Header-Based Group Chat Protocol Documentation

Protokoll-Überblick

Das header-basierte Protokoll orientiert sich an HTTP und verwendet eine klare Struktur mit Headern und optionalem JSON-Body. Alle Nachrichten (Server-Client, UDP P2P, TCP P2P) verwenden das gleiche Format.

Nachrichtenstruktur

```
MESSAGE_TYPE PROTOCOL_VERSION
Header-Key: Header-Value
Header-Key: Header-Value
...
[Leerzeile]
[JSON-Body]
```

Beispiel:

```
REGISTER 1.0
Host: 192.168.1.100
Content-Length: 67

{"nickname": "Alice", "ip": "192.168.1.100", "udp_port": 12345}
```

Kommunikationsarten

1. Client ↔ Server (TCP)

- **Zweck**: Registrierung, Benutzerverwaltung, Broadcast-Nachrichten
- **Transport**: TCP (persistent connection)
- **Port**: 8888 (Standard)

2. Client ↔ Client UDP (Chat-Initiierung)

- **Zweck**: Chat-Anfragen und -Antworten
- **Transport**: UDP (connectionless)
- **Port**: Dynamisch zugewiesen

3. Client ↔ Client TCP (Chat-Sessions)

- **Zweck**: Direkter Peer-to-Peer Chat
- **Transport**: TCP (session-based)

- **Port**: Dynamisch zugewiesen

Vollständige Nachrichtenliste

Server ↔ Client Nachrichten (TCP)

Client → Server

Nachricht	Zweck	Body	Zusätzliche Header
`REGISTER`	Client-Registrierung	`{"nickname": "string", "ip": "string", "udp_port": number}`	`Host: client_ip`
`UNREGISTER`	Client-Abmeldung	Leer	`Host: client_ip`
`BROADCAST`	Broadcast-Nachricht senden	`{"message": "string"}`	`Host: client_ip`

Server → Client

Nachricht	Zweck	Body	Zusätzliche Header
`REGISTER_OK`	Registrierung erfolgreich	`{"message": "string"}`	-
`USER_LIST`	Aktuelle Benutzerliste	`{"users": [{"nickname": "string", "ip": "string", "udp_port": number}, ...]}`	-
`USER_JOINED`	Neuer Benutzer beigetreten	`{"nickname": "string", "ip": "string", "udp_port": number}`	-
`USER_LEFT`	Benutzer hat verlassen	`{"nickname": "string"}`	-
`BROADCAST_MSG`	Broadcast-Nachricht empfangen	`{"sender": "string", "message": "string"}`	-
`BROADCAST_OK`	Broadcast erfolgreich gesendet	`{"message": "string"}`	-
`ERROR`	Fehlermeldung	`{"message": "string", "code": number}`	-

Client ↔ Client UDP Nachrichten (Chat-Initiierung)

Nachricht	Richtung	Zweck	Body	Header

`CHAT_REQUEST`	Initiator → Ziel	Chat-Anfrage	`{"tcp_port": number}`	`From: nickname`
				`To: nickname`
				`Request-ID: unique_id`
				`Host: sender_ip`

`CHAT_RESPONSE`	Ziel → Initiator	Chat-Antwort	`{"accepted": boolean, "tcp_port": number}`	`From: nickname`
				`To: nickname`
				`Request-ID: same_as_request`
				`Host: sender_ip`

Client ↔ Client TCP Nachrichten (Chat-Sessions)

Nachricht	Zweck	Body	Header
`CHAT_HELLO`	Session-Initialisierung	`{"nickname": "string"}`	`From: nickname` `Host: sender_ip`

`CHAT_MSG`	Chat-Nachricht	`{"message": "string"}`	`From: nickname`
			`To: nickname`
			`Timestamp: HH:MM:SS`
			`Host: sender_ip`

`CHAT_CLOSE`	Session beenden	Leer	`From: nickname`
			`To: nickname`
			`Host: sender_ip`

Protokoll-Ablauf

1. Client-Start und Registrierung

```
1. Client → Server: REGISTER
   REGISTER 1.0
   Host: 192.168.1.100
   Content-Length: 67

   {"nickname": "Alice", "ip": "192.168.1.100", "udp_port": 12345}

2. Server → Client: REGISTER_OK + USER_LIST
   REGISTER_OK 1.0
   Content-Length: 35

   {"message": "Erfolgreich registriert"}

3. Server → Client: USER_LIST
   USER_LIST 1.0
   Content-Length: 120

   {"users": [{"nickname": "Bob", "ip": "192.168.1.101", "udp_port": 12346}]}
```

2. Broadcast-Nachricht

```
1. Client → Server: BROADCAST
   BROADCAST 1.0
   Host: 192.168.1.100
   Content-Length: 25
   
   {"message": "Hallo alle!"}

2. Server → Alle Clients: BROADCAST_MSG
   BROADCAST_MSG 1.0
   Content-Length: 45
   
   {"sender": "Alice", "message": "Hallo alle!"}

3. Server → Sender: BROADCAST_OK
   BROADCAST_OK 1.0
   Content-Length: 30
   
   {"message": "Broadcast gesendet"}
```

3. Peer-to-Peer Chat-Initiierung

1. Alice → Bob (UDP): CHAT_REQUEST
   CHAT_REQUEST 1.0
   From: Alice
   To: Bob
   Host: 192.168.1.100
   Request-ID: Alice_1640995200_1234
   Content-Length: 18

   {"tcp_port": 54321}

2. Bob → Alice (UDP): CHAT_RESPONSE
   CHAT_RESPONSE 1.0
   From: Bob
   To: Alice
   Host: 192.168.1.101
   Request-ID: Alice_1640995200_1234
   Content-Length: 35

   {"accepted": true, "tcp_port": 54322}

3. Alice → Bob (TCP): CHAT_HELLO
   CHAT_HELLO 1.0
   From: Alice
   Host: 192.168.1.100
   Content-Length: 20

   {"nickname": "Alice"}

4. Chat-Session

```
1. Alice → Bob: CHAT_MSG
   CHAT_MSG 1.0
   From: Alice
   To: Bob
   Host: 192.168.1.100
   Timestamp: 14:30:15
   Content-Length: 35

   {"message": "Hallo, wie geht's?"}

2. Bob → Alice: CHAT_MSG
   CHAT_MSG 1.0
   From: Bob
   To: Alice
   Host: 192.168.1.101
   Timestamp: 14:30:18
   Content-Length: 25

   {"message": "Gut, danke!"}

3. Alice → Bob: CHAT_CLOSE
   CHAT_CLOSE 1.0
   From: Alice
   To: Bob
   Host: 192.168.1.100
   Content-Length: 0

   [Kein Body]
```

Standard-Header

Pflicht-Header (alle Nachrichten)

- **Erste Zeile**: `MESSAGE_TYPE PROTOCOL_VERSION`
- **Content-Length**: Größe des JSON-Body in Bytes

Optionale/Spezifische Header

- **Host**: IP-Adresse des Senders
- **From**: Nickname des Senders (P2P)
- **To**: Nickname des Empfängers (P2P)
- **Request-ID**: Eindeutige ID für Request/Response-Zuordnung
- **Timestamp**: Zeitstempel für Chat-Nachrichten

Fehlerbehandlung

Häufige Fehlerszenarien

1. **Unbekannter Nachrichtentyp**

```
ERROR 1.0
Content-Length: 45

{"message": "Unbekannter Nachrichtentyp", "code": 400}
```

2. **Nickname bereits verwendet**

```
ERROR 1.0
Content-Length: 55

{"message": "Nickname bereits vergeben", "code": 409}
```

3. **Ungültiges JSON**

```
ERROR 1.0
Content-Length: 40

{"message": "Ungültiges JSON-Format", "code": 400}
```

Besonderheiten

UDP-Buffer-Größe

- Standard: 4096 Bytes (für Header + JSON)
- Ausreichend für Chat-Anfragen mit Metadaten

TCP-Connection Management

- **Server-Verbindung**: Persistent während gesamter Session
- **P2P-Verbindungen**: Pro Chat eine eigene TCP-Verbindung

Encoding

- Alle Nachrichten: UTF-8
- Zeilentrenner: `\r\n` (CRLF wie HTTP)
- Header-Body-Trennung: `\r\n\r\n`

Threading-Modell

- **Server-Handler**: Separater Thread für Server-Nachrichten
- **UDP-Handler**: Separater Thread für Chat-Anfragen

- **TCP-Chat-Server**: Separater Thread für eingehende Chats
- **Per-Chat-Handler**: Separater Thread pro aktiver Chat-Session

Erweiterbarkeit

Das Protokoll ist durch zusätzliche Header leicht erweiterbar:

- **Priorität**: `Priority: high|normal|low`
- **Verschlüsselung**: `Encryption: AES256`
- **Dateityp**: `Content-Type: image/jpeg`
- **Session-ID**: `Session-ID: unique_session_id`