAGE il software si chiude; se invece la disconnessione è avvenuta perché l'utente ha instaurato un collegamento con un altro utente, allora partono i thread startHosting per chi instaura il collegamento e connectedToHost per chi riceve l'invito (ossia il guest).
  Infine, quando termina il collegamento tra due peer, essi possono riconnettersi al discovery server oppure terminare l'esecuzione del software.

Nota 1: per permettere al discovery server di ricevere richieste da parte di nodi connessi via Internet, è necessario andare nelle impostazioni del router e attivare il port forwarding, ossia il trasferimento dei dati da un dispositivo a un altro tramite una porta aperta del server.

Nota 2: l'analisi dei file del sistema è avvenuta seguendo un approccio bottom-up, cioè seguendo un ordine logico in cui sono definite per prima le funzioni più specifiche, fino ad ottenere quella più generale (il main). Per un'analisi top-down basta seguire l'ordine logico inverso, partendo dalla funzione main.

Nota 3: nel momento in cui si instaura un collegamento tra due peer, non c'è più differenza logica tra socket e connection object. Infatti, il peer che si mette in ascolto per il collegamento con l'altro peer non può ricevere connessioni da altri nodi (per come è strutturato il software), per cui quell'unico connection object che ottiene ha le stesse funzioni dell'oggetto socket dell'altro peer.

Nota 4: per questo software non è stata realizzata un'interfaccia grafica. Infatti, la realizzazione di una GUI per sistemi che utilizzano il multithreading richiederebbe risorse troppo dispendiose, specialmente se implementate sotto forma di web app utilizzando moduli come *Flask*.

4.6. Guida per l'installazione

Per utilizzare il software è necessario:

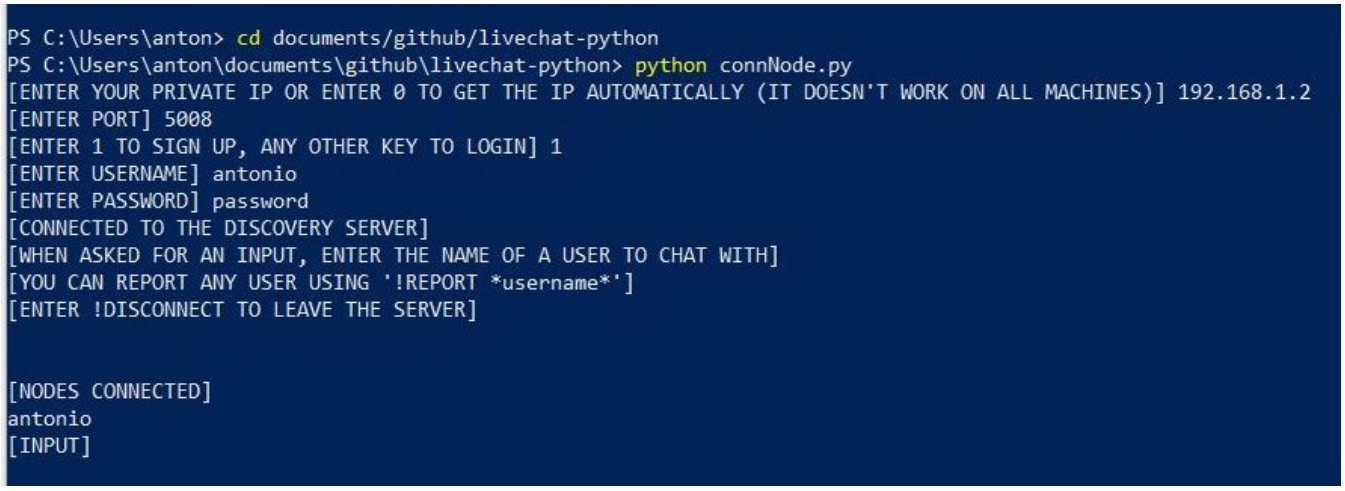
1) Installare Python;
2) Installare i seguenti moduli Python: bcrypt (per cifrare le password degli account degli utenti), psycopg2 (per collegarsi a PostgreSQL). Le altre librerie sono standard, ma nel caso in cui per qualche motivo non siano già installate, basta lanciare da terminale il comando pip install nomeModulo;
3) Installare il file .zip dell'applicazione dal repository di GitHub ed estrarre la cartella;
4) Installare PostgreSQL 14 e configurare connDB in base alle credenziali utilizzate;
5) Creare il database 'livechat_users';
6) Importare in PostgreSQL lo script SQL per la creazione della tabella degli utenti, presente nel repository e chiamato *client.sql*;
7) Andare nei file *connNode.py* e *discoveryServer.py* e modificare la costante DISCOV_SERVER in riga 8 inserendo, rispettivamente, l'IP pubblico del discovery server e l'IP privato del dispositivo che fungerà da discovery server.

I file sono scaricabili da [questo repository].

4.7. Screenshot

Avvio di discoveryServer.py:

Avvio di connNode.py e registrazione dell'utente Antonio:



Login dell'utente Michele:

```
PS C:\Users\anton\Documents\GitHub\Livechat-python> python connNode.py
[ENTER YOUR PRIVATE IP OR ENTER 0 TO GET THE IP AUTOMATICALLY (IT DOESN'T WORK ON ALL MACHINES)] 192.168.1.2
[ENTER PORT] 5008
[ENTER 1 TO SIGN UP, ANY OTHER KEY TO LOGIN] 2
[ENTER USERNAME] michele
[ENTER PASSWORD] 12345678
[CONNECTED TO THE DISCOVERY SERVER]
[WHEN ASKED FOR AN INPUT, ENTER THE NAME OF A USER TO CHAT WITH]
[YOU CAN REPORT ANY USER USING '!REPORT *username*']
[ENTER !DISCONNECT TO LEAVE THE SERVER]


[NODES CONNECTED]
antonio
michele
[INPUT]
```

Collegamento tra Michele (a sinistra) e Antonio (a destra):

```
[NODES CONNECTED]
antonio
michele
[INPUT] antonio
[ATTEMPTING TO CONNECT TO THE NODE]
[INPUT] [USE !DISCONNECT TO LEAVE THE CHAT]
[USE !FILE *path* TO SEND A FILE]
[CONNECTED TO THE USER]

.
Ciao Antonio
antonio: Ciao Michele
```

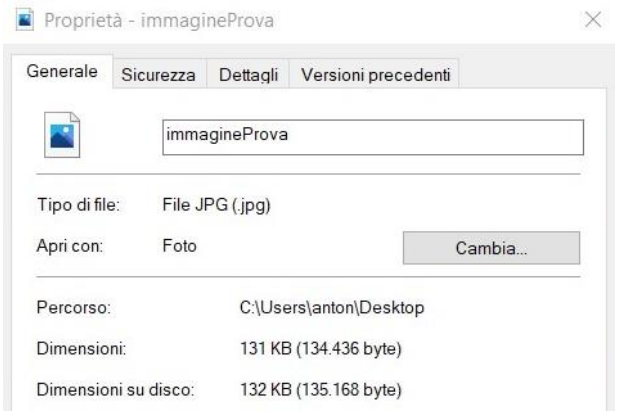
```
[NODES CONNECTED]
antonio
[INPUT]
[NODES CONNECTED]
antonio
michele
[michele STARTED A CHAT]
[ATTEMPTING TO CONNECT TO THE NODE]
[USE !DISCONNECT TO LEAVE THE CHAT]
[USE !FILE *path* TO SEND A FILE]
[CONNECTED TO THE USER]
michele: .
michele: Ciao Antonio
Ciao Michele
```

Nota: sono due terminali diversi.

Schermata visualizzata dal discovery server dopo la disconnessione dei due utenti:

```
PS C:\Users\anton\Documents\GitHub\Livechat-python> python discoveryServer.py
[DISCOVERY SERVER STARTED]
[NEW CONNECTION] ('[redacted]', 5007): antonio
[NEW CONNECTION] ('[redacted]', 5008): michele
[DISCONNECTION] ('[redacted]', 5008): michele
[DISCONNECTION] ('[redacted]', 5007): antonio
```

Contenuto che Michele vuole inviare ad Antonio:



Michele sta inviando il file ad Antonio:

```
!FILE C:/Users/anton/Desktop/immagineProva.jpg
[FILE SENT]
```

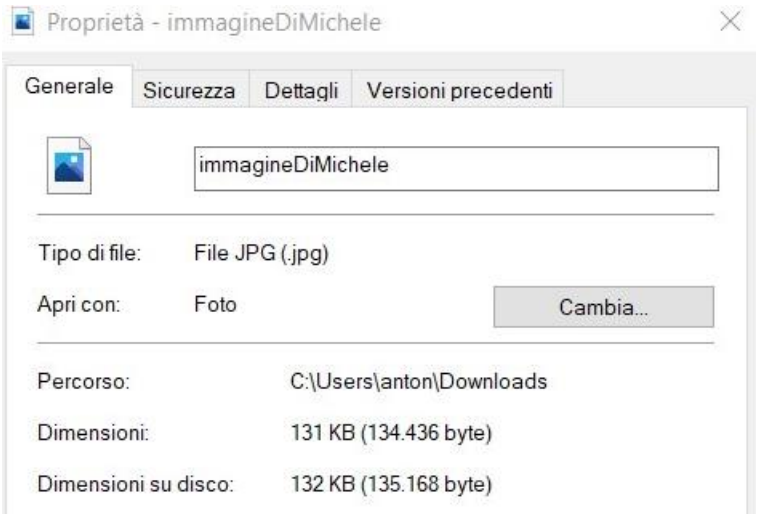
Antonio scarica il file:

```
[michele WANTS TO SEND YOU A FILE]
[PRESS ENTER TO CONTINUE]


[INSERT FILE NAME] immagineDiMichele.jpg
[PRESS ENTER TO CONTINUE]

[INSERT A VALID PATH] C:/Users/anton/Downloads
[SEGMENT 1 DOWNLOADED]
[SEGMENT 2 DOWNLOADED]
[SEGMENT 3 DOWNLOADED]
[SEGMENT 4 DOWNLOADED]
[SEGMENT 5 DOWNLOADED]
[SEGMENT 6 DOWNLOADED]
[SEGMENT 7 DOWNLOADED]
[SEGMENT 8 DOWNLOADED]
```

Download completato:

```
[SEGMENT 129 DOWNLOADED]
[SEGMENT 130 DOWNLOADED]
[SEGMENT 131 DOWNLOADED]
[SEGMENT 132 DOWNLOADED]
[DOWNLOAD COMPLETED]
Ho scaricato il file
!DISCONNECT
[DISCONNECTED]
```

Proprietà del file scaricato:



Situazione nel database:

	username [PK] character varying	code [PK] integer	password character varying	reports integer
1	antonio	0	$2b$12$lEl4O/TIwH.Re.YGrTypmOyfi9128i88dCABUrOEG/g1BuJvPI6.i	0
2	michele	1	$2b$12$QJyaLTaik3pZiN2bqSOyauUFZbHIpI9mw9Ize6mhn6T13xdvf4OQO	0

5. Fonti

Sono state utilizzate le seguenti fonti:

- What is Peer to Peer Network, and How does it work? - (blockchain-council.org)
- Difference between Client-Server and Peer-to-Peer Network - javatpoint
- Handbook of Peer-to-Peer Networking - Google Books
- What is P2P(Peer-to-Peer Process)? - GeeksforGeeks
- Peer to Peer Architecture (enjoyalgorithms.com)
- Peer-to-Peer Blockchain Networks: The Rise of P2P Crypto Exchanges | Bybit Learn
- Unstructured Peer-to-Peer Networks - SOICT
- proj_supernode_paper.pdf (kevinregan.com)
- The centralized peer-to-peer (P2P) system. A peer E sends a message to... | Download Scientific Diagram (researchgate.net)
- Peer to Peer (P2P) e Client Server: cosa sono e come funzionano? | Young Platform
- Briar, l'App di messaggistica suggerita da Julian Assange agli ucraini (techprincess.it)
- What is Peer-to-Peer Gaming, and How Does it Work? - VGKAMI
- Information about Peer-to-Peer games on PC | Ubisoft Help
- What is BitTorrent? | Is Torrenting Safe? (kaspersky.com)
- TNTVillage deve rimuovere tutti i contenuti ancora scaricabili. Tribunale di Milano conferma la sentenza | DDay.it
- TNT Village: guida alla comunità dei torrent italiani - ChimeraRevo
- Il caporale Chelsea Manning. La storia di Manning: dall'arresto per… | by Dario Centofanti | POPINGA
- Dark Web Browser: What Is Tor, Is It Safe & How to Use It | Avast
- Security Considerations for Peer-to-Peer Networks | by Randika Lakshan | Medium
- Protecting Your Network from Ransomware Attacks | Ransomware (rapid7.com)
- socket — Low-level networking interface — Python 3.11.1 documentation