High-Level-Dokumentation

Projekt: Dezentrales Chatprogramm (BSRN)
Sommersemester 2025

Diese Dokumentation beschreibt die Architektur, Komponenten, Konfiguration, Technologien sowie zentrale technische Entscheidungen des Chatprojekts.

Gruppe A7
Douaa Boukbib
Meriem Mansour
Mahrukh Shabbir
Ayoub Bouda
Christian Duku

1. Ziel des Projekts

Das Ziel des Projekts war es, ein dezentrales Chatprogramm in Python zu entwickeln, das Text- und Bildnachrichten zwischen mehreren Teilnehmern in einem lokalen Netzwerk ohne zentrale Serverinfrastruktur ermöglicht. Dabei kommt ein eigenes textbasiertes Protokoll namens SLCP (Simple Local Chat Protocol) zum Einsatz.

2. Systemarchitektur

Das Programm basiert auf einer modularen Architektur mit drei Prozessen:

- **Netzwerkprozess** (network.py) für UDP-/TCP-Kommunikation
- **Discoveryprozess** (discovery.py) für Teilnehmererkennung
- **Benutzeroberfläche** (ui_cli.py) zur Interaktion per Kommandozeile

Alle drei Prozesse sind eigenständig lauffähig und können einzeln gestartet werden. Der Discoveryprozess kann z. B. über den Befehl start_discovery separat ausgeführt werden.

Die Prozesse sind über **Interprozesskommunikation (IPC)** mittels Queues verbunden. Der Einstiegspunkt ist main.py, über das alle Prozesse gemeinsam gestartet werden.

3. Hauptkomponenten

main.py	Startet Konfiguration und Prozesse
ui_cli.py	CLI-Eingabeschleife für Benutzerbefehle
network.py	Verarbeitet SLCP-Kommandos über UDP/TCP
discovery.py	Erkennt Peers via WHOIS/IAM
ipc.py	Definiert Queues für IPC
config.py	Hilfsfunktionen für Konfigurationsdateien
config.toml	Persistente Konfigurationsdaten (Port, Handle etc.)
slcp.py	Parsen und Erzeugen von SLCP-Nachrichten
Network_utils.py	Hilfsfunktionen für Netzwerk (z. B.

	Broadcast-Erkennung)
image_tools.py	Bildspeicherung und Anzeige von Bildern

4. Konfigurationsdateien

Die Konfiguration besteht aus zwei Elementen:

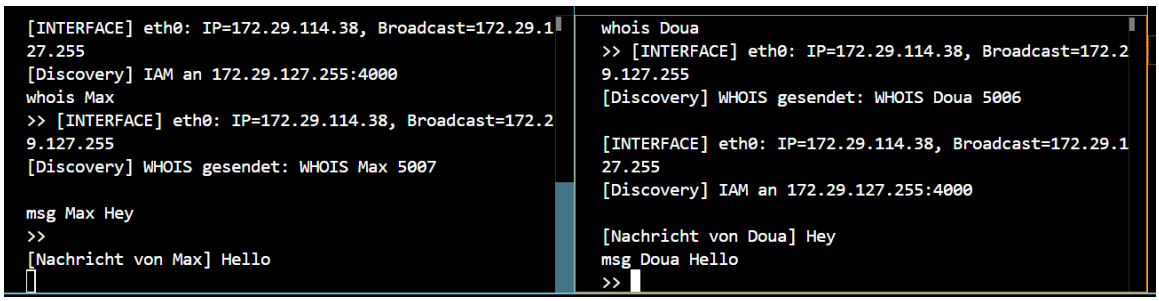
config.toml: Enthält Konfigurationsdaten der Clients, z. B. Handle, Port, Autoreply

config.py: Liest/schreibt Werte aus/in die .toml-Datei, z. B.:

```
[defaults]
port_range = [5000, 5100]
autoreply = "Ich bin gerade nicht erreichbar."
imagepath = "./bilder"
```

5. Bedienung

- Das Programm wird über die Kommandozeile gestartet:
  python3 main.py
  Beispiele für Eingaben:
  join – Netzwerkteilnahme starten
- msg <Name> <Text> – Nachricht senden
- img <Name> <Pfad> – Bild senden
- whois <Name> – IP/Port von Teilnehmer abfragen
- autoreply <Text> – Autoreply setzen
- config – Konfiguration anzeigen
- start_discovery – Discoveryprozess manuell starten
- exit – Programm beenden



6. Kommunikation

- **Textnachrichten**: Über UDP direkt zwischen Clients
- **Bildnachrichten**: Header über UDP, Bilddaten über TCP
- **Peer Discovery**: WHOIS/IAM-Mechanismus über UDP-Broadcasts
- **Autoreply**: Automatische Nachrichten bei Abwesenheit

6. Herausforderungen & Lösungen

Herausforderungen	**Lösungen**
Gleichzeitiger Empfang & Versand	Mehrere Threads pro Prozess
Nur ein Discoveryprozess pro Host	Lock-Datei zur Synchronisierung
Bildversand mit Kommentar	Trennung von Header (UDP) und Daten (TCP)
Automatisierte Antworten	Autoreply-Logik im Discoveryprozess
Prozesse müssen einzeln startbar sein	Umsetzung über Parameter und Queue-Architektur

7. Verwendete Technologien

Programmiersprache:
• Python 3.10+

Bibliotheken:
• socket, threading, multiprocessing, os, toml,

Entwicklungsumgebung:

- Visual Studio Code
- Git + GitHub für Versionsverwaltung

Weitere Tools:

- Doxygen für Code-Kommentare
- PlantUML – zur Darstellung der Systemarchitektur

8. Erweiterbarkeit

Dank der modularen Struktur lässt sich das System problemlos erweitern. Mögliche Erweiterungen sind:

- Dateiübertragung
- Verschlüsselung
- Emojis im CLI
- Web-Oberfläche
- Multicast-basierte Kommunikation