Chat-Programm

Frankfurt University of Applied Sciences

Mulugeta Gebregergis
Nurettin Tasoluk
Noel-Leon Hildenbrand
Mohammad Inal

Modul: BSRN
Professor: Prof. Markus Miettinen
Gruppe: A
Abgabe: 22.06.2025

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

File Documentation

2.1 cli.py File Reference

Kommandozeileninterface (CLI) zur Chat-Steuerung im P2P-Netzwerk.

Functions

- cli.run_cli (handle, port, whoisport, pipe_conn)
  Startet das interaktive Kommandozeileninterface.

2.1.1 Detailed Description

Kommandozeileninterface (CLI) zur Chat-Steuerung im P2P-Netzwerk.

Dieses Modul erlaubt Benutzern, mit einem Peer-to-Peer-Netzwerk zu interagieren. Es unterstützt Befehle zur Anzeige aktiver Peers, zum Senden von Nachrichten und Bildern sowie zum Beitreten und Verlassen des Netzwerks. Die Kommunikation erfolgt über UDP-Broadcasts und direkte TCP-Verbindungen.

Author

Mulugeta Gebregergis

Date

Juni 2025

2.1.2 Function Documentation

2.1.2.1 run_cli()

```
cli.run_cli (
              handle,
              port,
              whoisport,
              pipe_conn)
```

Startet das interaktive Kommandozeileninterface.

Dieses Interface akzeptiert benutzerseitige Eingaben für folgende Befehle: PEERS, MSG, IMG, JOIN, LEAVE, WHO, BEENDEN. Der Discovery-Prozess wird über eine Pipe abgefragt.

Parameters

handle	Benutzername (Kennung) dieses Clients.
port	Portnummer dieses Clients.
whoisport	Port für Broadcast-Anfragen im lokalen Netzwerk.
pipe_conn	Verbindung zum Discovery-Prozess (Pipe für Peer-Updates).

2.2 client.py File Reference

Dieses Programm sendet Textnachrichten und Bilder über TCP/IP im lokalen Netzwerk.

Functions

- client.zensiere_nachricht (text)

  Ersetzt beleidigende Wörter mit Sternchen.
- client.send_msg (ip, port)

  Sendet eine Textnachricht an einen Peer über IPv4.
- client.send_image (ip, port)

  Sendet eine Bilddatei an einen Peer über IPv4.

Variables

- **client.config** = load_config("config/settings3.toml")

  Lädt Konfigurationswerte aus der TOML-Datei.
- **client.HANDLE** = config["handle"]
- **client.PORT** = config["port"]
- **client.WHOISPORT** = config["whoisport"]
- **client.AUTOREPLY** = config["autoreply"]
- **client.IMAGEPATH** = config["imagepath"]
- int **client.BUFFER_SIZE** = 4096
- list client.BLACKLIST

  Liste mit Wörtern, die in Nachrichten zensiert werden sollen.

2.2.1 Detailed Description

Dieses Programm sendet Textnachrichten und Bilder über TCP/IP im lokalen Netzwerk.

Das war ein weiterer Teil (Noel) des Peer-to-Peer Chat-Projekts. Ich habe diesen Client gebaut, damit man mit anderen Peers über IP kommunizieren kann – entweder mit Text oder mit einem Bild.

Ursprünglich hatte ich Unterstützung für IPv6 drin, aber das hat bei meinem Kommilitonen nicht zuverlässig funktioniert (wir hatten oft `No route to host` oder `Connection refused`) obwohl wir die richtigen Adressen verwendet haben. Deshalb habe ich den Code zurück auf IPv4 reduziert – das ist einfacher, stabiler und wir hatten damit weniger Probleme.

Außerdem war es schwer, bei IPv6 richtige Adressen zu bekommen, die auf beiden Rechnern funktionieren. Wir haben viel Zeit mit `ip addr show` und `ipconfig` verbracht, aber ohne Erfolg.
Zwischenzeitlich konnte ich zwar Textnachrichten senden, aber mein Kommilitone konnte nicht antworten.
IPv4 war letztendlich einfach klarer und zuverlässiger.

2.2.2 Function Documentation

2.2.2.1 send_image()

```
client.send_image (
            ip,
            port)
```

Sendet eine Bilddatei an einen Peer über IPv4.

Parameters

ip	Die Ziel-IP-Adresse des Empfängers.
port	Der Port, an dem der Empfänger lauscht.

Note

Der Pfad zum Bild wird über die Konsole eingegeben. Das Bild wird vollständig als Binärdaten gesendet.

2.2.2.2 send_msg()

```
client.send_msg (
            ip,
            port)
```

Sendet eine Textnachricht an einen Peer über IPv4.

Parameters

ip	Die Ziel-IP-Adresse des Empfängers.
port	Der Port, an dem der Empfänger zuhört.

Note

Die Nachricht wird über die Konsole eingegeben. Bevor gesendet wird, zensiere ich sie automatisch.

2.2.2.3 zensiere_nachricht()

```
client.zensiere_nachricht (
            text)
```

Ersetzt beleidigende Wörter mit Sternchen.

Parameters

text	Der Original-Text, den der Nutzer geschrieben hat.

Returns

Eine zensierte Version des Texts.

2.2.3 Variable Documentation

2.2.3.1 BLACKLIST

```
list client.BLACKLIST
```

Initial value:

```
00001 =  [
00002     "arschloch", "penner", "fuck", "idiot", "dummkopf", "arsch",
00003     "spast", "miststück", "arschh", "fuckk", "scheiße", "fick dich"
00004 ]
```

Liste mit Wörtern, die in Nachrichten zensiert werden sollen.

Die Idee kam uns in einer ihrer Übungen und hat uns ein wenig an Exceptions aus Java erinnert. Wir wollten eine einfache Möglichkeit, beleidigende Wörter in Nachrichten zu ersetzen. Die Liste ist beliebig erweiterbar, falls wir später noch mehr Wörter zensieren wollen.

2.3 discovery.py File Reference

Erweiterter Discovery-Prozess für das P2P-Chatprogramm.

Functions

- discovery.discovery_process (handle, port, whoisport, pipe_conn)
  Führt den Discovery-Prozess im Netzwerk durch.

2.3.1 Detailed Description

Erweiterter Discovery-Prozess für das P2P-Chatprogramm.

Dieses Modul ermöglicht die automatische Erkennung, Verwaltung und Kommunikation zwischen Peers im Peer-to-Peer-Netzwerk. Es unterstützt sowohl JOIN- und LEAVE-Nachrichten als auch WHO- und KNOWUSERS-↩ Nachrichten zur erweiterten Peer-Erkennung.

Author

Mulugeta Gebregergis, Mohammad Inal

Date

Juni 2025

2.3.2 Function Documentation

2.3.2.1 discovery_process()

```
discovery.discovery_process (
            handle,
            port,
            whoisport,
            pipe_conn)
```

Führt den Discovery-Prozess im Netzwerk durch.

Die Funktion erkennt neue Peers im lokalen Netzwerk, verarbeitet Beitritts- (JOIN), Austritts- (LEAVE), WHO- und KNOWUSERS-Nachrichten und aktualisiert die Peer-Liste, die regelmäßig an die CLI gesendet wird.

Parameters

handle	Der Benutzername dieses Peers.
port	Der Port dieses Peers für TCP-Nachrichten.
whoisport	Der UDP-Port für Broadcast-Erkennung.
pipe_conn	Die Pipe zur Kommunikation mit dem CLI-Prozess.

2.4 loader.py File Reference

Lädt Konfigurationsdaten aus einer TOML-Datei.

Functions

- loader.load_config (dateipfad="config/settings.toml")
  Lädt die Konfigurationsdaten aus einer .toml-Datei.

2.4.1 Detailed Description

Lädt Konfigurationsdaten aus einer TOML-Datei.

Dieses Modul enthält eine Funktion zum Einlesen einer Konfigurationsdatei im TOML-Format. Dadurch kann auf mehrfachen, redundanten Code beim Einlesen verzichtet werden.

Author

Noel-Leon Hildenbrand

Date

Juni 2025

2.4.2 Function Documentation

2.4.2.1 load_config()

```
loader.load_config (
              dateipfad = "config/settings.toml")
```

Lädt die Konfigurationsdaten aus einer .toml-Datei.

Diese Funktion kapselt den Ladevorgang in eine zentrale Stelle, damit nicht in mehreren Modulen derselbe Einlesecode geschrieben werden muss.

Parameters

dateipfad	Der Pfad zur TOML-Datei. Standard ist „config/settings.toml“.

Returns

Gibt den Inhalt der Konfigurationsdatei als Dictionary zurück.

2.5 main.py File Reference

Einstiegspunkt für das P2P-Chatprogramm.

Functions

- main.main ()
  Hauptfunktion des Programms.

2.5.1 Detailed Description

Einstiegspunkt für das P2P-Chatprogramm.

Dieses Skript verbindet alle Hauptkomponenten: Konfigurations-Loader, Server, Discovery und CLI. Es sorgt für den Start aller Prozesse und koordiniert deren Ablauf.

Author

Mulugeta Gebregergis

Date

Juni 2025

2.5.2 Function Documentation

2.5.2.1 main()

```
main.main ()
```

Hauptfunktion des Programms.

Diese Funktion lädt die Konfiguration, startet den Server, Discovery-Prozess und CLI. Sie ist die zentrale Koordinationsstelle für den Ablauf der Anwendung.

2.6 server.py File Reference

TCP-Server für den Chat – verarbeitet Nachrichten und Bilder.

Functions

- server.start_server (port, autoreply, imagepath, handle)
  Startet den TCP-Server im Hintergrund.

2.6.1 Detailed Description

TCP-Server für den Chat – verarbeitet Nachrichten und Bilder.

Dieses Skript startet einen einfachen TCP-Server, der über Threads Nachrichten und Bilddaten verarbeitet. Es wird auf eine direkte Antwort mit Autoreply geachtet und empfangene Bilder werden im angegebenen Pfad gespeichert.

Author

Nurettin Tasoluk

Date

Juni 2025

2.6.2 Function Documentation

2.6.2.1 start_server()

```
server.start_server (
              port,
              autoreply,
              imagepath,
              handle)
```

Startet den TCP-Server im Hintergrund.

Der Server lauscht auf eingehende Verbindungen und verarbeitet sie mit einem eigenen Thread. Nachrichten werden angezeigt, Bilder werden gespeichert, Autoreply kann optional aktiviert werden.

Parameters

port	Der Port, auf dem der Server lauscht.
autoreply	Automatische Antwort, die gesendet wird (wenn aktiviert).
imagepath	Pfad, wo empfangene Bilder gespeichert werden sollen.
handle	Der eigene Benutzername – wird bei Autoreply verwendet.

Index