BAI4-RN	Praktikum Rechnernetze	HBN/SLZ
WS 13	Aufgabe 3 – Zuverlässiger Datentransfer	2.12./9.12.

Wir entwickeln in JAVA eine Client-/Serveranwendung "**FileCopy**" zum Kopieren von Dateien von einem Client auf einen Server, die UDP verwendet und Reihenfolgeerhaltung sowie zuverlässigen Datentransfer gemäß „**Selective-Repeat**"-Verfahren (wie in der Vorlesung angegeben) selbst implementiert. Der Server-Code und verschiedene Hilfsklassen sind dabei vorgegeben.

"FileCopy" - Client

Der FileCopyClient 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-Größe N [int]
5. Fehlerrate (ERROR_RATE) zur Auswertung für den Server [long]

Anschließend überträgt das Clientprogramm die Datei (Quellpfad) an den FileCopy-Server (dort Speicherung unter Zielpfad) mittels einer Folge von UDP-Paketen, wobei das **erste** UDP-Paket (Sequenznr. 0) als Steuer-Paket keine Dateidaten, sonden Parameter für den Server in der folgenden Syntax enthält:

```
0                    [8 Byte Sequenznummer]
Zieldateipfad        [String beliebiger Länge in UTF-8 – Codierung]
;                    [Trennzeichen]
Window-Größe N       [int]
;                    [Trennzeichen]
Fehlerrate ERROR_RATE [long]
```

Der Client muss dafür sorgen, dass er verlorengegangene Pakete entdeckt und neu überträgt (nach **Selective-Repeat**-Verfahren). Lediglich zur dynamischen Berechnung der **Timeoutzeit** soll der Algorithmus des TCP-Verfahrens verwendet werden (siehe Vorlesung).

FSM-Spezifikation des Clients („Sender"):

Datenstrukturen:

- Sendepuffer mit N Plätzen (N: Window-Größe)
- sendbase (Sequenznummer des ältesten Pakets im Sendepuffer)
- nextSeqNum (Sequenznummer des nächsten zu sendenden Pakets)

Ereignisse und Aktionen:

Das nächste Paket der zu übertragenden Datei ist gelesen:

- Lege das Paket im Sendepuffer ab, falls der Puffer nicht voll ist
- Sende das Paket mit Sequenznummer nextSeqNum
- Starte Timer für das Paket
- Erhöhe nextSeqNum

Timeout von Timer für Paket mit Sequenznummer n:

- Wiederhole das Senden von Paket n
- Timer für Paket n neu starten

ACK(n) empfangen und Paket n ist im Sendepuffer

- Markiere Paket n als quittiert

- Timer für Paket n stoppen
- Timeoutwert mit gemessener RTT für Paket n neu berechnen
- Wenn n = sendbase, dann lösche ab n alle Pakete, bis ein noch nicht quittiertes Paket im Sendepuffer erreicht ist, und setze sendbase auf dessen Sequenznummer

"FileCopy" – Server (vorgegeben)
Der Server ist nicht multi-threadfähig (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 Selective-Repeat-Verfahren als Empfänger und versendet entsprechende Pakete, die jeweils nur eine Sequenznummer enthalten, als positive Quittung (ACK).
Jedes ACK erhält allerdings eine zusätzliche Verzögerung von 10 ms (→ Simulation einer hohen Ausbreitungsverzögerung).
Der Server ist in der Lage, zum Testen jeweils das n-te empfangene Pakete als fehlerhaft zu melden (Parameter ERROR_RATE = n).
Die oben beschriebenen Parameter(Zieldateipfad, Window-Größe N und Fehlerrate) werden vom Client mit dem ersten Paket (Sequenznr. 0) übermittelt.

FSM-Spezifikation des Servers („Empfänger“):

Datenstrukturen:
- Empfangspuffer mit N Plätzen (N: Window-Größe)
- rcvbase (die Sequenznummer des nächsten Pakets, das bei Reihenfolgeerhaltung ausgeliefert werden muss)

Ereignisse und Aktionen:
Paket mit Sequenznummer n korrekt empfangen und n in [rcvbase, rcvbase+N-1]
- Sende ACK(n)
- Sortiere Paket n anhand der Sequenznummer in den Empfangspuffer ein (falls dort noch nicht vorhanden)
- Liefere - beginnend bei rcvbase - alle Pakete aus, die sich in lückenlos aufsteigender Reihenfolge im Empfangspuffer befinden und lösche diese aus dem Empfangspuffer
- Setze rcvbase = Sequenznummer des letzten ausgelieferten Pakets + 1

Paket mit Sequenznummer n korrekt empfangen und Paket n in [rcvbase-N,rcvbase-1]
- Sende ACK(n)

Paket mit Sequenznummer n korrekt empfangen und (n < rcvbase-N oder n ≥ rcvbase+N) oder Paket ist nicht korrekt:
- Nichts tun

1. Aufgabe: Ergänzung der Implementierung

Implementieren Sie einen **FileCopy-Client** in JAVA. Ergänzen Sie hierzu das mitgelieferte „Gerüst" in der Datei `FileCopyClient.java` (auch durch weitere Klassen/Threads)!
Simulieren Sie dabei eine zusätzliche Übertragungsverzögerung von 1 ms pro Paket, indem Sie den sendenden Thread für jedes Paket 1 ms „schlafen" lassen (→ L/R > 1 ms).

Geforderte Ergebnisausgaben:

1. Gesamt-Übertragungszeit für eine Datei
2. Anzahl an Timerabläufen (Timeouts)
3. der gemessenen Mittelwert für die RTT

Sie sollten folgendes berücksichtigen:

- Definieren Sie Sequenznummern als long-Zahlen und starten Sie für jede Datei bei 0 (das Paket mit Sequenznr. 0 enthält die Parameter für den Server)! Integrieren Sie eine Sequenznummer in jedes UDP-Datenpaket (die ersten 8 Byte des Daten-Bytearrays). Benutzen Sie dafür die Methoden der mitgegebenen Klasse `FCpacket` (speichern Sie im Sendepuffer nur Pakete vom Typ `FCpacket`, damit Sie auch weitere Zusatzinfos für ein gesendetes Paket ablegen können)!
- Implementieren Sie den Empfang der Quittungen beim Sender (Client) als einen weiteren Thread und benutzen Sie den Sendepuffer, der die unbestätigten Pakete speichert, zur Synchronisation *(→ Vorlage: Erzeuger/Verbraucher-Problem mit beschränktem Puffer aus der BS-Vorlesung!!)*
- Starten Sie für jeden Timer einen Thread der mitgegebenen Klasse `FC_Timer` und passen Sie den Code für Ihre Implementierung an (Methode `timeoutTask()`).
- Berechnen Sie dynamisch die aktuelle Timeoutzeit gemäß TCP-Algorithmus!

2. Aufgabe: Auswertung

Testen Sie Ihre Anwendung mit der Datei **`FCData.pdf`** und Werten für die Fenstergröße (max. Anzahl unbestätigter Pakete = WINDOW_SIZE) von 1, 15 und 100 bei einer ERROR_RATE von 1000 (d.h. ohne Fehler) sowie bei einer Fenstergröße von 100 und ERROR_RATE von 100 und 10 (siehe Tabelle unten).

- Verwenden Sie für die folgenden Auswertungsläufe **zwei verschiedene Rechner** für Client und Server sowie **lokale Verzeichnisse** für die zu übertragenden Dateien (um die Messung nicht durch zusätzlichen Netzwerkverkehr zu verfälschen).

- Lassen Sie dieselbe Datei für jede Parameterkonfiguration 3 mal übertragen (Achtung: Der Server muss wieder bereit sein nach einer Übertragung!) und notieren Sie die **Mittelwerte für die oben genannten Ergebnisausgaben**!

- Verwenden Sie die Methode `testOut` nur für Tests, nicht für Auswertungsläufe!

WINDOW_SIZE	ERROR_RATE	Übertragungszeit [ms]	Anzahl Timeouts	Mittlere RTT [ms]
1	1000			
15	1000			
100	1000			
100	100			
100	10			

Begründen Sie jeweils die Plausibilität Ihrer Ergebnisse!

Viel Spaß!