HIGH LEVEL DOKUMENTATION

PEER TO PEER CHAT PROGRAMM BY

Mulugeta Gebregergis

Nurettin Tasoluk

Noel-Leon Hildenbrand

Mohammad Inal

Professor: Prof. Markus Miettinen

Gruppe: A

Abgabe: 22.06.2025

Inhaltsverzeichnis

Inhalt

HIGH LEVEL DOKUMENTATION 1
Inhaltsverzeichnis 2
1. Einleitung 3
2. Architekturübersicht 3
3. Kommunikationsablauf und Protokoll 3
4. Konfigurationsdatei und CLI 4
5. Problemstellen und Herausforderungen 4
6. Ursprünglicher IPv6-Ansatz – ein gescheiterter Versuch 4
7. Probleme bei der Peer-Discovery – Peers sichtbar machen 5
8. Technische Herausforderungen im Netzwerk 6
9. Bedienung des Programms 7
10. Fazit 7
11. Erkenntnisse und Lernerfolge 8
12. Ausblick 9

1. Einleitung

Das folgende Dokument beschreibt den Aufbau und die Funktionsweise eines selbst entwickelten Peer-to-Peer-Chatprogramms. Ziel war es, Text- und Bildnachrichten im lokalen Netzwerk ohne zentralen Server zu ermöglichen. Die Umsetzung erfolgte im Modul „Betriebssysteme und Rechnernetze (BSRN)“ an der Frankfurt University of Applied Sciences. Grundlage ist ein eigenes Protokoll zur Peer-Kommunikation.

2. Architekturübersicht

Das System ist modular aufgebaut:

- main.py: Einstiegspunkt, startet Server, CLI und Discovery.
- cli.py: Benutzeroberfläche im Terminal.
- client.py: Senden von Nachrichten und Bildern.
- server.py: TCP-Server für eingehende Nachrichten.
- discovery.py: Peer-Erkennung über JOIN/WHO.
- loader.py: Liest Konfiguration aus TOML-Dateien.

Alle Module kommunizieren über Sockets und Pipes miteinander und ermöglichen die dezentrale Kommunikation zwischen Peers.

3. Kommunikationsablauf und Protokoll

Verwendet wird das SLCP (Simple Local Chat Protocol):

- JOIN: Anmeldung im Netzwerk via Broadcast
- WHO: Aktive Peers abfragen
- KNOWUSERS: Antwort mit Peer-Infos
- MSG: Textnachricht über TCP
- IMG: Bildübertragung in zwei Schritten

JOIN, WHO und LEAVE nutzen UDP-Broadcast, während Nachrichten (MSG/IMG) über TCP direkt an andere Peers gesendet werden.

4. Konfigurationsdatei und CLI

Die Datei settingsX.toml enthält alle wichtigen Einstellungen wie Handle, Port, Autoreply und Speicherpfad für empfangene Bilder. Die CLI erlaubt Steuerung über Befehle: PEERS, MSG, IMG, JOIN, WHO, LEAVE, BEENDEN. Dabei sind JOIN und WHO essenziell, um Peers im Netzwerk zu entdecken.

5. Problemstellen und Herausforderungen

Während der Entwicklung des Peer-to-Peer-Chatprogramms sind wir auf zahlreiche technische Herausforderungen gestoßen, die uns wichtige Einblicke in die Komplexität dezentraler Netzwerksysteme gegeben haben. Im Folgenden gehen wir auf die zentralen Problemstellen detailliert ein und reflektieren über unsere Lösungsansätze, getroffene Entscheidungen und bestehende Einschränkungen.

6. Ursprünglicher IPv6-Ansatz – ein gescheiterter Versuch

Zu Beginn des Projekts war die Idee, das Chatprogramm zukunftssicher zu gestalten, indem wir die Kommunikation über das IPv6-Protokoll umsetzen. Wir wollten sicherstellen, dass unsere Anwendung auch in modernen Netzwerkumgebungen mit IPv6-Adressen problemlos funktioniert. Die Umsetzung erwies sich jedoch als äußerst fehleranfällig:

- **Verbindungsprobleme:** Trotz korrekter Adressangabe erhielten wir häufig Fehlermeldungen wie No route to host oder Connection refused, insbesondere beim Verbindungsaufbau zwischen zwei Peers.
- **Adressfindung:** Es war äußerst schwierig, passende und gültige IPv6-Adressen zu identifizieren, die gleichzeitig auf beiden Rechnern erreichbar waren. Befehle wie ip addr show (Linux/macOS) oder ipconfig (Windows) halfen nicht zuverlässig weiter.

- **Einseitige Kommunikation:** In manchen Fällen konnten wir zwar Nachrichten erfolgreich senden, jedoch reagierte der Empfänger nicht – ein Zeichen für asymmetrische Konnektivität im IPv6-Netz.
- **Fazit:** Aufgrund dieser instabilen Ergebnisse entschieden wir uns bewusst, auf IPv4 zurückzugreifen. IPv4 war nicht nur einfacher in der Anwendung, sondern auch konsistenter in der Kommunikation zwischen unseren Testgeräten. Der Wechsel stellte sich als entscheidender Schritt zur Stabilisierung unserer Architektur heraus.

7. Probleme bei der Peer-Discovery – Peers sichtbar machen

Ein zentrales Element im P2P-Chatprogramm ist die Fähigkeit, aktive Peers im Netzwerk zuverlässig zu erkennen und sichtbar zu machen. Diese Funktion basiert auf dem Discovery-Modul, welches JOIN- und WHO-Nachrichten verarbeitet. Hierbei traten jedoch folgende Probleme auf:

- **Inkonsistenz bei der Peer-Erkennung:** Trotz gesendeter JOIN- und WHO-Nachrichten wurden aktive Peers nicht immer korrekt erkannt und angezeigt. Dies erschwerte das gezielte Versenden von Nachrichten erheblich, da der Nutzer zunächst nicht wusste, mit wem er kommunizieren kann.
- **Teilweise funktionierende Lösung:** In einem konkreten Testlauf gelang es uns, mithilfe des WHO-Kommandos eine gültige Peer-Liste im Terminal anzuzeigen. Dieser erfolgreiche Fall wurde als Screenshot dokumentiert und ist Bestandteil unserer Projektpräsentation (siehe PowerPoint-Folie).
- **Persistenz des Problems:** Trotz zahlreicher Versuche mit Sockets, Broadcast-Einstellungen, Wartezeiten und Alternativprotokollen (z. B. manuelles erneutes JOIN) konnte dieses Verhalten nicht vollständig behoben werden. Das Discovery-System bleibt somit in bestimmten Netzwerkkonstellationen unzuverlässig – ein Punkt, der weiterführend analysiert und verbessert werden müsste.

8. Technische Herausforderungen im Netzwerk

Neben den oben genannten spezifischen Problemen gab es weitere technische Hürden, die das gesamte Projekt beeinflusst haben:

- **UDP-Broadcast und Firewalls:** Das Versenden von WHO- und JOIN-Nachrichten über UDP-Broadcast funktioniert nur dann zuverlässig, wenn die Netzwerkkonfiguration dies zulässt. Firewalls oder isolierte Netzwerke (z. B. Campus-WLAN mit VLANs) können diese Pakete blockieren, was dazu führt, dass keine Discovery stattfinden kann.
- **JOIN ist notwendig, aber nicht ausreichend:** Nach dem Start des Programms wird automatisch eine JOIN-Nachricht versendet. Um jedoch die vollständige Peer-Liste zu erhalten, ist ein anschließender WHO-Befehl zwingend notwendig. Ohne diese manuelle Abfrage bleiben viele Teilnehmer für den Nutzer unsichtbar.
- **Fehlertoleranz bei der Bildübertragung:** Die Übertragung von Bildern setzt voraus, dass die angegebene Dateigröße exakt mit der tatsächlichen Byteanzahl übereinstimmt. Ein zu großer oder zu kleiner Wert führt zu einem inkorrekten Bildempfang, was nicht nur zu Benutzerfrust, sondern auch zu potenziellen Sicherheitsrisiken (Pufferüberlauf etc.) führen kann.

Diese Herausforderungen verdeutlichen, dass die Entwicklung verteilter Systeme weit mehr als nur das Schreiben von Code erfordert. Es geht darum, sich mit Netzwerktopologien, Firewalls, Adressierung und Zustandsverhalten auseinanderzusetzen. Die beschriebenen Probleme stellen daher keine Misserfolge dar, sondern repräsentieren wichtige Meilensteine im Lernprozess unseres Teams. Sie zeigen auf, welche Aspekte wir im weiteren Verlauf des Projekts priorisieren müssen – insbesondere im Hinblick auf Stabilität und Nutzbarkeit in realen Netzwerkumgebungen.

9. Bedienung des Programms

1. Öffne auf beiden Rechnern jeweils zwei Terminals nebeneinander.
2. Wechsle ins Projektverzeichnis mit: cd ~/Desktop/p2p_chat
3. Starte das Programm mit: python3 main.py
   Voraussetzung: main.py muss im aktuellen Ordner sein.
4. Das Terminal zeigt: [TCP-Server] Lauscht auf Port XXXX (IPv4)
   Bedeutet: Server ist bereit. Entsprechende Befehle werden angezeigt.
5. JOIN wird automatisch gesendet – der Client meldet sich im Netzwerk an.
6. **WICHTIG**: Danach WHO eingeben. Der WHO-Befehl sendet eine UDP-Broadcast-Nachricht an alle aktiven Discovery-Server. Diese antworten mit KNOWUSERS, das Handle, IP und Port aller bekannter Peers enthält.
   → Ohne WHO können Teilnehmer übersehen werden, wenn deren JOIN verpasst wurde.
7. Anschließend PEERS: zeigt alle gefundenen Peers wie z. B.:
   - Mulu 172.20.10.6:5012
   - Milad 172.20.10.9:5013
8. MSG: IP und Port eingeben, danach Nachricht. Empfang erfolgt direkt.
9. IMG: Empfängername, Bildgröße in Bytes und Pfad zum Bild eingeben.
   Das Bild wird im Zielsystem gespeichert. Text und Bild werden getrennt verarbeitet.

10 Fazit

Im Rahmen des Moduls Betriebssysteme und Rechnernetze (BSRN) entstand ein funktionsfähiges, modular strukturiertes Peer-to-Peer-Chatprogramm, das eine dezentrale Kommunikation im lokalen Netzwerk ermöglicht. Das System erfüllt grundlegende Anforderungen an Stabilität, Modularität und Verständlichkeit und bildet damit eine solide technische Grundlage für weiterführende Entwicklungen im Bereich verteilter Systeme.

Durch den Einsatz eines eigenen Protokolls (SLCP) und die gezielte Trennung der Verantwortlichkeiten in verschiedene Module (CLI, Server, Client, Discovery,

Konfiguration) wurde eine Architektur geschaffen, die sowohl nachvollziehbar als auch flexibel erweiterbar ist. Die Entscheidung, auf IPv4, statt IPv6 zu setzen, hat sich aus Gründen der Netzwerkinfrastruktur als technisch sinnvoll und stabil erwiesen.

Besonders hervorzuheben ist der dezentrale Ansatz: Das Programm benötigt keinen zentralen Server und kann dennoch aktiv andere Teilnehmer erkennen, mit ihnen kommunizieren und Informationen wie Text- oder Bildnachrichten austauschen. Diese dezentrale Kommunikation erfolgt vollständig lokal, ressourcenschonend und unabhängig von externer Infrastruktur – ein Aspekt, der besonders in sicherheitskritischen oder netzwerkisolierten Umgebungen Vorteile bietet.

11 Erkenntnisse und Lernerfolge

Im Verlauf des Projekts konnten wir tiefe Einblicke in netzwerknahe Programmierung, Socket-Kommunikation, Inter-Process Communication (IPC), Multithreading und Protokolldesign gewinnen. Auch der Umgang mit realitätsnahen Problemen wie Firewalls, IP-Adressierung oder Broadcast-Verhalten hat unsere Fähigkeit zur praxisorientierten Fehleranalyse und Problemlösung geschärft.

Neben den technischen Kenntnissen haben wir auch wichtige soziale und organisatorische Erfahrungen gesammelt: Trotz zeitlichem Druck, technischer Rückschläge und paralleler Verpflichtungen gelang es uns, **die Zusammenarbeit als Team stets respektvoll, lösungsorientiert und offen zu gestalten**. Besonders in herausfordernden Phasen war die kontinuierliche, transparente Kommunikation entscheidend, um als Gruppe gemeinsam tragfähige Entscheidungen zu treffen. Diese Erfahrung stärkt nicht nur unsere Fachkompetenz, sondern auch unsere Fähigkeit zur kooperativen Projektarbeit in realen, professionellen Teamsituationen.

12 Ausblick

Obwohl das System bereits zuverlässig arbeitet, sehen wir zahlreiche potenzielle Erweiterungen, die das Chatprogramm funktional und technisch bereichern könnten:

- **Grafische Benutzeroberfläche (GUI):** Eine intuitive, moderne Oberfläche könnte die Nutzung deutlich vereinfachen und benutzerfreundlicher gestalten – ideal auch für weniger technikaffine Nutzer.
- **Gruppenchats:** Aktuell basiert das System auf 1:1-Kommunikation. Eine Erweiterung um Gruppenkommunikation würde den Funktionsumfang erheblich erweitern.
- **Dateifreigabe:** Neben Texten und Bildern könnte das System zukünftig auch andere Dateitypen übertragen – inklusive Fortschrittsanzeige und Fehlerprüfung.
- **Ende-zu-Ende-Verschlüsselung (E2EE):** Um Datenschutz und Sicherheit zu verbessern, wäre eine sichere Verschlüsselung aller Nachrichten (Text wie Bild) ein logischer nächster Schritt.
- **Persistente Peer-Liste:** Eine lokale Speicherung bekannter Peers könnte helfen, Wiederverbindungen nach einem Neustart herzustellen.
- **Mobile Version / Cross-Plattform:** Eine plattformübergreifende Umsetzung (z. B. mit Kivy oder Electron) könnte das Programm auch auf Smartphones oder Tablets nutzbar machen.