Wir entwickeln und evaluieren in JAVA eine Client-/Serveranwendung **„Reliable File Transfer“ (RFT)** zum Kopieren von Dateien von einem Client auf einen Server, die UDP verwendet und Reihenfolgeerhaltung sowie zuverlässigen Datentransfer gemäß vereinfachter TCP-Spezifikation (wie in der Vorlesung angegeben) selbst implementiert.
Der vorgegebene Java-Code ist dafür zu ergänzen, testen und anzuwenden.

RFT - Client
Der RFT-Client erhält vom Benutzer folgende Parameter als Argumente übergeben:

1. Hostname des Servers [String]
2. Quellpfad (inkl. Dateiname) der zu sendenden lokalen Datei [String]
3. Zielpfad (inkl. Dateiname) der zu empfangenden Datei (falls bereits vorhanden, wird die Datei überschrieben) [String]
4. Window Size (Sendepuffergröße = Empfangspuffergröße) in Anzahl Bytes [long]
5. Fehlerrate n (jedes n-te Paket ist fehlerhaft) zur Auswertung für den Server [long]
6. Fast Retransmit-Mode [true/false]
7. Test Output-Mode (Testausgaben ein-/ausschalten) [true/false]

Anschließend überträgt das Clientprogramm die Datei (Quellpfad) an den RFT-Server (dort Speicherung unter Zielpfad) mittels einer Folge von UDP-Paketen (Serverport: 53480), wobei das **erste** UDP-Paket (Sequenznr. -1) als Steuer-Paket keine Dateidaten, sonden Parameter für den Server (Zieldateipfad, Window Size (Empfangspuffergröße), Fehlerrate, TestOutput-Mode) enthält.

Der Client muss dafür sorgen, dass er verlorengegangene Pakete entdeckt und neu überträgt (wie beim vereinfachtem **TCP**-Sender). Zur dynamischen Berechnung des **Timeoutintervalls** soll ebenfalls der Algorithmus des TCP-Verfahrens verwendet werden.

FSM-Spezifikation des RFT-Clients („Sender“):

Datenstrukturen:

- Sendepuffer in der angegebenen Größe N Bytes (N: Window-Size) zur Speicherung aller gesendeten, aber unbestätigten Pakete
- sendbase (Sequenznummer des ältesten Pakets im Sendepuffer) [Initialisierung: 0]
- nextSeqNum (Sequenznummer des nächsten zu sendenden Pakets) [Initialisierung: 0]

Ereignisse und Aktionen:
Das nächste Paket der zu übertragenden Datei ist gelesen:

- Lege das Paket im Sendepuffer ab, falls es noch in den Puffer passt
  (die Anzahl unbestätigter Bytes darf nicht größer als N sein), sonst solange blockieren
- Sende das Paket mit Sequenznummer nextSeqNum
- Speichere den Sendezeitpunkt (Timestamp) für das Paket (zur RTT-Bestimmung)
- Erhöhe nextSeqNum um die Anzahl der Datenbytes des Paketes
- Falls der Timer nicht läuft, starte Timer(Timeoutintervall)

Timeout:

- Sende Paket mit Sequenznummer sendbase erneut (ohne Timestamp)
- Starte Timer (Timeoutintervall) neu

ACK empfangen und ACK > sendbase

- Timeoutintervall neu berechnen (mit aktueller RTT als sampleRTT)
- Setze sendbase = ACK

- Lösche alle Pakete aus dem Sendepuffer mit Sequenznr. < sendbase
  (ggf. Sendethread aufwecken)
- Wenn der Sendepuffer NICHT leer ist
  - dann starte Timer für Paket sendbase neu
  - sonst: Timer stoppen („canceln“)

Bei Fast Retransmit-Mode: ACK empfangen und ACK == sendbase

- Wenn 3. Wiederholung desselben ACKs → Timeout-Aktionen

RFT – Server

Der Server empfängt UDP-Pakete auf Port 53480. Er ist nicht multi-threadingfähig (es kann nur ein Kopierauftrag zur Zeit verarbeitet werden). Ein neuer Kopierauftrag wird beim Eintreffen des ersten UDP-Pakets gestartet, wobei der Client durch IP-Adresse und Quellport eindeutig festgelegt ist (logische „Verbindung“: evtl. eintreffende UDP-Pakete von anderen Absendern werden ignoriert, solange der Kopierauftrag noch nicht beendet ist). Wenn sich der Client für 3 Sekunden nicht gemeldet hat, erklärt der Server den Kopierauftrag (und die Speicherung der Dateidaten) für beendet und steht für einen neuen Auftrag zur Verfügung.
Der Server arbeitet ebenfalls nach dem vereinfachten TCP-Verfahren als Empfänger und versendet entsprechende Pakete, die jeweils nur eine Sequenznummer enthalten, als positive Quittung (ACK).
Der Server ist in der Lage, zum Testen jeweils das n-te empfangene Pakete als fehlerhaft zu melden (Parameter Fehlerrate = n). Die Fehlerwahrscheinlichkeit für ein Paket ist also 1/n.

FSM-Spezifikation des RFT-Servers („Empfänger“):

Datenstrukturen:

- Empfangspuffer zur Speicherung aller korrekt empfangenen Pakete, die außerhalb der Reihenfolge ankommen
- rcvbase: die Sequenznummer des nächsten Pakets in Reihenfolge, auf dessen Empfang gewartet wird [Initialisierung: 0]
- rcvWindow: die Anzahl der freien Bytes im Empfangspuffer

Ereignisse und Aktionen:

Paket mit Sequenznummer seqNum korrekt empfangen

- Wenn seqNum > rcvbase
  - Speichere Paket im Empfangspuffer, falls nicht voll
    (rcvWindow >= Anzahl Paket-Datenbytes)
- Wenn seqNum == rcvbase
  - Schreibe Paket in die Zieldatei (Auslieferung in Reihenfolge)
  - Setze rcvbase = seqNum + Anzahl Paket-Datenbytes
  - Schreibe - beginnend bei rcvbase - alle Pakete, die sich in lückenlos aufsteigender Reihenfolge im Empfangspuffer befinden, in die Zieldatei, lösche sie aus dem Empfangspuffer und setze
    rcvbase = Sequenznummer des letzten geschriebenen Pakets + Anzahl Datenbytes
- Sende ACK(rcvbase)

Sonst:

- Sende ACK(rcvbase)

1. Aufgabe: Ergänzung der Implementierung

Ergänzen Sie den mitgelieferten Java-Code in den Dateien **`RFTClient.java`** und **`RFTClientRcvThread.java`** (Kommentar: /* ToDo */) gemäß der o.g. RFT-Client-Spezifikation!

Hinweis:
Berechnen Sie die aktuelle Timeoutzeit gemäß TCP-Algorithmus (siehe Vorlesung)! Bedenken Sie, dass für Pakete, die nach einem Timerablauf neu gesendet wurden, keine RTT-Messung stattfindet. Die estimatedRTT sollten Sie bei der ersten Berechnung mit 4 * sampleRTT initialisieren.

2. Aufgabe: Test

Sie können RFT-Client und RFT-Server auf demselben Rechner starten (localhost - Schnittstelle) oder auf zwei verschiedenen Rechnern.
Verwenden Sie die vorgegebene Paketgröße von 1.000 Byte Nutzdaten pro Paket.

Testen Sie Ihre Anwendung mit der Datei **`RFTDataSmall.jpg`** (102 Pakete), TestOutputMode = true und einer Window Size (Sende-/Empfangspuffergröße) von 10.000 Byte (entspricht 10 Paketen)

1. Fehlerrate von 1000 (d.h. keine Fehler), FastRetransmit = false
   → es sollten keine Timeouts auftreten
2. Fehlerrate von 10 (jedes 10. Paket ist fehlerhaft), FastRetransmit = false
   → es sollten 11 Timeouts auftreten
3. Fehlerrate von 10 (jedes 10. Paket ist fehlerhaft), FastRetransmit = true
   → es sollten keine Timeouts auftreten

Dokumentieren Sie Ihre Testergebnisse!

1.000 Window: 1200ms
2.000 Window: 901ms
2.500 Window: 847ms
3.000 Window: 818ms
3.500 Window: 875ms
4.000 Window: 851ms
8.000 Window: 858ms
16.000 Window: 859ms

3. Aufgabe: Experimentelle Auswertung

Verwenden Sie die vorgegebene Paketgröße von 1.000 Byte Nutzdaten pro Paket.
Übertragen Sie bei den folgenden Experimenten die Datei **`RFTDataLarge.mp4`** (6.537 Pakete) mit TestOutputMode = false!

1. Ermitteln Sie die bzgl. der Gesamt-Übertragungszeit optimale Window Size, wenn keine Fehler auftreten (Fehlerate 100.000). Starten Sie dazu bei einer Window Size von 1.000 Byte (1 Paket) und erhöhen Sie diese systematisch, bis keine signifikanten Änderungen in der Gesamt-Übertragungszeit mehr eintreten.
   Begründen Sie das beobachtete Verhalten!

2. Multiplizieren Sie die ermittelte optimale Window Size mit 100 und führen Sie die Übertragung erneut aus. Begründen Sie das beobachtete Verhalten!

   205ms ??

3. Verwenden Sie die optimale Window Size und führen Sie Übertragungen mit den Fehlerraten 5, 10, 20, 50 durch, und zwar jeweils mit und ohne FastRetransmit-Mode!
   Begründen Sie das beobachtete Verhalten!

Dokumentieren Sie Ihre Experimente!

Tipp: Sie können die Testausgaben (auf stderr) auch in eine Datei umlenken, wenn Sie z.B. **`2> trace.txt`** beim Aufruf über eine Shell anhängen. Bedenken Sie aber, dass mit den Testausgaben die Laufzeitmessung stark beeinflusst wird!

Viel Spaß!

1.000 Window: 464ms
2.000 Window: 288ms
3.000 Window: 235ms
4.000 Window: 199ms
5.000 Window: 219ms
16.000 Window: