QUIC in Java

Kai Penazzi

January 26, 2026

HS-Offenburg

Inhaltsverzeichnis

QUIC

Was ist QUIC?

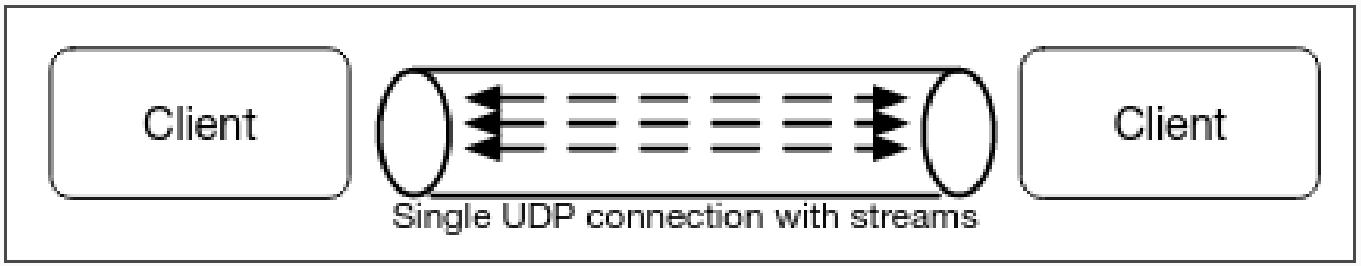
QUIC will alles können was TCP kann, nur besser

Was macht TCP aus?

- Verbindungsorientiert
- Zuverlässige Übertragung
- Geordnete Zustellung
- Flusskontrolle
- Fehlerkontrolle

Grundlagen

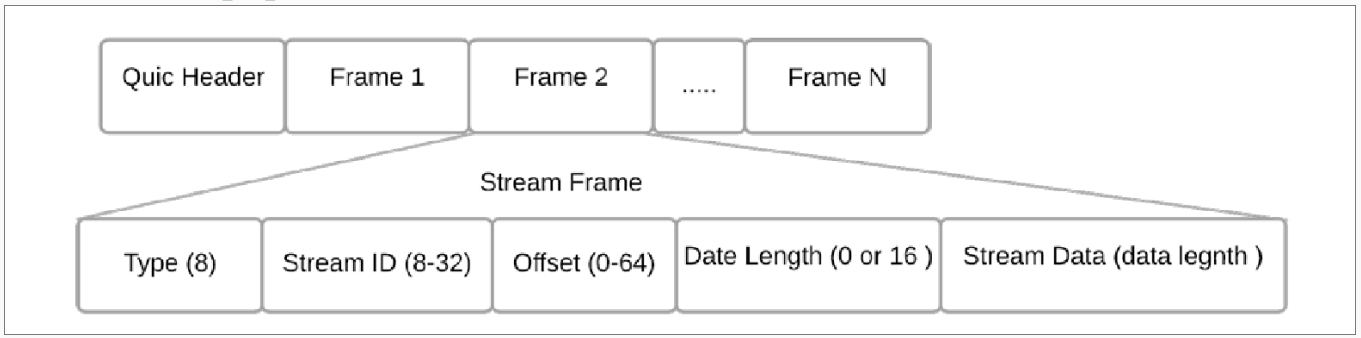
- Basiert auf UDP
- Stream Orientiert
- im Userspace



[6]

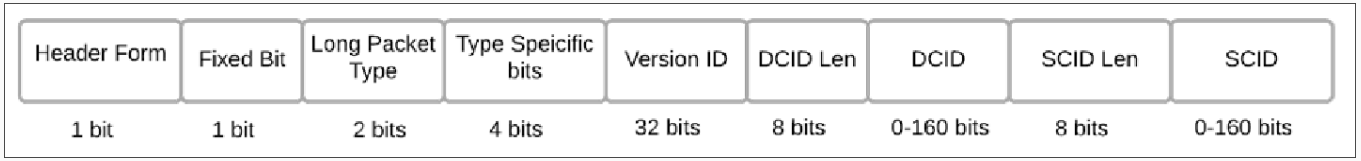
Paket

- Packet: [2]

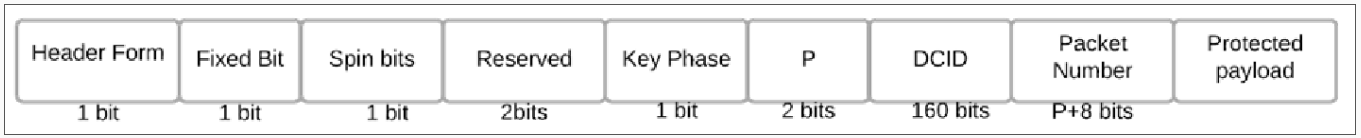


Header

- Longheader: [2]



- Shortheader: [2]



- Header typ: [2]

Type	Name	Description
0x00	Initial	Transports the first CRYPTO frames transmitted by the client and server during the key exchange and the Acknowledgment frames in both directions.
0x01	0-RTT	A 0-RTT packet sends "early" data from the client to the server before the handshake is completed.
0x02	Handshake	Packet is for sending and receiving encrypted handshake messages and acknowledgments between the server and the client.
0x03	Retry	The packet contains a server-generated address validation token. It's used by a server that wants to retry a connection.

Verbindungsaufbau

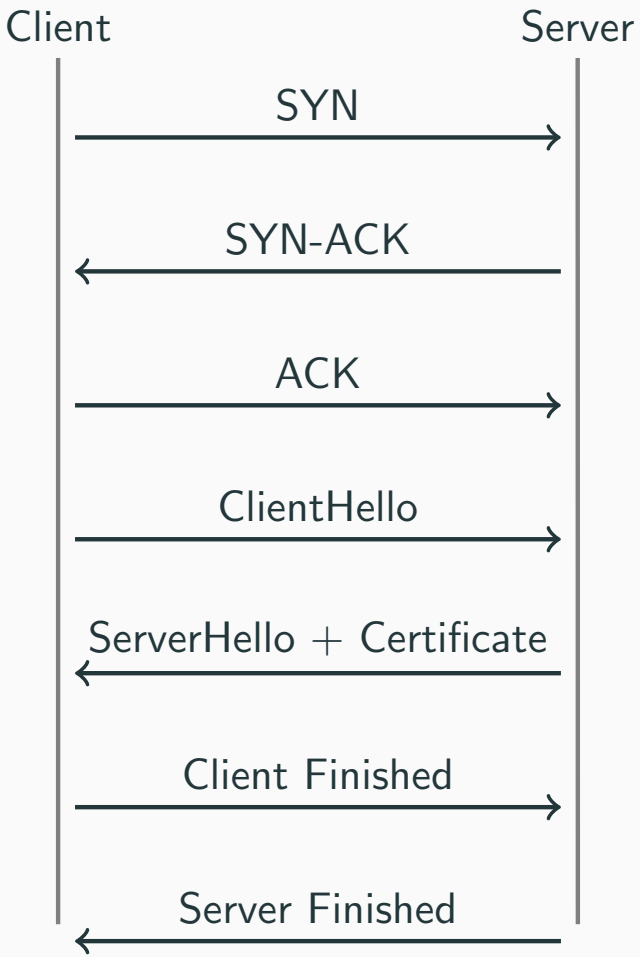


Figure 1: TCP + TLS

Figure 2: QUIC

Flusskontrolle

TCP:

- Algorithmen wie NewReno oder CUBIC
- Überwacht die Menge an unbestätigten Paketen
- Bremst das Senden neuer Pakete bei Paketverlusten

QUIC:

- Übernimmt die Grundprinzipien von TCP
- Verfolgt die Übertragung auf Ebene der Bytes-in-Flight
- Feinkörnigere Kontrolle der Daten

Packet- und Datenzuordnung

TCP:

- Pakete mit denselben Daten teilen sich die gleiche Sequenznummer
- ACKs beziehen sich auf Sequenznummern, nicht direkt auf einzelne Pakete

QUIC:

- Jedes Paket erhält eine neue Sequenznummer
- Daten werden über Stream-ID und Stream-Offset identifiziert
- Ermöglicht exakte Nachverfolgung von bestätigten und verlorenen Paketen

Ack-Zurücknehmen

- Bereits bestätigte Pakete können nicht zurückgenommen werden
- Vereinfachte Verlustbehandlung auf Sender- und Empfängerseite

Ack-Ranges

- Mehrere ACK-Ranges in einem Paket möglich
- Schnellere Paketverlust-Erkennung
- Reduziert unnötige Retransmissions, besonders bei Netzwerken mit hohem Paketverlust

Korrektur für verzögerte ACKs

- ACKs kodieren die Verzögerung zwischen Paketeingang und ACK-Versand
- Ermöglicht genauere RTT-Schätzungen beim Empfänger

Fehlererkennung

TCP: Prüfsumme auf Paketebene zur Sicherstellung von Header- und Datenintegrität

QUIC:

- Integrierte TLS-1.3-Verschlüsselung für Integrität und Manipulationsschutz
- Jedes Paket enthält einen Authentifizierungs-Tag zur Erkennung von Änderungen
- Ersetzt klassische Prüfsumme durch kryptografisch gesicherte Integrität

Forward Error Correction: Standardmäßig nicht vorhanden, aber einfacher nachrüstbar dank Userspace-Implementierung

Head-of-Line Blocking

Problem bei TCP:

- HTTP baut oft mehrere parallele TCP-Verbindungen auf
- Verzögerungen oder Paketverluste in einer Verbindung blockieren Daten in anderen Verbindungen

QUIC-Lösung:

- Mehrere unabhängige Streams über eine einzige Verbindung
- Effektive Vermeidung von Head-of-Line Blocking, schnellerer und effizienterer Datenfluss

Connection Migration

QUIC unterstützt Connection Migration:

- Verbindung bleibt bestehen, selbst wenn sich IP-Adresse des Clients ändert

Bei TCP würde die Verbindung abbrechen und neu aufgebaut werden müssen

QUIC ermöglicht dies durch:

- Übertragung über UDP
- Eindeutige Identifizierung der Verbindung durch die **Connection ID** im Header

0-RTT

- QUIC unterstützt **0-RTT**:
  - Ermöglicht, dass der Client beim Wiederaufbau einer Verbindung sofort Daten senden kann
  - Der vollständige Handshake muss nicht erneut durchlaufen werden
- Verwendung eines speziellen **0-RTT Long Headers**:
  - Identifiziert die Verbindung eindeutig über die Connection-ID

Verschlüsselung

- QUIC implementiert **TLS 1.3** direkt im Protokoll
- Daten werden verschlüsselt und vor Manipulation geschützt
- Ein zusätzlicher Verschlüsselungslayer ist nicht erforderlich

User-Space Implementierung

- QUIC wird vollständig im Userspace implementiert
- Erweiterungen, Updates oder experimentelle Features lassen sich einfacher umsetzen
- Kein Eingriff in den Kernel oder bestehende Netzwerk-Stacks notwendig

Java-Frameworks

Analyse von Java-QUIC-Frameworks

Betrachtete Open-Source-Implementierungen für Java:

- Kwik
- Quiche4J
- (Netty-QUIC)

Ziel der Analyse:

- Abstraktionsgrad der APIs vergleichen
- Integrationsaufwand bewerten
- Einsatzszenarien

Connection - Kwik

```
String applicationProtocolId = "....";
QuicClientConnection connection = QuicClientConnection.newBuilder()
    .uri(URI.create("http://127.0.0.1:7000"))
    .applicationProtocol(applicationProtocolId)
    .build();

connection.connect();
```

Stream-Management - Kwik

```
QuicStream quicStream = connection.createStream(true);
OutputStream output = quicStream.getOutputStream();
output.write(...)
output.close();
InputStream input = quicStream.getInputStream();
input.read(...)
```

Connection - Quiche4j

```
final Config config = new ConfigBuilder(Quiche.PROTOCOL_VERSION).build();
final byte[] connId = Quiche.newConnectionId();
final Connection conn = Quiche.connect("127.0.0.1:7000", connId, config);
```

Receive-Loop - Quiche4j

```
final byte[] buf = new byte[1350];
while(true) {
    DatagramPacket packet = new DatagramPacket(buf, buf.length);
    socket.receive(packet);
    final byte[] buffer = Arrays.copyOfRange(
    packet.getData(), packet.getOffset(), packet.getLength());
    final int read = conn.recv(buffer);
    if(read <= 0) break;
}
```

Send-Loop - Quiche4j

```
final byte[] buf = new byte[1350];
while(true) {
    DatagramPacket packet = new DatagramPacket(buf, buf.length);
    socket.receive(packet);
    final byte[] buffer = Arrays.copyOfRange(
    packet.getData(), packet.getOffset(), packet.getLength());
    final int read = conn.recv(buffer);
    if(read <= 0) break;
}
```

Write and Read - Quiche4j

```
if(conn.isEstablished()) {
    conn.streamSend(0, "hello".getBytes(), true);
}

if(conn.isEstablished()) {
    final byte[] buf = new byte[1350];
        for(long streamId : conn.readable()) {
        while(true) {
        final int len = conn.streamRecv(streamId, buf);
        if(len <= 0) break;
            // do stuff
        }
    }
}
```

QuicFactory - Kwik

```
public class MyQuicFactory implements ApplicationProtocolConnectionFactory {
    @Override
        public ApplicationProtocolConnection createConnection(
            String protocol,
            QuicConnection quicConnection) {
        return new MyQuicConnection(quicConnection);
    }
}
```

ApplicationProtocolConnection - Kwik

```
public class MyQuicConnection implements ApplicationProtocolConnection {
    private final QuicConnection quicConnection;

    public MyQuicConnection(QuicConnection quicConnection) {
        this.quicConnection = quicConnection;
    }

    @Override
    public void acceptPeerInitiatedStream(QuicStream stream) {
        // do stuff with stream
    }
}
```

Start - Kwik

```
KeyStore keystore = KeyStore.getInstance("PKCS12");
try (FileInputStream keystoreStream = new FileInputStream(new File(KEYSTORE_PAT
    .getAbsolutePath())) {
    keystore.load(keystoreStream, KEYSTORE_PASSWORD.toCharArray());
}
Logger log = new SysOutLogger();
List<QuicConnection.QuicVersion> supportedVersions = new ArrayList<>();
supportedVersions.add(QuicConnection.QuicVersion.V1);
ServerConnectionConfig config = ServerConnectionConfig.builder()
    .build();
ServerConnector connector = ServerConnector.builder()
    .withPort(7000)
    .withSupportedVersions(supportedVersions)
    .withKeyStore(keystore, KEY_ALIAS, KEYSTORE_PASSWORD.toCharArray())
    .withConfiguration(config)
    .withLogger(log)
    .build();
connector.registerApplicationProtocol("quic", new MyQuicFactory());
connector.start();
```

Connection - Quiche4j

```
final Config config = new ConfigBuilder(Quiche.PROTOCOL_VERSION).build();
final byte[] connId = Quiche.newConnectionId();
final Connection conn = Quiche.accept(sourceConnId, originalDestinationId, conf
```

State- und Event-Transparenz

Typische Metriken in den Statistiken:

- Anzahl empfangener Pakete
- Anzahl gesendeter Pakete
- Anzahl verlorener Pakete
- Geschätzte Round-Trip-Time (RTT)
- Aktuelles Congestion Window
- Geschätzte Übertragungsrate

Fazit

Vergleich

Kwik:

- High-Level-API: Handshake, Socket-Management, Stream-Verwaltung automatisch
- Ideal für schnelle Implementierungen ohne tiefe Netzwerkkenntnisse

Quiche / Quiche4J:

- Nur die QUIC-State-Machine wird bereitgestellt
- Entwickler muss UDP-Socket und Eventloop selbst implementieren
- Maximale Flexibilität, aber hoher Implementierungsaufwand

QUIC

- TCP-Grundprinzipien werden übernommen
- Aufbau auf UDP:
  - Keine Anpassungen an Router oder Endgeräte nötig
  - Funktioniert ohne Kernel-/Hardwareänderungen
- User-Space-Implementierung:
  - Schnellere Entwicklung und Rollout neuer Protokollfunktionen
- Features:
  - Stream-Multiplexing
  - Integrierte Verschlüsselung


netty/netty-incubator-codec-quic.
original-date: 2020-11-06T15:51:08Z.


Saleh Alawaji.
IETF QUIC v1 design.


Peter Doornbosch.
ptrd/kwik.
original-date: 2019-12-18T20:46:54Z.


Jana Iyengar and Ian Swett.
QUIC loss detection and congestion control.
Num Pages: 30.


Oleksii Kachaiev.
kachayev/quiche4j.
original-date: 2020-09-05T18:49:34Z.


Puneet Kumar.

QUIC (quick UDP internet connections) – a quick study.