

# **Technische Dokumentation für das BYMY – Chat Projekt**

**Bilal, Yunus, Mustafa, Yusuf**

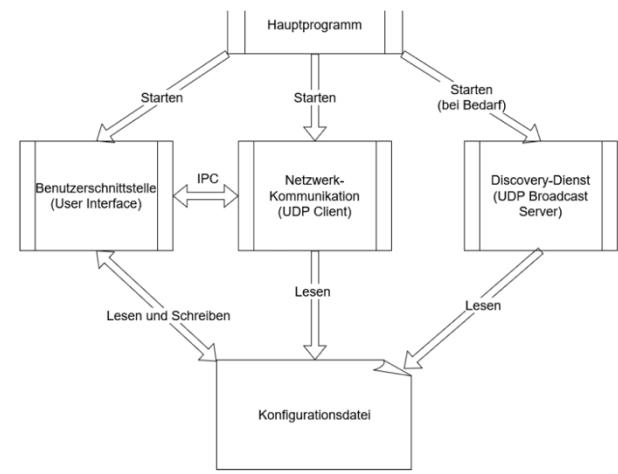
# Einführung in das BYMY-Chat-System:

Das BYMY-Chat System ist eine Kommunikationslösung, die für die Interaktion zwischen Benutzern in einem Peer-to-Peer-Netzwerk entwickelt wurde.

Im Gegensatz zu herkömmlichen Chat-Systemen, die auf zentralen Servern basieren, bietet BYMY eine dezentrale Architektur, die es den Benutzern ermöglicht, direkt miteinander zu kommunizieren, ohne die Notwendigkeit eines zentralen Servers. Diese Lösung ist besonders für Umgebungen geeignet, in denen Datenschutz und Unabhängigkeit von zentralen Instanzen von Bedeutung sind.

Das System besteht aus mehreren Modulen, die in Python geschrieben sind und in einem Linux-Umfeld ausgeführt werden.

- Die Hauptfunktionen des Systems umfassen die Kommunikation von Textnachrichten sowie der Versand von Bildern.
- Der Discovery-Prozess sorgt dafür, dass alle Benutzer im selben Netzwerk automatisch erkannt werden, während der Netzwerk-Prozess für das Senden und Empfangen von Nachrichten und Bildern zuständig ist.
- Das CLI (Command Line Interface) ermöglicht es Benutzern, über ein Terminal-Interface mit dem System zu interagieren.



Die Anwendung nutzt UDP (User Datagram Protocol) für schnelle Nachrichtenübertragungen und TCP (Transmission Control Protocol) für den zuverlässigen Versand von Bildern. Beide Kommunikationsprotokolle ermöglichen eine effiziente und robuste Interaktion zwischen den Benutzern, wobei TCP sicherstellt, dass die Bilddaten vollständig und fehlerfrei ankommen.

Ziel dieses Projekts ist es, ein flexibles und skalierbares Chat-System zu schaffen, das nicht nur die Kommunikation zwischen Benutzern ermöglicht, sondern auch leicht erweitert und an verschiedene Anforderungen angepasst werden kann. Es ist modular aufgebaut, sodass Entwickler problemlos neue Funktionen hinzufügen oder bestehende erweitern können.

Diese Dokumentation bietet eine detaillierte Erklärung der Systemarchitektur, beschreibt die Installation und Konfiguration der benötigten Komponenten und erläutert die verschiedenen Funktionen des Systems.

# Anleitung zur Installation und Ausführung des BYMY-Chats

Das BYMY-Chat System ist eine dezentrale Chat-Anwendung, die es Benutzern ermöglicht, direkt miteinander zu kommunizieren, ohne auf einen zentralen Server angewiesen zu sein. Diese Anleitung führt den Nutzer durch die Installation, Konfiguration und Ausführung des Systems auf einem Linux-basierten Betriebssystem.

## 1. Systemvoraussetzungen

Bevor das System installiert wird, sollte sichergestellt werden, dass die folgenden Softwarekomponenten auf dem Computer vorhanden sind:

- ⇒ **Betriebssystem:** Linux (Ubuntu, Fedora, etc.)
- ⇒ **Python 3.8+** und **pip** (Python-Paketmanager)
- ⇒ **Visual Studio Code** (VS-Code) zum Bearbeiten des Codes
- ⇒ **Terminal** für die Befehlszeileninteraktionen

## 2. Projekt-Dateien herunterladen

Das Projekt kann entweder durch Klonen des Repositories oder durch Herunterladen der ZIP-Datei bezogen werden.

```
git clone <repository-url>
cd <repository-name>
```

## 3. Abhängigkeiten installieren

Das BYMY-Chat-System benötigt zwei externe Python Bibliotheken:

```
pip3 install prompt_toolkit
pip3 install toml
```

Die übrigen benötigten Bibliotheken (wie os, sys, time usw.) sind Teil der Python-Standardbibliothek und müssen nicht separat installiert werden.

## 4. Konfiguration anpassen

Die Datei config.toml sollte geöffnet und der **handle** (Benutzername) angepasst werden.

```
handle = "Username"
autoreply = "Ich bin gerade nicht da."
```

**WICHTIG:** Die restlichen Angaben sollten nicht geändert werden.

## 5. System starten

Sobald die Konfiguration abgeschlossen ist, sollte sichergestellt werden, dass die Skripte ausführbar sind.

```
chmod +x main.sh  
chmod +x stop_all.sh
```

Das Haupt-Skript kann wie folgt gestartet werden:

```
./main.sh
```

Dieses Skript startet alle erforderlichen Prozesse (Discovery-, Netzwerk- und CLI-Prozesse). Das stop\_all-Skript wird automatisch beim Verlassen des Systems mit dem **exit**-Befehl ausgeführt, daher ist es nicht notwendig, es separat zu starten.

## 6. Einzelne Prozesse starten (optional)

Das Programm kann nicht nur über das main-Skript gestartet werden, die Prozesse können auch einzeln manuell starten.

Dazu können folgende Befehle für jedes Modul in separate Terminals verwendet werden:

```
python3 discovery_process.py      # Discovery-Prozess  
python3 network_process.py        # Netzwerk-Prozess  
python3 cli_process.py           # CLI-Prozess
```

# Systemarchitektur und Kommunikation zwischen den Modulen

Das BYMY-Chat-System ist modular aufgebaut, um eine flexible, skalierbare und erweiterbare Lösung zu bieten. Es basiert auf einer dezentrale Architektur, bei der mehrere Prozesse parallel laufen und über definierte Kommunikationskanäle miteinander verbunden sind. Diese Module arbeiten asynchron, was bedeutet, dass sie gleichzeitig verschiedene Aufgaben ausführen, ohne sich gegenseitig zu blockieren.

Hauptmodule:

- **Discovery-Prozess:** Verantwortlich für die Erkennung neuer Benutzer im Netzwerk.
- **Netzwerk-Prozess:** Zuständig für das Senden und Empfangen von Nachrichten und Bildern.
- **CLI-Prozess:** Erlaubt die Benutzerinteraktion über die Kommandozeile.
- **Konfigurationsmodul:** Handhabt die zentrale Konfigurationsdatei (config.toml), die die Benutzereinstellungen wie Handle und automatische Antworten enthält.

Die Kommunikation zwischen den Modulen erfolgt über Pipes und Netzwerkverbindungen, wodurch eine nahtlose Interaktion zwischen den Prozessen ermöglicht wird.

## Kommunikation zwischen den Prozessen

Die Module des Systems kommunizieren hauptsächlich auf zwei Wegen:

1. **Pipes:** Diese dienen als Kommunikationskanäle für die Übertragung von Daten zwischen den Prozessen. Die wichtigsten Pipes im System sind:
    - cli\_to\_network.pipe: Verbindet den CLI-Prozess mit dem Netzwerk-Prozess, um Nachrichten und Befehle zu übermitteln.
    - network\_to\_cli.pipe: Sorgt dafür, dass der Netzwerk-Prozess Nachrichten und Bildinformationen an den CLI-Prozess zurücksendet, damit der Benutzer diese sehen kann.
  2. **Netzwerkkommunikation:** Für die Übertragung von Nachrichten und Bildern zwischen den Benutzern werden UDP (für Textnachrichten) und TCP (für die zuverlässige Bildübertragung) verwendet. Diese Protokolle gewährleisten, dass die Kommunikation schnell und zuverlässig erfolgt.
- **UDP:** Bietet die Grundlage für schnelle Nachrichtenübermittlung. Das Protokoll ist nicht garantierend, aber effizient für unkritische Nachrichten.
  - **TCP:** Wird für die Bildübertragung verwendet, um sicherzustellen, dass große Datenmengen fehlerfrei und vollständig übermittelt werden.

## **Modulare Architektur und Verantwortlichkeiten**

Jedes Modul im BYMY-Chat-System übernimmt eine spezifische Funktion und ist auf den asynchronen Betrieb ausgelegt, sodass die Kommunikation unabhängig voneinander ablaufen kann. Dies fördert eine klare Trennung der Aufgaben und erlaubt eine einfache Erweiterung und Wartung des Systems.

Discovery-Prozess:

- Zweck: Erkennt und verwaltet alle Benutzer, die dem Netzwerk beigetreten sind. Der Prozess sorgt dafür, dass neue Benutzer automatisch erkannt werden, sobald sie ins Netzwerk aufgenommen werden.
- Funktion: Nutzt UDP-Broadcasts, um regelmäßig nach neuen Benutzern zu suchen. Sobald ein Benutzer beitritt, wird dieser in allen relevanten Systemen registriert.

Netzwerk-Prozess:

- Zweck: Verantwortlich für das Senden und Empfangen von Nachrichten und Bildern.
- Funktion: Der Prozess empfängt Nachrichten und Bilddaten über Pipes vom CLI-Prozess und sendet diese an den richtigen Empfänger. Textnachrichten werden mit UDP und Bilder mit TCP gesendet.

CLI-Prozess:

- Zweck: Bietet die Benutzeroberfläche über das Terminal, über die Nachrichten gesendet und empfangen werden können.
- Funktion: Der CLI-Prozess empfängt Benutzereingaben, sendet diese über die Pipe an den Netzwerk-Prozess und zeigt empfangene Nachrichten an. Er ermöglicht es dem Benutzer, mit dem System zu interagieren, z. B. durch Senden von Nachrichten, Ändern der Konfiguration oder Beenden der Sitzung.

Konfigurationsmodul:

- Zweck: Verarbeitet die Konfigurationsdatei config.toml und stellt sicher, dass alle Systemparameter korrekt geladen werden.
- Funktion: Lädt Einstellungen wie den Handle des Benutzers und die Autoreply-Nachricht. Diese Konfiguration ist zentral für den Betrieb des Systems.

# Kommunikationsflüsse und Datenübertragung

## Nachrichtenübertragung

Die Übertragung von Nachrichten im BYMY-Chat-System erfolgt über UDP. Dieser Schritt beschreibt den gesamten Fluss von Textnachrichten im System, beginnend mit der Benutzereingabe und der endgültigen Anzeige der Nachricht beim Empfänger.

1. Benutzereingabe:
  - o Der Benutzer gibt eine Nachricht über das CLI-Interface ein.
  - o Der CLI-Prozess verarbeitet die Benutzereingabe und sendet die Nachricht über eine Pipe an den Netzwerk-Prozess.
2. Nachricht an den Empfänger:
  - o Der Netzwerk-Prozess nimmt die Nachricht von der Pipe entgegen.
  - o Die Nachricht wird dann über UDP an die IP-Adresse und den Port des Empfängers gesendet.
3. Empfänger:
  - o Der Empfänger empfängt die Nachricht über UDP und zeigt sie im CLI an.
  - o Der CLI-Prozess des Empfängers zeigt die Nachricht im Terminal an, damit der Benutzer sie lesen kann.

## Bildübertragung

Die Übertragung von Bildern ist etwas komplexer und erfolgt über TCP, um sicherzustellen, dass die Bilddaten zuverlässig und in ihrer vollständigen Form ankommen.

1. Bildauswahl:
  - o Der Benutzer wählt ein Bild aus, das er versenden möchte.
  - o Das Bild wird vom CLI-Prozess an den Netzwerk-Prozess weitergeleitet.
2. Bildübertragung:
  - o Der Netzwerk-Prozess stellt eine TCP-Verbindung zum Empfänger her. TCP wird hier verwendet, weil es eine zuverlässige Übertragung sicherstellt und gewährleistet, dass die Bilddaten in ihrer Gesamtheit und ohne Fehler ankommen.
  - o Das Bild wird in kleinen Datenpaketen über die TCP-Verbindung übertragen. Jedes Paket enthält einen Teil der Bilddaten, und die vollständigen Daten werden erst dann angezeigt, wenn alle Pakete erfolgreich übertragen wurden.
3. Empfänger:
  - o Der Empfänger empfängt das Bild über TCP, speichert es in dem vom Programm erstellten receive Ordner.

## Protokollwahl: UDP vs. TCP

Die Wahl der Protokolle spielt eine zentrale Rolle bei der Effizienz und Zuverlässigkeit des Systems:

- UDP (User Datagram Protocol):
  - **Vorteile:** Schnell und effizient, ideal für die Übertragung von kurzen Textnachrichten.
  - **Nachteile:** Da es keine Garantie für die Zustellung gibt, kann es zu Paketverlusten kommen.
  - **Verwendung im System:** Für Textnachrichten, bei denen Geschwindigkeit wichtiger ist als die Garantie der vollständigen Zustellung.
- TCP (Transmission Control Protocol):
  - **Vorteile:** Garantiert die fehlerfreie und vollständige Übertragung von Daten.
  - **Nachteile:** Langsamere Übertragung als UDP aufgrund der zusätzlichen Überprüfungen und Bestätigungen.
  - **Verwendung im System:** Für die Bildübertragung, bei der eine vollständige und fehlerfreie Übertragung entscheidend ist.

# Fehlerbehandlung und häufige Probleme

In diesem Abschnitt wird erklärt, wie das BYMY-Chat-System mit typischen Fehlern umgeht, die bei der Nutzung oder beim Betrieb auftreten können. Eine gute Fehlerbehandlung ist entscheidend, um sicherzustellen, dass das System auch bei unerwarteten Problemen zuverlässig funktioniert.

## Probleme mit der Kommunikation

UDP-Probleme:

- Symptom: Nachrichten kommen nicht an oder werden verzögert zugestellt.
- Ursache: UDP ist ein unzuverlässiges Protokoll, und es gibt keine Garantie für die Zustellung von Nachrichten.
- Lösung:
  - Das System überprüft regelmäßig, ob eine Nachricht bestätigt wurde. Wenn keine Bestätigung kommt, wird die Nachricht erneut gesendet.
  - Bestätigungssystem: Wenn der Empfänger keine Bestätigung der Nachricht sendet, wird sie vom Absender wiederholt.

TCP-Probleme:

- Symptom: Die Bildübertragung wird unterbrochen oder Daten gehen verloren.
- Ursache: TCP-Verbindungsabbrüche oder Netzwerkinstabilität.
- Lösung:
  - Das System stellt bei Verbindungsabbrüchen automatisch eine neue Verbindung her und setzt die Übertragung fort.
  - Wiederholungsmechanismus: Sollte ein Paket fehlen oder die Verbindung unterbrochen werden, stellt das System sicher, dass fehlende Daten erneut gesendet werden.

## Fehler bei der Paketübertragung:

- Symptom: Übertragene Nachrichten oder Bilder erscheinen fehlerhaft oder unvollständig.
- Ursache: Datenverlust oder Fehler bei der Übertragung.
- Lösung:
  - Fehlererkennung und Wiederholung der Übertragung werden durch die Wahl von TCP für die Bildübertragung und UDP für Nachrichten gewährleistet. Für UDP-Nachrichten werden Bestätigungen und erneute Versuche implementiert.

## Probleme mit den Pipes

Fehlende oder defekte Pipes:

- Symptom: Fehlermeldungen wie „Pipe nicht gefunden“ oder „Verbindung konnte nicht hergestellt werden“.
- Ursache: Die Pipe-Dateien wurden nicht korrekt erstellt oder sind defekt.
- Lösung: Es wird empfohlen, sicherzustellen, dass alle benötigten Pipes (cli\_to\_network.pipe und network\_to\_cli.pipe) korrekt erstellt werden. Dies erfolgt durch den Befehl mkfifo:

```
mkfifo cli_to_network.pipe  
mkfifo network_to_cli.pipe
```

## Fehler bei der Kommunikation über die Pipes:

- Symptom: Nachrichten oder Daten werden nicht zwischen den Prozessen übertragen.
- Ursache: Es kann ein Problem mit den Pipe-Verbindungen oder einem Blockieren eines Prozesses vorliegen.
- Lösung:
  - Das System sollte sicherstellen, dass alle Prozesse korrekt ausgeführt werden, ohne dass sie durch Fehler oder Blockierungen gestoppt werden.
  - Die Pipe-Verbindungen sollten in jedem Terminalfenster korrekt geöffnet werden, bevor der jeweilige Prozess gestartet wird.

## Fehlende Abhängigkeiten und Installationsprobleme

Fehlende Python-Bibliotheken:

- Symptom: Fehler beim Importieren von Modulen wie prompt\_toolkit oder toml.
- Ursache: Die erforderlichen Python-Bibliotheken wurden nicht installiert.
- Lösung: Es wird empfohlen, sicherzustellen, dass die Bibliotheken mit pip installiert werden:

```
pip3 install prompt_toolkit toml
```

## Inkompatible Versionen:

- Symptom: Inkompatibilitätsfehler beim Starten des Systems.
- Ursache: Eine veraltete Python-Version oder inkompatible Bibliotheksversionen.
- Lösung: Das System setzt voraus, dass Python 3.8+ installiert ist. Überprüfen Sie mit pip3 list, ob alle erforderlichen Bibliotheken in der richtigen Version vorliegen.

## Probleme bei der Konfiguration

Fehlerhafte config.toml-Datei:

- Symptom: Das System startet nicht oder verhält sich unerwartet.
- Ursache: Eine fehlerhafte Konfiguration in der config.toml-Datei.
- Lösung: Es ist wichtig, dass alle erforderlichen Parameter wie handle und autoreply korrekt gesetzt sind:

```
handle = "DeinBenutzername"
autoreply = "Ich bin gerade nicht da."
```

Wichtig: Andere Felder wie port und whoisport sollten nicht verändert werden, es sei denn, der Benutzer ist sich sicher, was er tut.

## Allgemeine Fehlerbehebung

System startet nicht:

- Symptom: Das System startet nicht oder zeigt Fehler an.
- Ursache: Probleme mit den Skripten oder nicht ausführbaren .sh-Dateien.
- Lösung: Der Benutzer sollte sicherstellen, dass alle Skripte ausführbar sind:

```
chmod +x main.sh
chmod +x stop_all.sh
```

## Fehler beim Beenden des Systems:

- Symptom: Das System wird nicht richtig beendet oder Prozesse bleiben aktiv.
- Ursache: Ein Problem mit dem stop\_all.sh-Skript oder der Kommunikation zwischen den Prozessen.
- Lösung:
  - Der Benutzer sollte den Exit-Befehl ausführen, um sicherzustellen, dass alle Prozesse korrekt beendet werden.
  - Falls das nicht funktioniert, kann das stop\_all.sh-Skript manuell ausgeführt werden, um alle Prozesse zu stoppen.