Aufgabe 3a: Programmierung eines Dateitransfer-Clients

Zu entwickeln ist ein Client, der an einen Dateitransfer-Server (vorgegeben, in Java) Dateien transferiert und hierzu UDP-Pakete verwendet, d.h. selbsttätig sich um Reihenfolge und Verluste kümmern muss. Als Verfahren soll das auch in der Vorlesung behandelte „Selective-Repeat“ verwendet werden.

Die Realisierung kann in Java, Python, C oder C++ erfolgen. Für Java gibt es bereits einen Rumpf sowie verschiedene Hilfsklassen. Eine GUI ist nicht erforderlich, da es sich um ein Kopierutility handelt.

Client:

Der FileCopyClient erhält vom Benutzer folgende Parameter als Argumente übergeben:

1. Hostname oder IP-Adresse des Servers
2. Portnummer des Servers
3. Quellpfad inkl. Dateiname der zu sendenden lokalen Datei
4. Zielpfad inkl. Dateiname der zu empfangenden Datei
5. Window-Größe N (als INT)
6. Fehlerrate zur Auswertung für den Server (ERROR_RATE, als LONG)

Als erstes Paket (Sequenznummer 0) überträgt der Client die Steuerdaten:

Datum	Datentyp	Beschreibung
Sequenznummer	8 Byte ohne Vorzeichen	Das erste Paket hat immer die Sequenz 0, danach wird je Paket hochgezählt
Zieldateipfad	String in UTF-8-Codierung mit beliebiger Länge ohne das Zeichen „;“	Pfad und Dateiname der Zieldatei auf dem Server (falls bereits vorhanden, wird die Datei überschrieben)
Trennzeichen	1 Char	„;“ (Semikolon) als Trennzeichen
Window-Größe	1 Int	Die Angabe der Window-Größe gemäß Verfahren
Trennzeichen	1 Char	„;“ (Semikolon) als Trennzeichen
Fehlerrate	1 Long	Die Angabe der Fehlerrate gemäß Verfahren

Anschließend überträgt der Client die Datei an der Server, die dort gespeichert wird. Der Client muss dafür sorgen, dass verloren gegangene Pakete von ihm entdeckt und gemäß dem **Selective-Repeat**-Verfahren neu übertragen werden. Lediglich zur dynamischen Berechnung der **Timeoutzeit** soll der Algorithmus des TCP-Verfahrens verwendet werden (siehe Vorlesung).

Die Client-Software ist vor dem Praktikum zu entwerfen und kann auf der Vorlage aufsetzen; das Entwurfsdokument muss zum Praktikum vorliegen.

Server:

Ein neuer Kopierauftrag wird beim Eintreffen des ersten UDP-Pakets gestartet, wobei der Client durch IP-Adresse und Quellport eindeutig festgelegt ist. Der Server ist nicht multi-threadfähig, es kann nur ein Kopierauftrag zur Zeit verarbeitet werden, 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 für beendet und schließt die Speicherung der Dateidaten ab. Damit steht der Client 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 (sogenanntes ACK). Jedes ACK erhält allerdings eine zusätzliche Verzögerung von 10 ms zur 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 Fehlerrate wird vom Client mit dem ersten Paket (Sequenznr. 0) als Teil der Steuerdaten übermittelt.

FSM-Spezifikationen

Client:

- 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

Server:

- 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 auf die Sequenznummer des letzten ausgelieferten Pakets n + 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):
    - keine Aktion

Allgemeine Randbedingungen und Hinweise:

- Besonderer Wert wird auf Fehlertoleranz und Stabilität des Servers gelegt. Der Server soll auch bei Fehlverhalten von Clients (z.B. Verbindungsabbrüche, Protokollfehler oder Überlast durch zu viele Clients) weiterlaufen.

- Der Listening-Port muss zur Laufzeit einstellbar sein (Parameter). Denken Sie beim Testen daran, dass Ports unterhalb bestimmter Grenzen vom Betriebssystem reserviert sein können.

- Zur Verarbeitung von Informationen über alle die jeweiligen Objekte sind jeweils eigene Datenstrukturen (Klassen) zur Beschreibung hilfreich.

- Integrieren Sie eine Sequenznummer in jedes UDP-Datenpaket (die ersten 8 Byte des Daten-Bytearrays). Benutzen Sie dafür die Methoden der mitgegebenen Klasse FCpacket und 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 nutzen Sie den Sendepuffer, der die unbestätigten Pakete speichert, zur Synchronisation.

Aufgabe 3b: Analyse von Dateitransfers

Testen Sie Ihre Anwendung mit der Datei **`FCData.pdf`** und verschiedenen Werten für die Fenstergröße (WINDOW_SIZE), also die max. Anzahl unbestätigter Pakete = von 1, 10 und 100 bei einer gleichzeitigen Fehlerrate (ERROR_RATE) von 10, 100 und 1000 (1000 entspricht einer Übertragung ohne Fehler).

Geben Sie dazu durch den Client direkt folgende Angaben aus:

a) Gesamtübertragungszeit für eine Datei
b) Anzahl an Timerabläufen und wiederholten Übertragungen
c) Anzahl an empfangenen ACKs
d) der gemessenen Mittelwert für die RTT aller ACKs

Verwenden Sie für die Auswertungsläufe immer **zwei unterschiedliche 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 jeweils fünf (5) mal übertragen, wobei der Server natürlich bereits wieder für eine erneute Übertragung bereit sein muss. Notieren sie alle Ergebnisse dieser Auswertungsläufe.

Für das Protokoll dokumentieren Sie bitte:

- die Ausgaben des Clients sowie die für Fenstergröße und Fehlerrate verwendeten Angaben, sowie
- eine kurze Analyse der erhobenen Daten mit Begründung, wodurch die Veränderungen zu erklären wären.

Die Aufzeichnung kann auch noch im Praktikum angefertigt werden.