Spezifikation des Peer-to-Peer Chat-Protokolls (P2P-CP)

Version: 1.0

Status: Entwurf (Draft)

Transport: UDP über IPv4

1. Einleitung und Architektur (Hannes)

1.1 Übersicht

Dieses Dokument spezifiziert ein proprietäres Peer-to-Peer (P2P) Chat-Protokoll für den Nachrichtenaustausch und die Dateiübertragung in lokalen Netzwerken oder dem Internet. Das Protokoll ist darauf ausgelegt, ohne zentralen Server (dezentral) zu operieren.

1.2 Systemarchitektur

- **Topologie:** Teilvermaschtes Peer-to-Peer-Netzwerk (Mesh).
- **Rollenverteilung:** Jeder Knoten (Peer) agiert gleichzeitig als:
    - **Sender/Empfänger:** Erstellung und Konsum von Nachrichten.
    - **Router:** Weiterleitung von Paketen für andere Peers (Forwarding).
    - **Manager:** Verwaltung der Netzwerktopologie und Nachbarschaftslisten.

1.3 Transport & Adressierung

- **Transportprotokoll:** UDP (User Datagram Protocol). Das Protokoll implementiert eigene Mechanismen für Zuverlässigkeit und Reihenfolge (Reliability Layer over UDP).
- **Identifikation:** Ein Knoten wird eindeutig durch das Tupel (IPv4-Adresse, Port) identifiziert.

2. Paketstruktur und Datenformate

Jedes Paket besteht aus einem Protokoll-Header und einem variablen Payload.
Alles soll in Big Endian implementiert sein.

2.1 Header-Definition (Ben)

Der Header wird jedem UDP-Payload vorangestellt.

Feld	Datentyp	Beschreibung
Type	Unsigned Integer (8-bit)	Definiert den Nachrichtentyp (siehe 2.2).
SequenceNumber	Unsigned Integer (32-bit)	Eindeutige, hostsynchrone ID um Chunks zu Identifizieren.
Destination-IP	Unsigned Integer (32-bit)	IPv4-Adresse des finalen Empfängers.
Source-IP	Unsigned Integer (32-bit)	IPv4-Adresse des ursprünglichen Absenders.
Destination-Port	Unsigned Integer (16-bit)	Port des Ziels
Source-Port	Unsigned Integer (16-bit)	Port des Absenders
PayloadLength	Unsigned Integer (32-bit)	Größe der Nutzdaten in Bytes (ohne Header).
Chunk-ID	Unsigned Integer (32-bit)	*Nur bei Dateien:* Laufende Nummer des Datenblocks
Chunk-Length	Unsigned Integer (32-bit)	*Nur bei Dateien und bei File_Info:* Gesamtanzahl der Chunks der Datei
TTL	Unsigned Integer (8-Bit)	*Bei jedem Hop minus 1 TTL*
Checksum	Hash (256-bit)	*Fürs ganze Payload im Hash (sha-256)*

-> 62 Byte Header

2.2 Nachrichtentypen (OpCodes) (Justus)

Die folgenden Codes definieren die Semantik des Pakets:

Typ	Binär-Code	Hex-Code	Beschreibung
ACK	00000001	0x01	Bestätigung, dass ein Frame erhalten wurde. (Nicht für jeden Chunk)
NO_ACK	00000010	0x02	Negative Bestätigung (Anforderung zur erneuten Sendung von Chunks aus spezifischem Frame)
HELLO	00000011	0x03	Anmeldung im Netzwerk (Broadcast an Nachbarn).
GOODBYE	00000100	0x04	Abmeldung vom Netzwerk (Broadcast an Nachbarn).
MSG	00000101	0x05	Übertragung einer Textnachricht. Darf die Größe eines Chunks nicht überschreiten.
FILE_CHUNK	00000110	0x06	Übertragung eines Dateifragments.
FILE_INFO	00000111	0x07	Übermittlung vom Dateinamen
HEART_BEAT	00001000	0x08	Keep-Alive Signal (Ping, 3 Sekunden)zur Verfügbarkeitsprüfung.
ROUTING_UPDATE	00001001	0x09	Übertragung der Routing Tabelle

2.2.1 Definition (John)

Typ 0x01 – ACK

Payload:

- leer (0 Byte)

Header-Anforderungen:

- sequenceNumber = Sequence Number des bestätigten Pakets
- payloadLength = 0

Typ 0x02 – NO_ACK

Payload-Format:

Offset	Länge	Beschreibung
0–3	4 Byte	Sequence Number der betroffenen Chunks
4–5	2 Byte	Missing Count (Anzahl fehlender Chunks, uint16)
6–9	4 Byte	Fehlender Chunk 1 (uint32)
...	je 4 Byte	Weitere fehlende Chunk-IDs

Hinweis:
Es kann sinnvoll sein, Missing Count zu begrenzen (z. B. max. 256).
Wenn mehr Chunks fehlen, müssen mehrere NO_ACK-Pakete erzeugt werden.

Typ 0x03 – HELLO

Payload:

- leer

Header-Anforderungen:

- payloadLength = 0

Typ 0x04 – GOODBYE

Payload:

- leer

Header-Anforderungen:

- payloadLength = 0

Typ 0x05 – MSG

Payload:

- UTF-8 codierter Text

Header-Anforderungen:

- payloadLength = Anzahl der UTF-8-Bytes

Typ 0x06 – FILE_CHUNK

Payload:

- Chunk-Rohdaten

Header-Anforderungen:

- chunkId = Index des Chunks
- chunkLength = Gesamtzahl aller Chunks

payloadLength = Größe des Chunk-Datenblocks in Bytes

Typ 0x07 – FILE_INFO

Payload-Format:

Offset | Länge | Beschreibung
0–n | variabel | filename (UTF-8 codiert)

Header-Anforderungen:

- payloadLength = payloadLength

Typ 0x08 – HEARTBEAT

Payload:

- leer

Header-Anforderungen:

- payloadLength = 0

Typ 0x09 – ROUTING_UPDATE

Payload-Format:

Offset | Länge | Beschreibung
0–1 | 2 Byte | entryCount (uint16)
Danach pro Eintrag:
• 4 Byte destIp (uint32)
• 2 Byte destPort (uint16)
• 1 Byte distance (uint8)

Header-Anforderungen:

- payloadLength = 2 + (entryCount × 7)

3. Kernfunktionalitäten

3.1 Zuverlässigkeitsschicht (Reliability Layer)(Artin)

Da UDP ein verbindungsloses Protokoll ist, implementiert die Anwendung folgende Mechanismen:

Die **SequenceNumber** wird als 32-bit unsigned Integer lokal pro Peer geführt und bei jedem versendeten Datei/Message um 1 erhöht.

MESSAGES:

- **Bestätigung:** Empfangene Pakete vom Typ MSG müssen mit einem ACK beantwortet werden.
- **Duplikaterkennung:** Empfänger führen eine Historie empfangener SequenceNumbers pro Source-IP und Source-Port. Messages mit bereits verarbeiteten Sequenznummern (die wir komplett empfangen haben) werden verworfen und ein neues ACK wird gesendet.
- **Re-Transmission bei fehlendem ACK:** Bleibt ein ACK aus nach n(default 3) Sekunden, müssen wir die Message erneut senden. Dies Versuchen wir 3-mal, bis wir aufgeben (best effort).

FILES:

- Wir arbeiten mit Frames und für jedes Frame wird ein ACK oder ein NO_ACK (mit fehlenden Chunk-IDs) gesendet
- Ein Frame ist 128 Chunks groß
- **Bestätigung:** Wenn alle Chunks eines Frames angekommen sind, müssen sie mit einem ACK beantwortet werden, fehlende Chunks werden gesammelt und mit einem NO_ACK wieder angefragt
- **Duplikaterkennung:** Empfänger führen eine Historie empfangener SequenceNumbers pro Source-IP und Source-Port. Jedes Frame kann durch die Chunk-ID zugeordnet werden, bei doppelten empfangenen Frames
- **Re-Transmission bei fehlenden ACK:** Bleibt ein ACK aus nach n(default 3) Sekunden, muss das Frame erneut gesendet werden. Dies Versuchen wir 3-mal, bis wir aufgeben (best effort), dann löschen wir die zu sendenden Daten.
- **Re-Transmission bei NO_ACK:** Wird ein NO_ACK empfangen, müssen die fehlenden Chunks erneut gesendet werden.

FILE INFO:

- **Bestätigung:** Empfangene Pakete vom Typ File Info müssen mit einem ACK beantwortet werden.
- **Duplikaterkennung:** Empfänger führen eine Historie empfangener SequenceNumbers pro Source-IP und Source-Port. File Info mit bereits verarbeiteten Sequenznummern (die wir komplett empfangen haben) werden verworfen und ein neues ACK wird gesendet.
- **Re-Transmission bei fehlendem ACK:** Bleibt ein ACK aus nach n(default 3) Sekunden, müssen wir die File Info erneut senden. Dies Versuchen wir 3-mal, bis wir aufgeben (best effort).

3.2 Nachrichtenaustausch (Messaging) (Lukas)

- Unterstützung für Unicast-Kommunikation zu direkten Nachbarn und entfernten Zielen (via Multi-Hop Routing).
- Nachrichten werden im Payload des Typs MSG transportiert.

3.3 Dateiübertragung (Lukas)

Dateien werden in Fragmente (Chunks) zerlegt, um die MTU nicht zu überschreiten.

- Fragmentierung:

Die maximale Größe eines Daten-Chunks berechnet sich wie folgt, um IP-Fragmentierung zu vermeiden:

(VPN ist Parametisiert)

Chunk.Size = MTU - IP_Header- UDP_Header – Protokoll_Header –PufferVPN)

Chunk.Size = 1500 - 20 - 8 - 62 - 150 = 1260 Byte

(Standardannahme für MTU im Ethernet: 1500 Bytes)

- **Ablauf:**
  - Sender zerlegt Datei in N Chunks.
  - Versand als FILE_CHUNK. Das Feld Chunk-Length enthält den Wert N (Gesamtanzahl).
  - Chunk-ID identifiziert sequenziell den aktuellen Index.

- Empfänger setzt Datei zusammen.
- Selective Repeat

4. Routing und Topologie-Management

Das Protokoll verwendet einen **Distance-Vector-Routing-Algorithmus**.

4.1 Routing-Tabelle (Julia)

Jeder Knoten pflegt eine Routing-Tabelle mit folgenden Einträgen:

Feld	**Beschreibung**
Destination-IP	IP-Adresse des erreichbaren Ziels
Destination - Port	Port des erreichbaren Ziels
Next-Hop-IP	IP des *direkten* Nachbarn, an den weitergeleitet wird.
Next-Hop-Port	Portnummer des *direkten* Nachbarn, an den weitergeleitet wird.
Distance	Metrik (Hop Count) zum Ziel.

4.2 NextNeighbor-Tabelle (Julia)

Neighbor IP	IP-Adresse der Nachbar
Neighbor Port	IP-Port der Nachbar
Last heard	Timer (Heartbeat awaits)
alive	Boolean

4.2.1 **NextNeighbor-Typen Beschreibung**

Last heard wartet Heartbeat timer x 2 + 1 Sekunden auf ein Heartbeat oder bei Erreichen von einem Paket (Chunk, Msg, etc.).

Alive wird auf false gesetzt so bald last heard > HeartBeat timer $* \mathbf{2} + \mathbf{1}$ oder Neighbor sendet GOODBYE

Wenn Alive == false.

Alle Einträge in der Routingtabelle die Über Neighbor ip + Neighbor Port gehen löschen.
Einträge in der Neighbortabelle bleiben erhalten, aber alive=false

4.3 Netzwerkteilnahme (Discovery) (Hannes)

- **Beitritt (Join):** Ein neuer Knoten sendet HELLO an bekannte Einstiegspunkte per Broadcast auf Anwendungsebene. Nachbarn fügen Knoten in Neighbor-Tabelle (falls schon existent dann stelle alive auf true) und erstellen Route zu dem Knoten, der ein Routing Update triggert.
- **Austritt (Leave):** Ein Knoten sendet GOODBYE an alle direkten Nachbarn per Broadcast auf Anwendungsebene. Nachbarn setzen Knoten auf Tot in der Neighbor-Tabelle und triggern ein Routing Update.

4.4 Routing-Updates (Jannis)

Updates der Routing-Informationen erfolgen **eventbasiert** und periodisch (Update Timer):

1. **Trigger:** Empfang von HELLO, GOODBYE, Tod oder Routing Tabelle sich verändert, timer
2. Routing Tabelle mit Poison Reverse und Split Horizon aktualisieren
3. ROUTING_UPDATE an alle Nachbarn senden

4.4.1 Routing Update Payload

Der gesamte UDP-Payload (nach dem Protokoll-Header) sieht im Speicher wie folgt aus.

Byte 0-1	Byte 2-5	Byte 6-7	Byte 8	Byte 9-12	
Entry-Count	Dest-IP (A)	Dest-Port(A)	Distance (A)	Dest-IP (B)	...

Der Empfänger muss die empfangene Liste durchgehen und jeden Eintrag überprüfen.

4.4.1.1 Vorbereitung:

1. **Sender-Identifizierung:** Lese die **Source-IP** und den **Source-Port** aus dem Protokoll Header des empfangenen Pakets. Dieser Knoten ist der **kandidierende Next-Hop** (NextHopCandidate).
2. **Status-Flag:** Setze eine lokale Variable TableChanged = false.

4.4.1.2 Schritte zur Tabellenaktualisierung:

1. **Payload parsen:** Lese den **Entry-Count** (N) aus den ersten 2 Bytes des Payloads.
2. **Schleife starten:** Iteriere N-mal durch die folgenden 7-Byte-Blöcke (ein Block pro Ziel).
3. **Für jeden Eintrag:**
    a. Lese die **Destination-IP, Destination-Port** und die **Distance-Metric** (ReceivedDistance).
    b. Berechne die Neue Distanz (NewDistance) über den NextHopCandidate:

NewDistance = ReceivedDistance + 1

    c. Suche in deiner **lokalen Routingtabelle** nach einem Eintrag für diese Destination-IP + Destination-Port.
4. **Entscheidungslogik (Shortest Path Wins):**

Szenario	Aktion	Status
Szenario A: Ziel ist **neu** in deiner Tabelle.	**Hinzufügen.** Erstelle neuen Eintrag: Next-Hop = Source-IP + Port, Distance = NewDistance.	TableChanged = true
Szenario B: NewDistance ist **kleiner** als deine CurrentDistance.	**Update.** Aktualisiere den Eintrag: Next-Hop = Source-IP + Port, Distance = NewDistance.	TableChanged = true
Szenario C: Der aktuelle Next-Hop ist **gleich** dem NextHopCandidate.	**Update/Wartung.** Aktualisiere die Distance = NewDistance. Dies ist notwendig, um Pfadänderungen nachzuverfolgen oder Schleifen zu erkennen. (Poison reverse)	TableChanged = true
Szenario D: NewDistance ist größer und der Next-Hop ist anders.	**Ignorieren.** (Der aktuelle Weg ist besser.)	Nichts

4.5 Heartbeat und Fehlererkennung (Bohdan)

Die Lebendigkeit von Nachbarn wird aktiv überwacht.

- **Mechanismus:** Periodisches Senden von HEART_BEAT-Paketen an direkte Nachbarn.
- **Inhalt:** Leerer Payload (Ping-Funktionalität).
- **Timeout-Regel:**
  - Alle 3 Sekunden soll ein HEART_BEAT gesendet werden
  - Erhält ein Knoten (HEART_BEAT $* \mathbf{2} + \mathbf{1}$) Sekunden lang keinen HEART_BEAT von einem Nachbarn, gilt dieser als "Tot".
  - **Reaktion:** Nachbar wird auf Tot gestellt. Alle Routen über diesen Nachbarn werden gelöscht. Ein sofortiges Routing-Update wird ausgelöst.

5. Stabilitäts- und Schutzmechanismen

5.1 Schleifenvermeidung (Loop Prevention) - Pedro

- **TTL (Time-to-Live):** Jedes Paket/Chunk besitzt eine logische TTL, bei jedem Hop wird die TTL um 1 verringert. Ist TTL = 0, wird das Paket/Chunk verworfen. Unsere standard ttl ist 64
- **Update-Kontrolle:** Ein Routing-Update wird *nicht* weitergeleitet, wenn die empfangene Information bereits mit der eigenen Tabelle übereinstimmt
- **Poison Reverse:** um Infinity zu senden nutzen wir den maximalen uint8 wert. Nachdem ein wert poisoned wurde, sollte er nicht direkt, sondern erst nach ein paar Sekunden gelöscht werden (heartbeat_timer x3)
- **Split Horizon**

5.2 Paketflusskontrolle (Fridi)

- **Routing-Logik:**
    - Ist Destination-IP und Port == Eigene IP und Port? -> Verarbeiten.
    - Ist Destination-IP und Port in Routing-Tabelle? -> Weiterleiten an Next-Hop.
    - Ziel unbekannt? -> Verwerfen.

5.3 Datenintegrität (Fridi)

- SHA 256 Checksum vom Payload