Dokumentation Beleg Rechnernetze

Jakub Kliemann

04.01.2024

Inhaltsverzeichnis

1 Aufgabenstellung

Erstellen Sie ein Programm (Client + Server) zur Übertragung beliebiger Dateien zwischen zwei Rechnern, basierend auf dem UDP-Protokoll. Das Programm soll mit der Sprache JAVA erstellt werden und im Labor S311 unter Linux lauffähig sein und dort vorgeführt werden. Folgende Punkte sind umzusetzen:

1. **Aufruf des Clients** (Quelle) auf der Konsole mit den Parametern: Zieladresse (IP oder Hostname) + Portnummer + Dateiname + Fenstergröße (1-100)
   (Bsp.: `filetransfer client is311p1 3333 test.gif 1`) (Fenstergröße 1 entspricht Protokoll SW, wird kein Wert angegeben, soll automatisch der Wert 1 verwendet werden)

2. **Aufruf des Servers** (Ziel) mit den Parametern: Portnummer
   (Bsp.: `filetransfer server 3333`). Um die Aufrufe für Client und Server so zu realisieren ist ein kleines Bash-Skript notwendig (z.B.: `java Clientklasse $1 $2 $3`)

3. Auf dem Zielrechner (Server) ist die Datei unter Verwendung des korrekten Dateinamens im Pfad des Servers abzuspeichern. Ist die Datei bereits vorhanden, soll an den Basisnamen der neuen Datei das Zeichen „1“ angehängt werden. Client und Server sollten auch auf demselben Rechner im selben Pfad funktionieren.

4. Messen Sie bei der Übertragung die Datenrate und zeigen Sie am Client periodisch (z.B. jede Sekunde) den aktuellen Wert und am Ende den Gesamtwert an. Sie können sich bei der Anzeige am Konsolenprogramm `wget` orientieren.

5. Implementieren Sie exakt das im Dokument `Beleg-Protokoll.md` vorgegebene Übertragungsprotokoll. Damit soll gewährleistet werden, dass Ihr Programm auch mit einem beliebigen anderen Programm funktioniert, welches dieses Protokoll implementiert.

6. Gehen Sie davon aus, dass im Labor Pakete fehlerhaft übertragen werden können, verloren gehen können oder in ihrer Reihenfolge vertauscht werden können (beide Richtungen!). Implementieren Sie eine entsprechende Fehlerkorrektur.

7. **Testen** Sie Ihr Programm ausgiebig, hierzu sind Debug-Ausgaben sinnvoll. Entwerfen Sie eine Testumgebung als Bestandteil des Servers, mittels derer Sie eine bestimmte Paketverlustwahrscheinlichkeit und Paketverzögerung für **beide Übertragungsrichtungen** simulieren können. Sinnvollerweise sollten diese Parameter über die Konsole konfigurierbar sein, z.B.: `filetransfer server 3333 0.1 150` für 10% Paketverluste und 150 ms mittlere Verzögerung für **beide** Richtungen. Bei der Vorführung im Labor werden wir einen Netzsimulator nutzen, welcher eine entsprechende Netzqualität simuliert.

8. Bestimmen Sie den theoretisch maximal erzielbaren Durchsatz bei 10% Paketverlust und 10 ms Verzögerung mit dem SW-Protokoll und vergleichen diesen mit Ihrem Programm. Begründen Sie die Unterschiede.

9. Erstellung eines Lernportfolios (Dokumentation Ihrer Entwicklungsschritte, des Lernfortschritts, der Misserfolge, etc.)

10. Dokumentieren Sie die Funktion Ihres Programms unter Nutzung von LaTeX. Notwendig ist mindestens ein Zustandsdiagramm für Client und Server. Geben Sie Probleme/Limitierungen/Verbesserungsvorschläge für die Belegaufgabe und das verwendete Protokoll an.

11. Der Abgabetermin ist auf der Website des Fachs zu finden, die Vorführung der Aufgabe findet dann zu den angekündigten Praktikumszeiten statt. Die Abgabe des Belegs erfolgt als tar-Archiv mit einem vorgegebenen Aufbau, Informationen hierzu werden im Dokument `Beleg-Abgabeformat.md` bereitgestellt. **Plagiate werden mit Note 5** bewertet!

12. Sie können zur Programmierung einen beliebigen Editor / Entwicklungsumgebung verwenden. Empfohlen wird die Entwicklungsumgebung IntelliJ IDEA, welche für Studenten kostenlos erhältlich ist.

13. Die Note 1 können Sie nur erhalten, wenn beide Protokolle (SW und GBN) korrekt implementiert sind.

2 Durchsatz

2.1 Berrechung des theoretischen Durchsatzes

Bestimmen Sie die Durchsatz für die gegebenen Größen $P_{\text{de}} = P_{\text{rü}} = 0.1$ und $\text{RTT} = 2 \cdot 10\,\text{ms} = 20\,\text{ms}$. Dazu ist folgende Fromel aus der Vorlesung gegeben:

$$\eta_{\text{sw}} = \frac{T_p}{T_p + T_w}(1 - P_{\text{de}})(1 - P_{\text{rü}}) \tag{1}$$

Außerdem sind folgende Werte bekannt:

$$\begin{aligned} L_p &= L_{\text{Daten}} + L_{\text{Header}} \\ &= 3\,\text{Byte} + 1400\,\text{Byte} = 1403\,\text{Byte} = \underline{11224\,\text{Bit}} \end{aligned}$$

$$\begin{aligned} L_{\text{ACK}} &= L_{\text{PNr}} + L_{\text{SNr}} + L_{\text{GBN}} \\ &= 2\,\text{Byte} + 1\,\text{Byte} + 1\,\text{Byte} = 4\,\text{Byte} = \underline{32\,\text{Bit}} \end{aligned}$$

Für die Berechnung müssen folgende Werte berrechnet werden für Annahme $r_b = 1GBit/s$:

$$\begin{aligned} T_a &= \frac{\text{RTT}}{2} = 10\,\text{ms} \\ T_p &= \frac{L_p}{r_b} = \frac{11224\,\text{Bit}}{10^9\,\text{Bit/s}} = 0.011224\,\text{ms} \\ T_{\text{ACK}} &= \frac{L_{\text{ACK}}}{r_b} = \frac{32\,\text{Bit}}{10^9\,\text{Bit/s}} = 0.000032\,\text{ms} \\ T_w &= 2T_a + T_{\text{ACK}} = 2 \cdot 10\,\text{ms} + 0.000032\,\text{ms} = 20.000032\,\text{ms} \\ \eta_{\text{sw}} &= \frac{0.011224\,\text{ms}}{0.011224\,\text{ms} + 20.000032\,\text{ms}}(1 - 0.1)(1 - 0.1) \approx 0.000454316 = \underline{\underline{450\,\text{KBit/s}}} \end{aligned}$$

2.2 Vergleich mit realem Durchsatz

Der praktische Durchsatz für die Datenübertragung liegt bei etwa 40 KBit/s, was weit unter dem erwarteten Wert von 450 KBit/s liegt. Diese Abweichung ist hauptsächlich auf die Verlangsamung der Übertragung durch Timeouts im Stop-and-Wait-Protokoll zurückzuführen.

Um den Datenübertragungsdurchsatz zu optimieren, könnte eine Strategie in Erwägung gezogen werden, bei der die Timeouts dynamisch angepasst werden. Dies könnte insbesondere dann erfolgen, wenn vermehrt oder vermindert Paketverluste auftreten. Durch eine adaptive Anpassung der Timeouts könnte das System effizienter auf Netzwerkbedingungen reagieren und eine bessere Leistung erzielen, was zu einer näheren Annäherung an den erwarteten Durchsatz von 450 KBit/s führen könnte.

3 Lernportfolio

In dem erstem Praktikum zum Beleg begann ich mit der Entwicklung einer stark vereinfachten Übertragung eines Startpakets, die ich im Verlauf weiter optimierte. Auf der Client-Seite habe ich dann die File Transfer (FT) Schicht soweit vervollständigt, dass die gesamte Datei in Paketen übermittelt wird. Anschließend habe ich die Stop-and-Wait (SW) Schicht ohne Fehlerkorrektur und ohne Berücksichtigung der Paketnummern implementiert.

Um meine erste Datei in einem fehlerfreien Kanal an den Server zu übertragen, musste ich dann die FT-Schicht des Servers vervollständigen, damit dieser alle Pakete empfangen und in eine Datei schreiben kann. Danach habe ich die korrekte Implementierung der Paketnummern und das wiederholte Senden im Fehlerfall hinzugefügt. Auf diese Weise konnte ich die Datenübertragung mit dem Idefix-Server testen.

Nachdem dies erfolgreich war, habe ich auf meinem Server auch die Berücksichtigung der Paketnummern implementiert. Als letzten Schritt zur erfolgreichen Übertragung habe ich die Übertragung der CRC über die gesamte Datei beidseitig gewährleistet und somit erfolgreich eine Datei auf den Idefix-Server hochgeladen.

Im weiteren Verlauf habe ich die regelmäßige Ausgabe der Datenrate des Clients implementiert. Abschließend habe ich noch Details wie die `closeConnection`-Funktion und das Erstellen einer neuen Datei (wenn bereits eine mit dem gleichen Namen existiert, z. B. `test.txt` → `test(1).txt`) auf der Seite des Servers eingerichtet. Danach habe ich meinen Code weiter verfeinert, bis ich ihn als übersichtlich genug empfand.

Probleme: Bei der Implementierung musste ich erheblichen Aufwand in die Hand nehmen, um die Paketnummer zu handhaben, da es in Java eigentlich keine vorzeichenlosen Bytes gibt. Schlussendlich erwies sich die Lösung jedoch als recht simpel, als mir klar wurde, dass die Bitzusammensetzung zwischen vorzeichenbehafteten (`signed`) und vorzeichenlosen (`unsigned`) Bytes im Grunde genommen identisch ist und nur bei der Ausgabe entsprechend transformiert werden muss.

Des Weiteren stieß ich auf Herausforderungen bei der Übertragung des CRC-Werts vom Server zum Client. Die Lösung erwies sich letztendlich als unkompliziert, nachdem ich die `backData`-Variable fand, welche dazu angedacht war.

Beim Programmieren des Timers für die regelmäßige Ausgabe der Datenrate hatte ich Schwierigkeiten, da ich zuvor noch nie einen Timer implementiert hatte. Diese Hürde konnte ich jedoch überwinden, nachdem ich meinen Professor aufsuchte und von ihm Unterstützung erhielt.

4 Dokumentation der Implementierung

4.1 Allgemein

Ich habe ausschließlich das Stop-and-Wait-Protokoll (SW-Protokoll) als ARQ-Schicht implementiert. Gemäß den Vorgaben werden Pakete mit einer Größe von 1400 Byte verschickt. Jedes Paket ist mit einer ein Byte großen Paketnummer versehen, die im Bereich von 0 bis 255 variiert, sowie einer 4-Byte großen (short) Session-ID. Es sei angemerkt, dass im üblichen Kontext des Stop-and-Wait-Protokolls Paketnummern im Bereich von 0 bis 1 verwendet werden. Für die Kommunikation mit dem bereitgestellten Server, der auch das Go-Back-N-Protokoll (GBN) verarbeiten soll, werden jedoch Paketnummern inkrementierend bis 255 genutzt.

4.2 Filetransfer

4.2.1 file_req()

Die `file_req()` Methode wird im Client aufgerufen, um eine Anfrage zur Übertragung einer Datei an den Server zu initiieren. Sie führt den Dateiübertragungsprozess durch, indem sie das Startpaket sendet, die Datei einliest, in Datenblöcke aufteilt und diese an den Server überträgt.

Funktionalität

1. **Startpaket senden**: Sendet ein Startpaket an den Server mit wichtigen Informationen wie Dateilänge und Dateiname.
2. **Einlesen der Datei**: Öffnet die Datei im Client, liest sie vollständig in einen Puffer und schließt die Datei.
3. **Datenübertragung in Blöcken**: Teilt die Dateidaten in Blöcke von 1400 Bytes auf und sendet sie nacheinander an den Server.
4. **CRC32-Berechnung und -übertragung**: Berechnet eine Prüfsumme (CRC32) über die gesamte Datei und sendet sie an den Server, um die Datenintegrität zu überprüfen.
5. **Überprüfung der Übertragung**: Vergleicht die empfangene Prüfsumme mit der berechneten Prüfsumme auf Serverseite, um die erfolgreiche Übertragung zu bestätigen.
6. **Datenratenmessung**: Misst die Datenübertragungsrate während des gesamten Prozesses und gibt diese periodisch sowie am Ende des Transfers aus.

Datenratenberechnung Die Methode misst die Übertragungszeit der gesamten Datei und berechnet daraus die Datenübertragungsrate in Kilobyte pro Sekunde.

4.2.2 file_init()

Die `file_init()` Methode wird im Server aufgerufen, um die Initialisierung des Dateiübertragungsprozesses zu veranlassen. Sie handhabt den Empfang des Startpakets vom Client, überprüft die Integrität und korrekte Struktur des Startpakets, liest die Datei-Daten ein und schreibt sie in eine Datei auf dem Server.

Funktionalität

1. **Empfang des Startpakets**: Wartet auf und empfängt das Startpaket vom Client, das wichtige Informationen wie Dateilänge und Dateiname enthält.
2. **Überprüfung des Startpakets**: Überprüft die Gültigkeit des erhaltenen Startpakets, indem es die Startkennung, Dateilänge und den Dateinamen validiert.
3. **Empfang der Dateidaten**: Empfängt die Datenblöcke der Datei in 1400-Byte-Blöcken vom Client.
4. **CRC32-Überprüfung**: Überprüft die Integrität der übertragenen Datei, indem es eine CRC32-Prüfsumme über die empfangenen Daten berechnet und mit der vom Client berechneten Prüfsumme vergleicht.
5. **Dateischreiben auf dem Server**: Schreibt die empfangenen Daten in eine Datei auf dem Server. Wenn eine Datei mit demselben Namen bereits vorhanden ist, hängt es eine 1 an den Basisnamen an, um die neue Datei zu speichern.

4.3 SW-Schicht

4.3.1 data_req()

Die `data_req()` Methode im Stop-and-Wait-Protokoll wird verwendet, um Datenpakete zu senden und die Bestätigung (ACK) vom Empfänger zu empfangen. Sie handhabt den Prozess des Sendens und Empfangens von Datenpaketen im Stop-and-Wait-Protokoll.

Funktionalität

1. **Generierung des Datenpakets**: Erstellt ein Datenpaket mit den empfangenen Daten und anderen erforderlichen Informationen für den Datenaustausch im Stop-and-Wait-Protokoll.
2. **Senden des Datenpakets**: Überträgt das erstellte Datenpaket an den Empfänger und wartet auf die Bestätigung (ACK).
3. **Wiederholter Versand im Fehlerfall**: Bei Nichterhalt einer ACK-Bestätigung wird das Datenpaket erneut gesendet. Es werden mehrere Versuche unternommen, das Paket zu senden, bis eine Bestätigung erfolgt oder die maximal erlaubten Versuche erschöpft sind.
4. **Überlaufbehandlung**: Die Methode verwaltet die Überlaufbedingungen der Paketnummern im Stop-and-Wait-Protokoll. Bei Erreichen des maximalen Wertes der Paketnummer erfolgt eine Rücksetzung auf den Anfangswert, um einen Überlauf zu verhindern.
5. **Verwaltung des Stop-and-Wait-Verfahrens**: Stoppt nach dem Senden eines Pakets den weiteren Sendevorgang, bis die Bestätigung (ACK) vom Empfänger empfangen wurde.

Rückgabewert Die Methode gibt einen booleschen Wert zurück, der angibt, ob das Datenpaket erfolgreich gesendet wurde und eine Bestätigung erhalten wurde.

4.3.2 data_ind_req()

Die `data_ind_req()` Methode im Stop-and-Wait-Protokoll wird verwendet, um Datenpakete zu empfangen und Bestätigungen (ACK) an den Sender zu senden. Sie handhabt den Prozess des Empfangs von Datenpaketen und die Erzeugung von ACK-Paketen im Stop-and-Wait-Protokoll.

Funktionalität

1. **Empfang von Datenpaketen**: Wartet auf den Empfang eines Datenpakets vom Sender und überprüft die Gültigkeit der Sessionnummer und der Paketnummer.

2. **Generierung von ACK-Paketen**: Erzeugt ein ACK-Paket als Bestätigung für den Empfang des Datenpakets. Das ACK-Paket enthält die Paketnummer des empfangenen Datenpakets und sendet es an den Sender zurück.

3. **Behandlung von Fehlern**: Behandelt Fehlerfälle wie Paketverluste, falsche Paketnummern und Überlaufbedingungen der Paketnummern.

4. **Überlaufbehandlung**: Überwacht und verwaltet die Überlaufbedingungen der Paketnummern im Stop-and-Wait-Protokoll, indem sie bei Erreichen des maximalen Wertes der Paketnummer auf den Anfangswert zurückgesetzt wird, um einen Überlauf zu verhindern.

Rückgabewert Die Methode gibt die empfangenen Datenpakete zurück, um sie von der Anwendungsschicht weiterverarbeiten zu lassen.

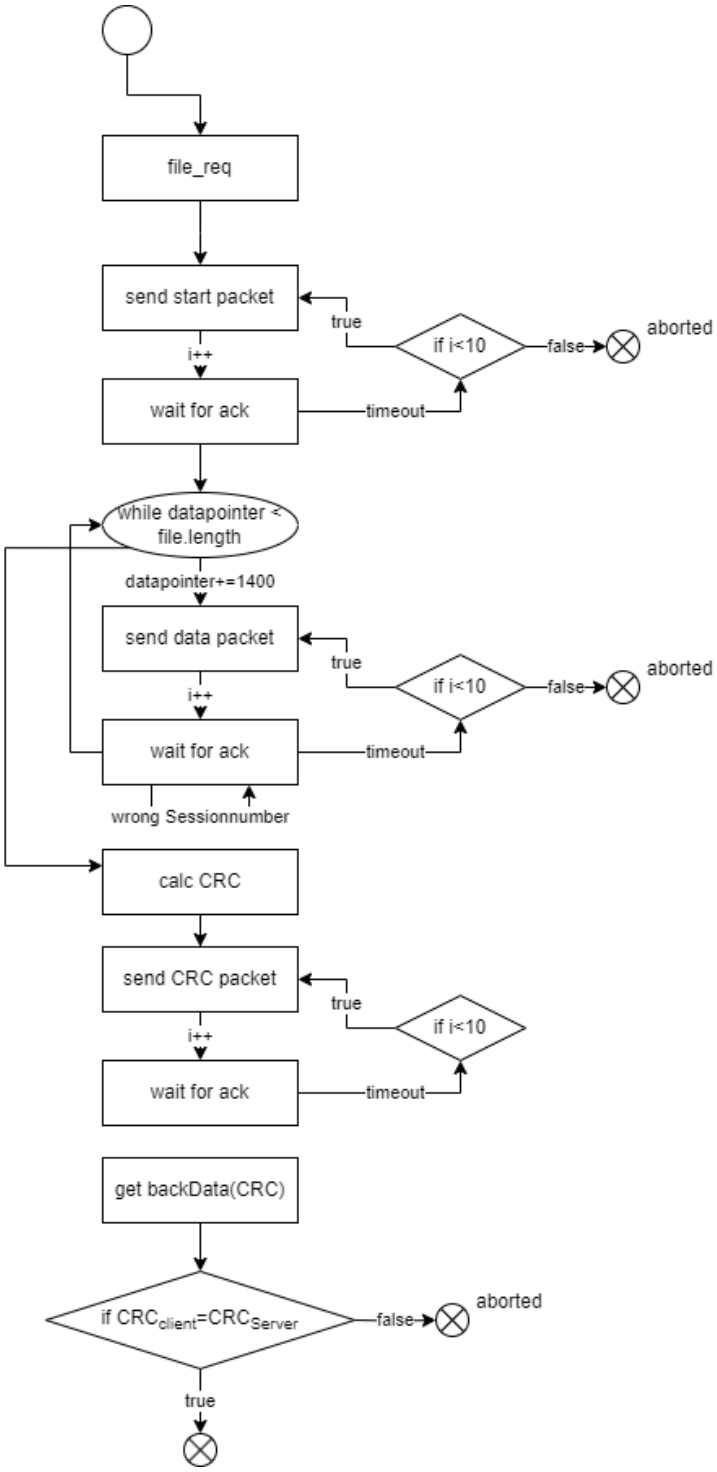

Abbildung 1: Zustandsdiagramm des Clients

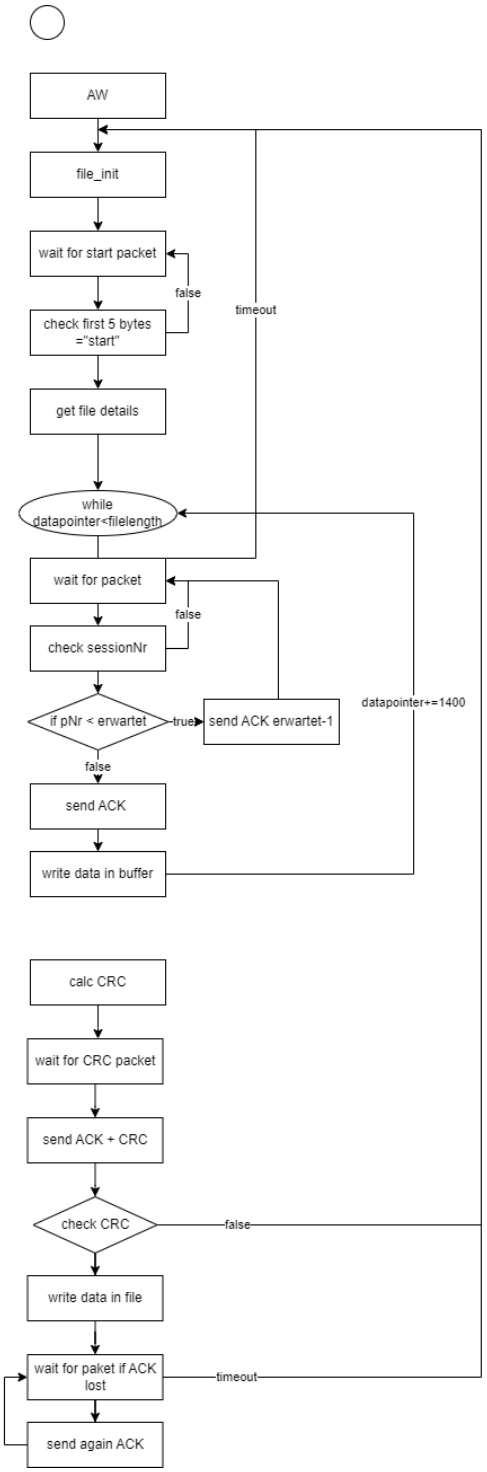

Abbildung 2: Zustandsdiagramm des Servers

5 Probleme/Limitierungen/Verbesserungsvorschläge

Während meiner Implementierung konnte ich keine erkennbaren Probleme oder Fehler feststellen. Allerdings besteht die Möglichkeit, die Leistungsfähigkeit durch eine verbesserte dynamische Anpassung der Timeouts zu steigern, wie bereits zuvor erwähnt wurde.

Es ist besonders wichtig zu unterstreichen, dass das Stop-and-Wait-Protokoll aufgrund seiner einfachen Struktur ineffizient sein kann, insbesondere in Bezug auf den Durchsatz. Bei vergleichbaren Übertragungsszenarien mit relevanten Datenmengen wäre es ratsam, alternative Protokolle wie das Go-Back-N-Protokoll (GBN) oder das Selective-Repeat-Protokoll (SR) in Betracht zu ziehen, wie sie in der Vorlesung vorgestellt wurden. Diese bieten in der Regel eine verbesserte Effizienz und Leistung im Vergleich zum Stop-and-Wait-Protokoll. Der Wechsel zu solchen Protokollen könnte dazu beitragen, die Gesamtleistung der Datenübertragung zu optimieren.