# Detaillierte Zusammenfassung: Client-Server-Kommunikation

## Einleitung

Diese Zusammenfassung beschreibt die Funktionsweise einer TCP-basierten Client-Server-Kommunikation, wie sie in C++ implementiert wurde. Der Server wartet auf Verbindungen, akzeptiert diese und verarbeitet Nachrichten. Der Client stellt eine Verbindung her, sendet Nachrichten und empfÃ¤ngt Antworten. Die gesamte Kommunikation erfolgt in einer klar definierten, sequentiellen Abfolge.

## 1. Allgemeine Architektur

1. Der Server wartet auf Verbindungen von Clients, akzeptiert diese und verarbeitet Nachrichten. Er kann auch Antworten senden.
2. Der Client stellt eine Verbindung zum Server her, sendet Nachrichten und empfÃ¤ngt Antworten.

## 2. Sequenzielle Abfolge der Interaktionen

### A) Server startet:

1. Server erstellt einen passiven Socket (socket).
2. Server bindet den Socket an eine Adresse und einen Port (bind).
3. Server schaltet den Socket in den Listen-Modus (listen).
4. Server wartet auf Verbindungen (accept).

### B) Client startet:

1. Client erstellt einen aktiven Socket (socket).
2. Client konfiguriert die Serveradresse (inet\_pton, htons).
3. Client versucht, sich mit dem Server zu verbinden (connect).

### C) Kommunikation zwischen Client und Server:

1. **Client sendet eine Nachricht**:
2. send(activeSocket, msg.c\_str(), msg.length(), 0).
3. **Server empfÃ¤ngt die Nachricht**:
4. recv(activeSocket, comBuffer, BUFFER\_SIZE, 0).
5. **Server sendet eine Antwort**:
6. send(activeSocket, reply.c\_str(), reply.length(), 0).
7. **Client empfÃ¤ngt die Antwort**:
8. recv(activeSocket, comBuffer, BUFFER\_SIZE, 0).

### D) Beenden der Verbindung:

1. Der Client beendet die Verbindung (close).
2. Der Server erkennt dies und beendet die Verarbeitung (close).

## 3. Zusammenfassung der wichtigsten Methoden

### Server:

1. socket: Erstellt einen passiven Socket.
2. bind: VerknÃ¼pft den Socket mit Adresse und Port.
3. listen: Versetzt den Socket in den Listen-Modus.
4. accept: Akzeptiert eine Client-Verbindung.
5. recv: EmpfÃ¤ngt Daten vom Client.
6. send: Sendet Daten an den Client.
7. close: SchlieÃŸt den aktiven Socket.

### Client:

1. socket: Erstellt einen aktiven Socket.
2. inet\_pton: Konvertiert die Server-IP.
3. connect: Stellt eine Verbindung zum Server her.
4. send: Sendet Daten an den Server.
5. recv: EmpfÃ¤ngt Daten vom Server.
6. close: SchlieÃŸt die Verbindung.

## 4. Beispielkommunikation

Angenommen, der Server und der Client sind gestartet. Der folgende Dialog beschreibt, was geschieht:

1. **Client initiiert die Verbindung:**
2. Der Client ruft connect auf, um sich mit dem Server zu verbinden.
3. Der Server erkennt die Anfrage und akzeptiert sie (accept).
4. **Client sendet eine Nachricht:**
5. Client: "Hallo Server" -> ruft send auf.
6. Server empfÃ¤ngt die Nachricht mit recv.
7. **Server verarbeitet und antwortet:**
8. Der Server verarbeitet die Nachricht ("Hallo Server") und antwortet mit "Hallo Client".
9. Server ruft send auf, um die Antwort zu schicken.
10. **Client empfÃ¤ngt die Antwort:**
11. Der Client empfÃ¤ngt "Hallo Client" mit recv.
12. **Client beendet die Verbindung:**
13. Der Client sendet "quit" und ruft close auf.
14. Der Server erkennt die Trennung und schlieÃŸt die Verbindung.