Transmit(TX) und Receive(RX) Programm in Python and Java

Bacher Gerwin, Glavas Jure, Jovanovic Ivan, Tomandl Tomas

Transmitter TX - Python

Transmitter TX Python

```python
#!/usr/bin/env python3
import socket, sys, math, hashlib, os

def main():  1 usage
    if len(sys.argv) != 5:
        print("Verwendung: python3 Transmitter_TX.py <EmpfängerIP> <ID> <DatenBytes> <MaxBytesProPaket>")
        sys.exit(1)

    ip, tx_id, size, pkt_size = (
        sys.argv[1],
        int(sys.argv[2]),
        int(sys.argv[3]),
        int(sys.argv[4]),
    )
```

socket	UDP-Kommunikation
sys	Zugriff auf Kommandozeilenparameter
math	Aufrunden
hashlib	MD5-Prüfsummen
os	Erzeugung von Daten

Transmitter TX Python

```
data = os.urandom(size)
checksum = hashlib.md5(data).digest()

total_packets = math.ceil(size / pkt_size) + 2

                          IPV-4          Datagram
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

filename = f"gen_{size}B.bin".encode('utf-8')
header = (
        tx_id.to_bytes( length: 2,  byteorder: 'big') +
        (0).to_bytes( length: 4,  byteorder: 'big') +
        total_packets.to_bytes( length: 4,  byteorder: 'big') +
        len(filename).to_bytes( length: 4,  byteorder: 'big') +
        filename
)
sock.sendto(header, (ip, 5005))
```

Datagram = UDP
Socket versendet und empfängt kurze, eigenständige Nachrichten (Datagramme) ohne vorherige Verbindung

Transmitter TX Python

```
offset = 0
for seq in range(1, total_packets - 1):
    chunk = data[offset:offset + pkt_size]
    packet = (
            tx_id.to_bytes( length: 2,  byteorder: 'big') +
            seq.to_bytes( length: 4,  byteorder: 'big') +
            chunk
    )
    sock.sendto(packet, (ip, 5005))
    offset += pkt_size
```

Transmitter TX Python

```
    footer = (
            tx_id.to_bytes( length: 2,  byteorder: 'big') +
            (total_packets - 1).to_bytes( length: 4,  byteorder: 'big') +
            checksum
    )
    sock.sendto(footer, (ip, 5005))

    sock.close()

if __name__ == "__main__":
    main()
```

Receiver RX - Python

Netzwerk-Socket einrichten

```
sock = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
sock.bind(('', PORT))
packets = {}
expected_txid = None
max_seq = None
filename = None
expected_md5 = None
```

Ein UDP-Socket wird geöffnet und an Port 5005 gebunden.
Ein Dictionary speichert Pakete.
Kontrollvariablen helfen bei der Strukturierung des Empfangs.

Pakete empfangen

```
data, addr =
sock.recvfrom(BUFFER_SIZE)
txid = int.from_bytes(data[0:2],
'big')
seq  = int.from_bytes(data[2:6],
'big')
```

Jedes Paket enthält eine Transaktions-ID und eine Sequenznummer.
Diese werden zum Sortieren und Überprüfen verwendet.

Erstes Paket verarbeiten

```
if seq == 0:
    expected_txid = txid
    max_seq =
int.from_bytes(data[6:10], 'big')
    name_len =
int.from_bytes(data[10:14], 'big')
    filename =
data[14:14+name_len].decode()
```

Das erste Paket liefert Metadaten wie maximale Sequenzzahl und Dateinamen. Diese dienen zur Vorbereitung auf den Dateiaufbau.

Datenpakete speichern

```
elif txid == expected_txid:
    packets[seq] = data[6:]
```

Normale Datenpakete werden anhand der Sequenznummer gespeichert.
Nur gültige Pakete mit richtiger Transaktions-ID werden verwendet.

Letztes Paket erkennen

```
elif seq == max_seq - 1:
    expected_md5 = data[6:22]
```

Das letzte Paket enthält den MD5-Hash. Dieser wird später zur Verifikation der Datei benutzt.

Datei schreiben & MD5 prüfen

```
with open(f"recv_{filename}", 'wb')
as f:
    for s in range(1, max_seq-1):
        chunk = packets[s]
        f.write(chunk)
        hasher.update(chunk)
```

Die Datei wird aus den gespeicherten Paketen zusammengesetzt.
Parallel wird ein MD5-Hash berechnet.

MD5 vergleichen

```
if comp == expected_md5:
    print("File received correctly
(MD5 ok).")
else:
    print("MD5 mismatch –
Übertragung fehlerhaft.")
```

Der berechnete Hash wird mit dem empfangenen verglichen.
Nur bei Übereinstimmung ist die Datei korrekt übertragen worden.

Transmitter TX - Java

Transmitter TX Java

```java
byte[] daten = new byte[datenGroesse];
new SecureRandom().nextBytes(daten);
MessageDigest md5 = MessageDigest.getInstance( algorithm: "MD5");
byte[] md5sum = md5.digest(daten);
```

Es werden zufällige Daten erzeugt und eine MD5-Prüfsumme berechnet. Diese Prüfsumme wird später zur Integritätsprüfung genutzt.

Transmitter TX Java

```java
String dateiname = "gen_" + datenGroesse + "B.bin";
byte[] nameBytes = dateiname.getBytes( charsetName: "UTF-8");
// 2 Bytes SendungsID, 4 Bytes Seq, 4 Bytes maxSeq, 4 Bytes NameLänge, NameBytes
int headerLen = 2 + 4 + 4 + 4 + nameBytes.length;
byte[] header = new byte[headerLen];
int pos = 0;
header[pos++] = (byte)(sendungsId >> 8);
header[pos++] = (byte)(sendungsId);
// Seq = 0
for (int i = 3; i >= 0; i--) header[pos++] = (byte)((0 >> (8*i)) & 0xFF);
// maxSeq
for (int i = 3; i >= 0; i--) header[pos++] = (byte)((maxSeq >> (8*i)) & 0xFF);
// Name-Länge
for (int i = 3; i >= 0; i--) header[pos++] = (byte)((nameBytes.length >> (8*i)) & 0xFF);
// Name-Bytes
System.arraycopy(nameBytes, srcPos: 0, header, pos, nameBytes.length);

socket.send(new DatagramPacket(header, header.length, empfaenger, PORT));
```

Das erste Paket enthält Metadaten: die Sendungs-ID, maximale Paketanzahl, die Sequenznummer 0 und den Dateinamen. Es dient dem Empfänger als Initialisierung.

Transmitter TX Java

```java
if (args.length != 4) {
    System.err.println("Verwendung: java Transmitter_TX <EmpfängerIP> <SendungsID> <DatenGroesseBytes> <MaxDatenBytesProPaket>");
    System.exit( status: 1);
}
InetAddress empfaenger = InetAddress.getByName(args[0]);
int sendungsId    = Integer.parseInt(args[1]);
int datenGroesse  = Integer.parseInt(args[2]);
int paketGroesse  = Integer.parseInt(args[3]);
```

Der Sender erwartet vier Parameter: IP-Adresse des Empfängers, eine Sendungs-ID, die Datengröße und die maximale Größe pro Datenpaket.

Transmitter TX Java

```java
int offset = 0;
for (int seq = 1; seq < maxSeq - 1; seq++) {
    int len = Math.min(paketGroesse, datenGroesse - offset);
    byte[] pkt = new byte[2 + 4 + len];
    pos = 0;
    pkt[pos++] = (byte)(sendungsId >> 8);
    pkt[pos++] = (byte)(sendungsId);
    // Seq
    for (int i = 3; i >= 0; i--) pkt[pos++] = (byte)((seq >> (8*i)) & 0xFF);
    // Daten
    System.arraycopy(daten, offset, pkt, pos, len);
    socket.send(new DatagramPacket(pkt, pkt.length, empfaenger, PORT));
    System.out.printf("[TX] Datenpaket %d gesendet (%d B)%n", seq, len);
    offset += len;
}
```

Reassigned local variable

Alle Daten werden in Blöcke aufgeteilt und mit einer fortlaufenden Sequenznummer versehen.

Transmitter TX Java

```
int lastSeq = maxSeq - 1;
byte[] footer = new byte[2 + 4 + md5sum.length];
pos = 0;
footer[pos++] = (byte)(sendungsId >> 8);
footer[pos++] = (byte)(sendungsId);
for (int i = 3; i >= 0; i--) footer[pos++] = (byte)((lastSeq >> (8*i)) & 0xFF);
System.arraycopy(md5sum, srcPos: 0, footer, pos, md5sum.length);

socket.send(new DatagramPacket(footer, footer.length, empfaenger, PORT));
System.out.println("[TX] Letztes Paket (MD5) gesendet.");

socket.close();
```

Das letzte Paket enthält den MD5-Hash. Der Empfänger kann damit überprüfen, ob die Datei korrekt und vollständig empfangen wurde.

Receiver RX - Java

Netzwerk-Socket einrichten

```
DatagramSocket socket = new DatagramSocket(PORT);
Map<Integer, byte[]> speicher = new HashMap<>();
Integer erwarteteId = null, maxSeq = null;
String dateiname = null;
byte[] erwartetesMd5 = null;
```

-Wir starten den Receiver, indem wir einen UDP-Socket öffnen und eine HashMap zum Speichern der Pakete erstellen.

- Die Variablen `erwarteteId`, `maxSeq` und `erwartetesMd5` dienen zur Paket- und Integritätskontrolle.

Pakete empfangen und identifizieren

```
byte[] buf = new byte[PUF_SIZE];
DatagramPacket pkt = new DatagramPacket(buf, buf.length);
socket.receive(pkt);
byte[] data = pkt.getData();
int len = pkt.getLength();
```

- Der Receiver empfängt ein Paket und extrahiert den Dateninhalt. Die Länge des Pakets wird ebenfalls gespeichert.

Verarbeitung des ersten Pakets (Header)

```
// TX_ID
int txId = ((data[0]&0xFF)<<8) | (data[1]&0xFF);
// Seq
int seq  = ((data[2]&0xFF)<<24)|((data[3]&0xFF)<<16)|((data[4]&0xFF)<<8)|(data[5]&0xFF);
```

-Die ersten sechs Bytes des Pakets enthalten die Sende-ID und die Sequenznummer.
-Der Empfänger identifiziert den Sendungs-ID und den Typ des Pakets.

Header-Paket erkennen

```
// Erstes Paket
if (seq == 0) {
    erwarteteId = txId;
    maxSeq = ((data[6]&0xFF)<<24)|((data[7]&0xFF)<<16)|((data[8]&0xFF)<<8)|(data[9]&0xFF);
    int nameLen = ((data[10]&0xFF)<<24)|((data[11]&0xFF)<<16)|((data[12]&0xFF)<<8)|(data[13]&0xFF);
    dateiname = new String(data, 14, nameLen, "UTF-8");
    System.out.printf("[RX] Empfang '%s', erwarte %d Pakete%n", dateiname, maxSeq);
}
```

-Das erste Paket (Seq = 0) enthält die Dateiinformationen.
-Der Receiver extrahiert die maximale Paketanzahl und den Dateinamen.

Empfang von Datenpaketen

```
else if (txId == erwarteteId) {
    int payloadLen = len - 6;
    byte[] chunk = new byte[payloadLen];
    System.arraycopy(data, 6, chunk, 0, payloadLen);
    speicher.put(seq, chunk);
    System.out.printf("[RX] Paket %d (%d B) zwischengespeichert%n", seq, payloadLen);
}
```

Jedes Datenpaket wird im Speicher (HashMap) abgelegt, basierend auf seiner Sequenznummer.

Empfang des letzten Pakets (Footer)

```
else if (maxSeq != null && seq == maxSeq - 1) {
    erwartetesMd5 = new byte[16];
    System.arraycopy(data, 6, erwartetesMd5, 0, 16);
    System.out.println("[RX] Letztes Paket empfangen.");
}
```

Das letzte Paket enthält den MD5-Hash der Datei. Dies ist entscheidend für die Integritätsprüfung.

Datei schreiben & MD5 prüfen

```
try (FileOutputStream fos = new FileOutputStream("recv_" + dateiname)) {
    for (int s = 1; s < maxSeq - 1; s++) {
        byte[] chunk = speicher.get(s);
        fos.write(chunk);
        md5.update(chunk);
    }
}
byte[] actual = md5.digest();
```

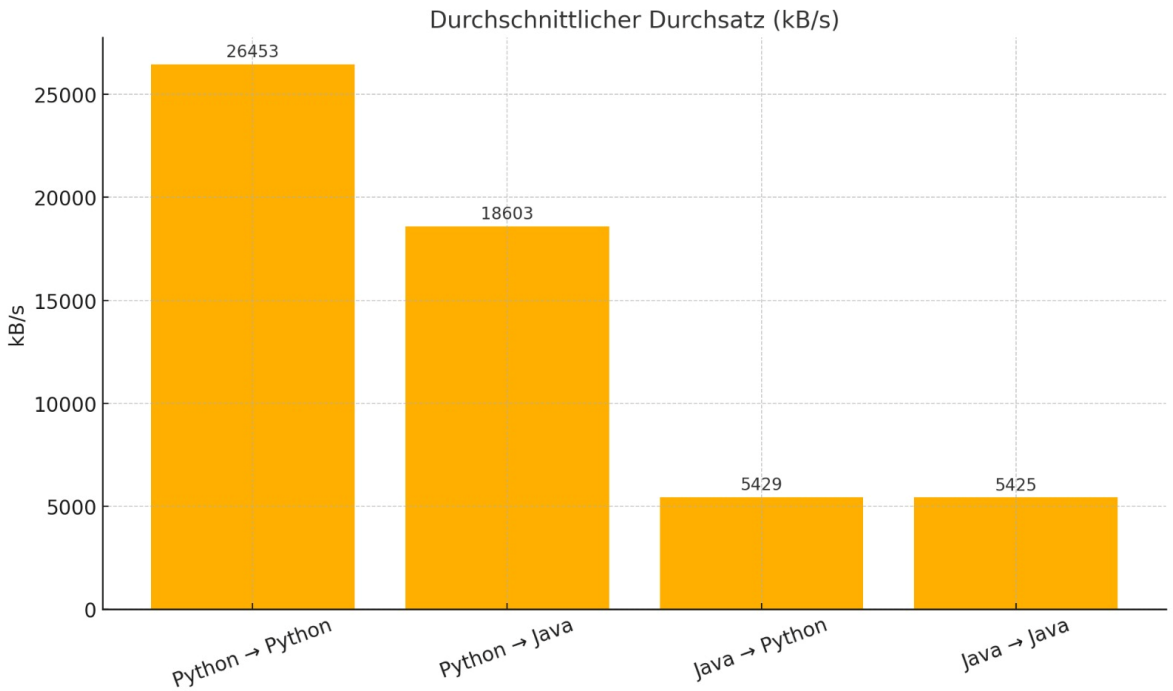
Alle gespeicherten Pakete werden in der richtigen Reihenfolge geschrieben und der MD5-Hash wird parallel berechnet.

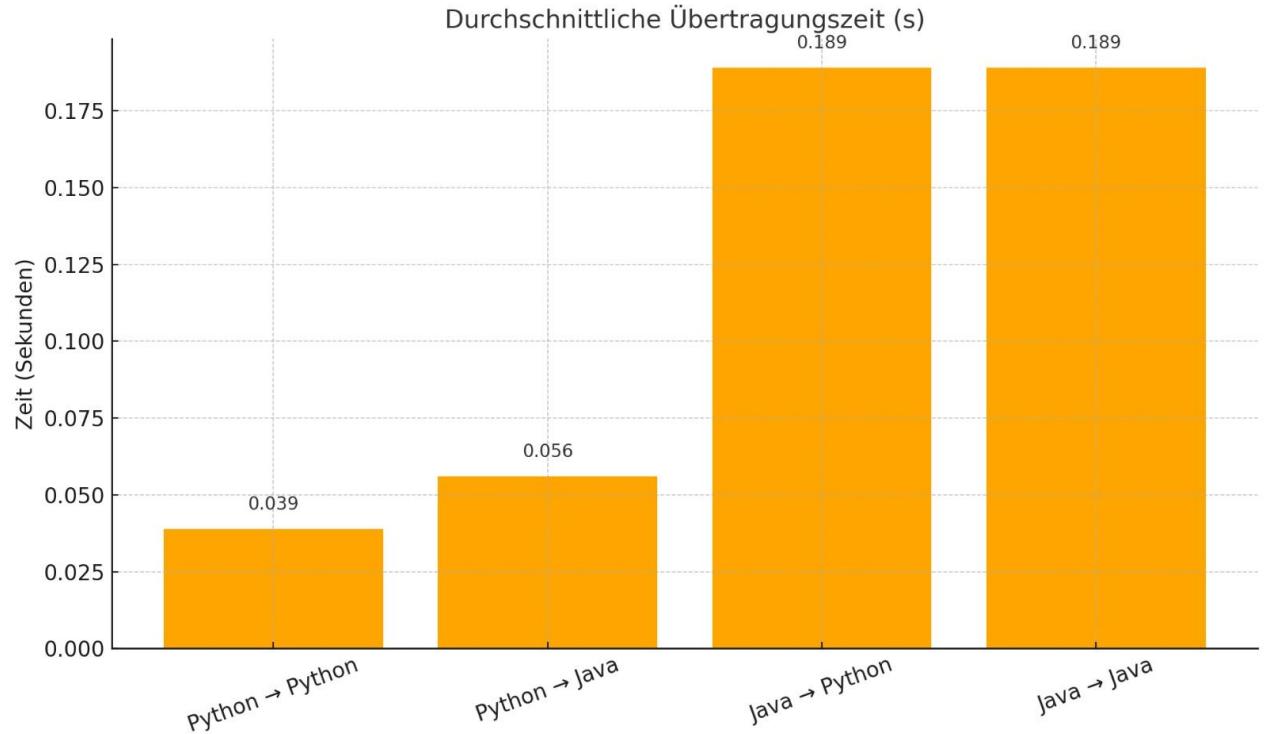
MD5 Integritätsprüfung

```
byte[] actual = md5.digest();
if (MessageDigest.isEqual(actual, erwartetesMd5)) {
    System.out.println("[RX] Datei korrekt empfangen (MD5 OK).");
} else {
    System.out.println("[RX] MD5 mismatch – Übertragung fehlerhaft.");
}
break;
```

Der berechnete MD5-Hash wird mit dem empfangenen verglichen. Bei Übereinstimmung ist die Datei fehlerfrei.

Ergebnisse


Durchschnittlicher Durchsatz (kB/s)
kB/s
0
5000
10000
15000
20000
25000
26453
18603
5429
5425
Python → Python
Python → Java
Java → Python
Java → Java


Durchschnittliche Übertragungszeit (s)
Zeit (Sekunden)
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.039
0.056
0.189
0.189
Python → Python
Python → Java
Java → Python
Java → Java

Analyse der Messwerte

Die beiden Diagramme zeigen die durchschnittliche **Übertragungszeit (in Sekunden)** sowie den **Datendurchsatz (in kB/s)** für vier verschiedene Kommunikationskombinationen zwischen Python und Java.

- **Python → Python** ist in beiden Kategorien am effizientesten: schnellste Übertragung mit 0.039 s und höchster Durchsatz mit 26.453 kB/s.
- Sobald Java beteiligt ist, insbesondere als **Sender**, verschlechtern sich die Werte deutlich.
- Bei **Java → Python** und **Java → Java** liegt der Durchsatz nur bei ca. 5.4 kB/s – das ist etwa **80 % niedriger** als bei Python → Python.
- Auch die Übertragungszeit steigt dort auf **0.189 s** – fast das Fünffache.
- Die Java-Implementierung scheint daher ein Performance-Flaschenhals zu sein.
- **Fazit:** Für maximale Effizienz sollte möglichst Python als Sender verwendet werden. Die Java-Implementierung sollte optimiert werden, um vergleichbare Leistung zu erzielen.