FRANKFURT UNIVERSITY OF APPLIED SCIENCES

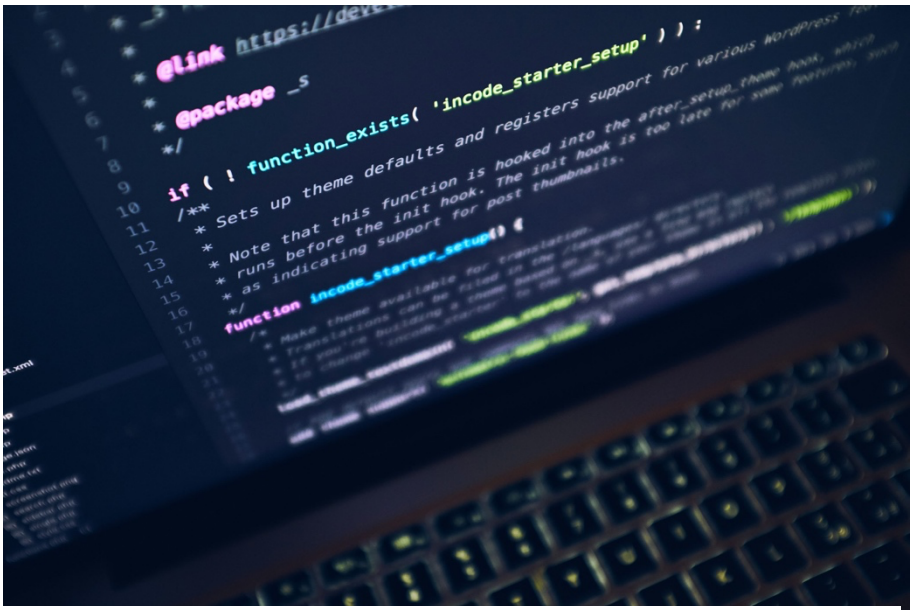
SLCP-Chatprogramm

Entwicklung eines dezentralen Peer-to-Peer-Chats im lokalen Netzwerk

Modul: Betriebssysteme und Rechnernetze – Frankfurt UAS, SoSe 2025

Prof. Dr.-Ing. Peter Ebinger

Erstell von:
Arda Toksun
Emil Suhr
David Marten
Robin Speichermann



Quelle: https://d11eteosk75kgr.cloudfront.net/styles/np8_full/s3/media/2024/04/03/api_attacks_emea.jpg?itok=V_4nUH-9

1.1 Einleitung

1.1 Ziel des Projekts

Ziel des Projekts ist die Entwicklung eines dezentralen Peer-to-Peer-Chatprogramms auf Basis des textbasierten SimpleLocal Chat Protocol (SLCP). Die Anwendung ermöglicht den Versand und Empfang von Text- und Bildnachrichten über TCP sowie die automatische Peer-Erkennung im lokalen Netzwerk mittels UDP. Dabei kommen moderne Konzepte wie Interprozesskommunikation, Threading, Konfigurationsverwaltung und plattformunabhängige Netzwerktechniken zur Anwendung.

1.2 Einordnung im Modul BSRN

Das Projekt wurde im Rahmen des Moduls "Betriebssysteme und Rechnernetze" (BSRN) an der Frankfurt University of Applied Sciences im Sommersemester 2025 unter Leitung von Prof. Dr.-Ing. Peter Ebinger umgesetzt. Es diente dazu, theoretische Kenntnisse über Netzwerkkommunikation, verteilte Systeme und Prozessverwaltung praktisch anzuwenden. Ziel war die Entwicklung einer stabilen, modular aufgebauten Chatanwendung mit SLCP-konformer Nachrichtenübertragung und Peer-Discovery im lokalen Netz.

2. Projektplanung + Code-Bezug

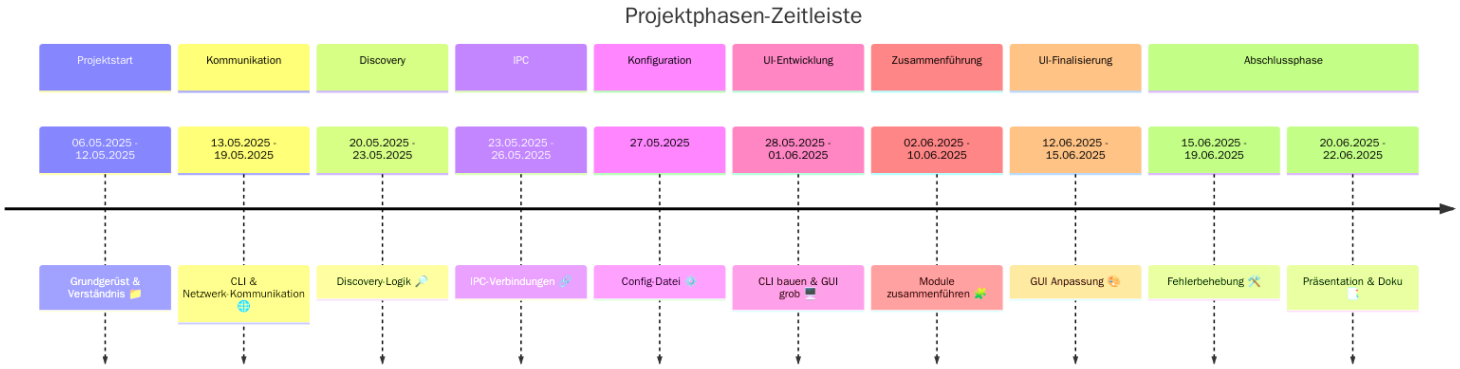
2.1 Projektorganisation und Verantwortlichkeiten

Das Projekt wurde im Team von vier Studierenden in arbeitsteiliger Entwicklung umgesetzt. Die Aufgabenverteilung basierte auf dem offiziellen Projektplan aus Meilenstein 2. Jeder Teilbereich wurde durch mindestens zwei Teammitglieder abgedeckt, wobei sich Verantwortlichkeiten wie folgt ergaben:

Arbeitspaket	**Inhalt**	**Hauptverantwortlich**
Architekturentwurf	Planung der Struktur und Erstellung des Architekturdiagramms	Alle
Netzwerk- & SLCP-Implementierung	Umsetzung des Protokolls (JOIN, MSG, IMG, QUIT, WHO) über TCP/UDP	Arda, Emil, Robin
Discovery-Dienst & IPC	Discovery-Service für Peer-Erkennung + zentrale Interprozesskommunikation	Emil, Robin
Benutzeroberfläche & Konfiguration	CLI-Befehle, Konfigurationsänderung, Konfigurationsdatei (TOML)	Arda, Emil
GUI	Entwicklung der grafischen Oberfläche mit tkinter (optional)	Robin, David
Dokumentation & Koordination	Erstellung der technischen Dokumentation, Teamkoordination und Versionsverwaltung (Git)	Alle
Tests & Fehlerbehandlung	Funktionstests, Debugging und Stabilitätsprüfung	Alle

2.2 Zeitplan

Zeitraum	**Aufgabenbereiche**
06.05 – 12.05	Grundstruktur festlegen, Setup initialisieren
13.05 – 19.05	Kommunikation über CLI und Netzwerk entwickeln
20.05 – 23.05	Discovery-Logik realisieren
23.05.-26.05.	IPC-Logik realisiert
27.05.-27.05.	Config Datei implementiert + Layout angepasst
28.05.-01.06	CLI + GUI Anpassungen + features hinufügen
02.06 – 10.06.	Module zusammenführen, Dokumentation starten
12.06.-15.06.	GUI Anpassung
15.06.-19.06.	Feinschliff, Tests und Bugs fixen
20.06.-22.06.	Präsentation und finale Dokumentation



2.3 Funktionalität des Programms

Das Programm umfasst folgende Kernfunktionen:

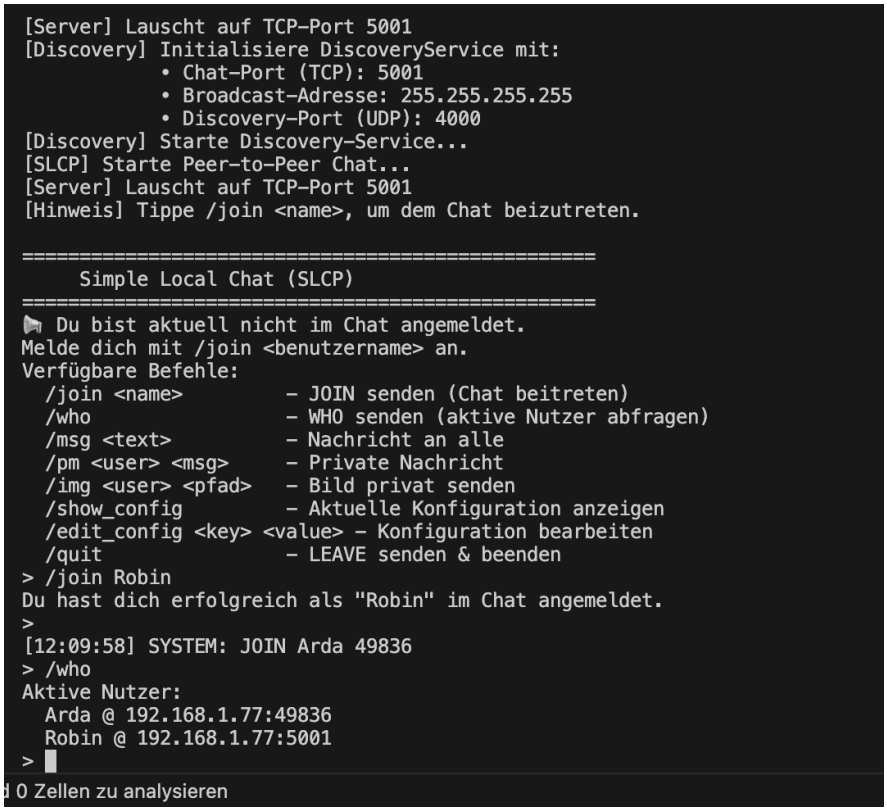
- **Versand und Empfang von Textnachrichten** über TCP (SLCP-konform)
- **Peer-to-Peer-Übertragung von Bildern** (PNG, JPG, GIF)
- **Automatische Peer-Erkennung** via UDP (JOIN/WHO/KNOWUSERS)
- **Autoreply-Modus nach Inaktivität** (optional manuell aktivierbar)
- **Laufzeitbearbeitung der Konfiguration** (insbesondere Benutzername & Autoreply Nachricht)
- **Anzeige aktiver Nutzer** über CLI und GUI
- **Persistente Speicherung empfangener Bilder**
- TCP-Basierte **Übertragung von KNOWUSERS-Informationen nach JOIN oder WHO**
- **Unterstützung von Direktnachrichten** durch gezieltes Routing an einzelne Peers

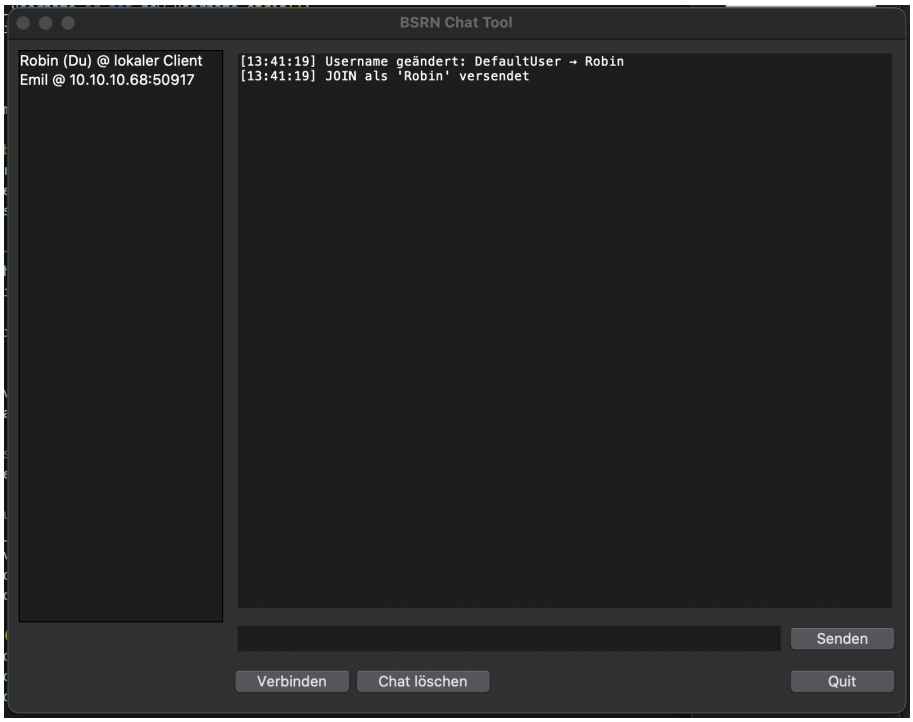
2.4 Bedienung über CLI & GUI

Das Chatprogramm kann einfach über die Kommandozeile oder alternativ über eine grafische Oberfläche (GUI) bedient werden. Alle Funktionen sind durch intuitive Befehle wie `/help` direkt zugänglich.

Befehl	**Beschreibung**
/help	Übersicht über verfügbare Kommandos
/join <name>	Chat beitreten und JOIN senden

/who	Aktive Nutzer anzeigen (sendet JOIN und WHO)
/msg <text>	Nachricht an alle Teilnehmer (Broadcast an alle Peers)
/pm <nutzer> <nachricht>	Private Nachricht an bestimmten Nutzer (unicast)
/img <nutzer> <pfad>	Bilddatei an Nutzer senden
/autoreply	Autoreply-Modus ein-/ausschalten (manuell)
/show_config	Aktuelle Konfiguration anzeigen
/edit_config handle <neuer_name>	Benutzername ändern zur Laufzeit
/edit_config autoreply <neue autoreply Nachricht>	Autoreply Nachricht live zur Laufzeit ändern
/quit	Chat verlassen (LEAVE) und Anwendung beenden





3. Architekturübersicht

3.1 Technischer Gesamtansatz

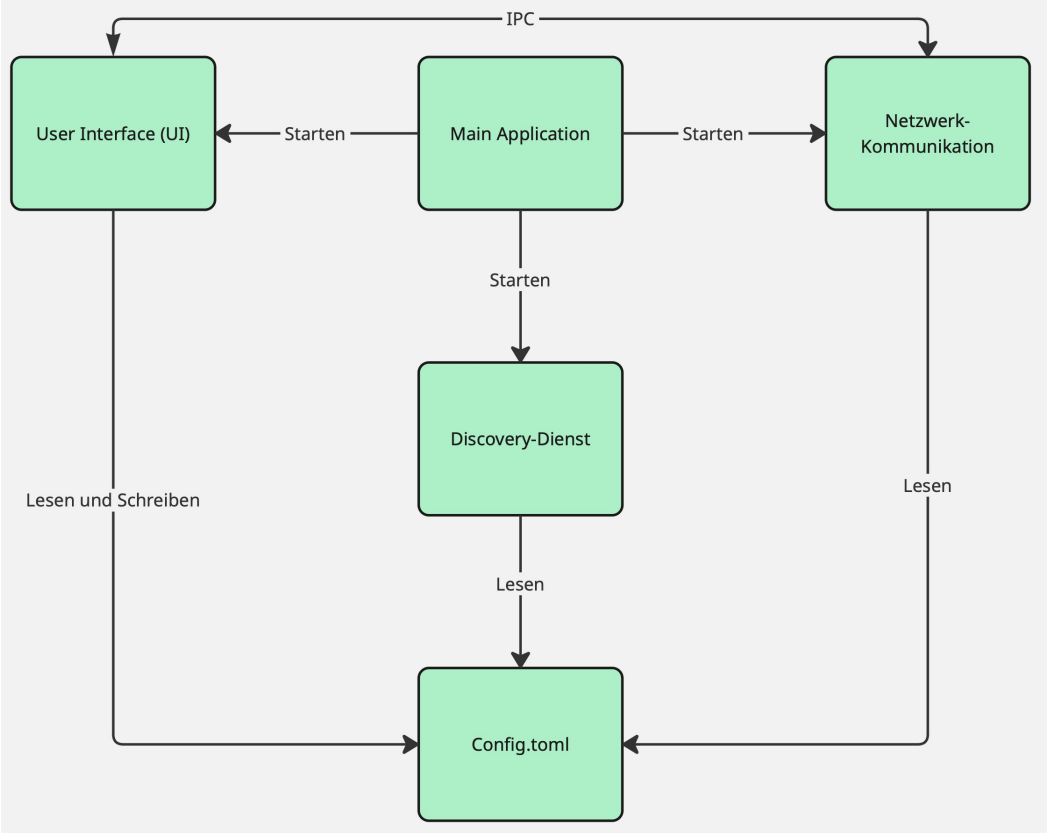
Im ersten Semester haben wir überwiegend mit Java gearbeitet. Für dieses Projekt wollten wir bewusst eine andere Sprache ausprobieren, da Python nicht nur für eine schnellere Entwicklung sorgt, sondern auch in der freien Wirtschaft sehr häufig eingesetzt wird.

Das Programm folgt einer dreigeteilten Architektur mit klarer Trennung der Komponenten:

- **Benutzerschnittstelle (CLI)**
- **Netzwerk-Kommunikation (Client/Server)**
- **Discovery-Dienst (UDP)**

Die Prozesse kommunizieren über IPC (Queues). Der Discovery-Dienst sendet JOIN/LEAVE-Nachrichten über UDP-Broadcast (Port 4000), während Chatnachrichten (Text, Bild) über TCP direkt übertragen werden.

3.2 Systemstruktur und Kommunikation



3.3 Modulübersicht

- `main.py`: Startet alle Prozesse und IPC-Verbindungen
- `cli.py`: Benutzeroberfläche über Konsole
- `chat_client.py`: Senden von Nachrichten/Bildern über TCP
- `chat_server.py`: Empfang eingehender TCP-Nachrichten
- `discovery.py`: UDP-Listener und Broadcast-Logik
- `ipc_handler.py`: Zentrale Nachrichtenverarbeitung zwischen Prozessen
- `config.toml`: Konfigurationsdatei mit Port, Handle und Autoreply