1 Einleitung

Im Internet werden Nachrichten hauptsächlich über die Transportprotokolle TCP und UDP übertragen. TCP gewährleistet eine zuverlässige, geordnete und verlustfreie Übertragung, kann aber durch seinen aufwendigen Verbindungsaufbau, eingeschränkte Erweiterungsmöglichkeiten und begrenzte integrierte Sicherheitsmechanismen in modernen, latenz- und sicherheitskritischen Anwendungen an seine Grenzen stoßen. Das QUIC-Protokoll wurde entwickelt, um diese Limitierungen zu überwinden und moderne Anforderungen wie geringe Latenz, zuverlässige Übertragung und integrierte Verschlüsselung zu erfüllen.
Diese Ausarbeitung gibt zunächst eine Einführung in die grundlegenden Konzepte und Eigenschaften des QUIC-Protokolls. Dabei werden zentrale Mechanismen wie Verbindungsaufbau, Multiplexing und integrierte Verschlüsselung erläutert. Darauf aufbauend werden anschließend die drei in Java verwendbaren QUIC-Frameworks Kwik, Quiche4J und Netty miteinander verglichen. Der Vergleich betrachtet unter anderem den Funktionsumfang, den Abstraktionsgrad sowie die Einsatzgebiete, für die sich die jeweiligen Frameworks am besten eignen.
Im letzten Teil der Arbeit wird eine eigene Implementierung vorgestellt, die auf dem Framework Kwik basiert. Dabei werden Designentscheidungen, die konkrete Umsetzung sowie gemachte Erfahrungen und Herausforderungen beschrieben. Ziel ist es, sowohl einen praxisnahen Einblick in die Arbeit mit QUIC in Java zu geben als auch die Eignung von Kwik für reale Anwendungsfälle zu evaluieren.

2 Quic vs TCP

2.1 Paket

QUIC-Kommunikation erfolgt über UDP-Datagramme, die ein oder mehrere QUIC-Pakete zwischen Endpunkten transportieren. Jedes QUIC-Paket besteht aus einem Header und einem oder mehreren Frames, die die eigentlichen Nutzdaten und Steuerinformationen enthalten. QUIC definiert dabei verschiedene Header- und Pakettypen, die unterschiedliche Funktionen wie Verbindungshandshake oder schnellen Datentransport übernehmen.

2.1.1 Header

QUIC läuft auf UDP, was bedeutet, dass die Zuordnung zu Anwendung und Endpunkten bereits durch die IP-Adressen und Ports auf UDP-Ebene erfolgt. Da QUIC auf Anwendungsschicht arbeitet, muss es nur noch die Zuordnung auf Verbindungsebene sicherstellen. QUIC verwendet zwei verschiedene Header-Typen. Der Long Header wird für den Verbindungsaufbau verwendet, während der Short Header bei bereits etablierte Verbindungen verwendet wird.

Longheader: [1]

- **Header Form (HF):** Identifiziert den Header-Typ.
- **Fixed Bit (FB):** Zeigt an, ob das Paket gültig ist oder nicht. Ist das Bit auf 0 gesetzt, ist das Paket ungültig.
- **Long Packet Type (T):** Gibt den Typ des Long-Header-Pakets an, siehe Tabelle 1.
- **Type-Specific Bits (S):** Bits, die spezifisch für die jeweiligen Long-Header-Pakettypen sind.
- **Version ID (VID):** 32 Bit zur Identifikation der QUIC-Version.
- **Destination Connection ID Length (DCID Len):** Länge der Ziel-Connection-ID.
- **Destination Connection ID (DCID):** Ziel-Connection-ID.

- **Source Connection ID Length (SCID Len):** Länge der Quell-Connection-ID.
- **Source Connection ID (SCID):** Quell-Connection-ID.

Shortheader: [1]

- **Header Form (HF):** Identifiziert den Header-Typ.
- **Fixed Bit (FB):** Zeigt an, ob das Paket gültig ist oder nicht.
- **Spin Bit:** Wird zur Latenzmessung verwendet.
- **Reserved Bits:** Für zukünftige Erweiterungen reserviert.
- **Key Phase:** Gibt an, welcher Verschlüsselungsschlüssel für das Paket verwendet wird.
- **Packet Number Length (P):** Länge der Paketnummer.
- **Destination Connection ID (DCID):** Ziel-Connection-ID.
- **Packet Number:** Monoton steigende Paketnummer zur Verlustdetektion und ACKs.
- **Packet Payload:** Enthält die Frames des Pakets (STREAM, ACK, PADDING etc.).

Diese Struktur ermöglicht es QUIC, die Stärken von TCP, wie etwa zuverlässige Datenübertragung, zu übernehmen, während gleichzeitig Verschlüsselung, Integrität und eine flexible Zuordnung von Verbindungen auf Anwendungsebene implementiert werden können. Die Verwendung unterschiedlicher Header-Typen sorgt dafür, dass das Protokoll effizient bleibt.

2.1.2 Frame

Ein Frame enthält die Daten eines Streams, der über QUIC übertragen wird.

- **Type:** Gibt die Art des Frames an, z. B. Stream-Daten, ACK oder Control-Informationen.

- **Stream ID:** Identifiziert den Stream, zu dem die Daten gehören. Ungerade IDs stehen für vom Client initiierte Streams, gerade IDs für vom Server initiierte Streams.
- **Offset:** Gibt die Position der Daten innerhalb des Streams an, ähnlich dem TCP-Offset.
- **Data Length:** Die Länge der übertragenen Daten in Bytes.
- **Stream Data:** Die eigentlichen Nutzdaten des Streams.

2.2 Verbindungsaufbau

Beim Aufbau einer Verbindung über TCP erfolgt zunächst ein dreistufiger Handshake (SYN, SYN-ACK, ACK), um eine Verbindung aufzubauen. Dieser Handshake verursacht mindestens eine Round-Trip-Time (RTT), bevor Daten übertragen werden können. Wird zusätzlich eine Verschlüsselung wie TLS verwendet, muss über der bereits etablierten TCP-Verbindung ein weiterer Handshake stattfinden, der in der Regel ein bis zwei zusätzliche RTTs benötigt. In Szenarien mit vielen kurzen Verbindungen summieren sich diese Verzögerungen und kann die Performance erheblich beeinträchtigen. Das QUIC-Protokoll verwendet UDP als Transportbasis und baut darauf eine eigene Verbindungsschicht auf. Bereits beim ersten Verbindungsaufbau werden die Verschlüsselungsschlüssel ausgehandelt, sodass dafür in der Regel eine RTT erforderlich ist. Eine Wiederverbindung kann sogar ohne zusätzlichen Verbindungsaufbau erfolgen (0-RTT). Durch diese Integration von Transport- und Sicherheitsmechanismen reduziert QUIC die Latenz beim Verbindungsaufbau erheblich und eignet sich besonders für Anwendungen mit häufigen oder kurzlebigen Verbindungen.

2.3 Paketverlust & Fehlerbehandlung

QUIC übernimmt viele grundlegende Konzepte der Verlustbehandlung aus TCP, darunter schnelle Wiederübertragungen sowie timeoutbasierte Mechanismen zur Verlustdetektion. Durch die Trennung von Transport- und Anwendungsschicht sowie den Betrieb im Userspace erweitert QUIC diese Mechanismen jedoch um zusätzliche Informationen.

2.3.1 Packet- und Datenzuordnung

Ein zentraler Unterschied zwischen TCP und QUIC liegt in der Handhabung der Sequenznummern. Während TCP eine einzige Sequenznummer für die gesamte Übertragung verwendet, verwendet QUIC ein abgewandelteres Konzept. Jedes Paket erhält eine monoton steigende Paketnummer, die für ACKs, RTT-Messungen und Loss Detection verwendet wird. Die eigentlichen Daten werden zusätzlich durch Stream-ID und Stream-Offset identifiziert, wodurch die Reihenfolge innerhalb eines Streams eindeutig bestimmt ist. Durch diese Trennung kann QUIC jederzeit genau nachvollziehen, welche Pakete bestätigt wurden und welche verloren gingen, was die RTT-Berechnung und die Verlustbehandlung deutlich vereinfacht.

2.3.2 Ack-Zurücknehmen

QUIC erlaubt kein Zurücknehmen bereits bestätigter Pakete. Dadurch wird die Verlustbehandlung auf beiden seiten vereinfacht.

2.3.3 Ack-Ranges

Im Gegensatz zu TCP, das ACKs höchstens drei SACK-Ranges unterstützt, erlaubt QUIC mehrere ACK-Ranges in einem einzelnen ACK-Paket. Diese Flexibilität beschleunigt die Paketverlust-Erkennung und reduziert unnötige Retransmissions insbesondere in Netzen mit hohem Paketverlust.

2.3.4 Korrektur für verzögerte ACKs

QUIC-ACKs kodieren explizit die Verzögerung, die beim Empfänger zwischen dem Eintreffen eines Pakets und dem Versenden der zugehörigen Bestätigung entsteht. Dadurch kann der Empfänger des ACKs die empfangsseitige Verzögerungen in der Schätzung der RTT berücksichtigen.

2.3.5 Fehlererkennung

TCP verwendet auf Paketebene eine Prüfsumme, um sicherzustellen, dass Header und Daten auf dem Transportweg nicht verändert wurden. QUIC gewährleistet die Integrität und Manipulationssicherheit seiner Pakete hingegen über die in das Protokoll integrierte TLS-1.3-Verschlüsselung. Jedes QUIC-Paket wird dabei verschlüsselt und enthält einen Authentifizierungs-Tag, sodass Änderungen am Header oder an den enthaltenen Frames vom Empfänger zuverlässig erkannt werden. Auf diese Weise übernimmt QUIC die Funktion der klassischen TCP-Prüfsumme, erweitert diese jedoch um kryptographisch gesicherte Integrität.
Wie auch bei TCP enthält QUIC standardmäßig keine Forward Error Correction Funktionalität, aufgrund der Implementierung im Userspace ist es jedoch deutlich einfacher, solche Erweiterungen nachträglich in QUIC zu integrieren.

References

[1] Saleh Alawaji. IETF QUIC v1 design.