Friedrich-Alexander-Universität
Technische Fakultät

Rechnerkommunikation

Aufgabenblatt 5

Zuverlässiges Transportprotokoll

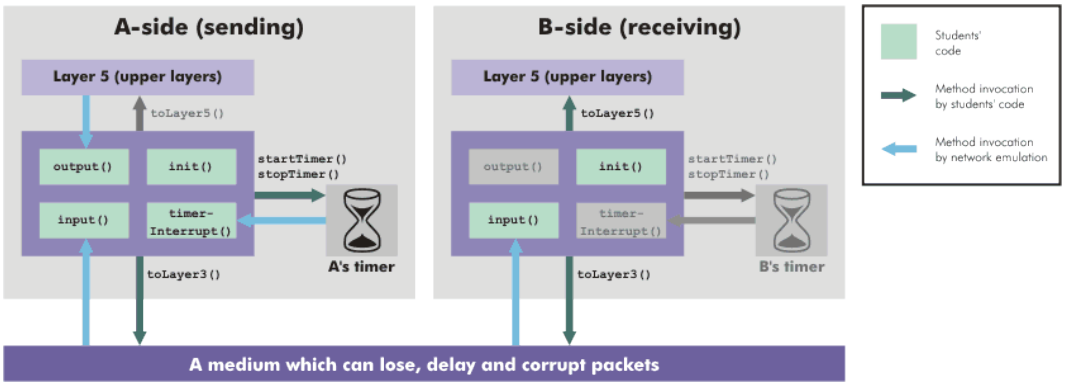
Abgabe vor Ort in der Übung (**in KW 25**).
Zusätzlich Upload der Quelldateien *ABP.java und GoBackN.java* als Team über StudOn.
Ihre Programme müssen kompilierbar und lauffähig sein!

Ziel dieser Aufgabe

In dieser Programmieraufgabe soll der Code des Senders und des Empfängers für die Transportschicht eines einfachen zuverlässigen Datentransfer-Protokolls erstellt werden. Die Transportschicht ist üblicherweise Teil des Betriebssystems, weshalb die Hardware- und Software-Umgebung hier emuliert wird. Die Programmierschnittstelle zu Ihren eigenen Routinen, d.h. der Code, der Ihre Routinen von höheren und niedrigeren Schichten des Protokolls aus aufruft, ist den Aufrufen in einem realen Unix-System sehr ähnlich. Das Starten und Stoppen von Timern wird ebenfalls emuliert, beim Ablauf des Timers wird ein Timer-Interrupt ausgelöst, der Ihre Timer-Interrupt-Behandlungsroutine aktiviert.

Von Ihnen zu implementierende Methoden

Die von Ihnen zu implementierenden Methoden kommen auf der sendenden Seite (A) und der empfangenden Seite (B) zum Einsatz. In dieser Aufgabe sollen Daten nur unidirektional (von A zu B) übertragen werden. Trotzdem muss die B-Seite in der Lage sein, Bestätigungsnachrichten an A zu senden. Die dazu benötigten Routinen sind in Form von unten beschriebenen Methoden zu implementieren. Diese Methoden werden von der von uns zur Verfügung gestellten Emulationsumgebung aufgerufen. Weiterhin müssen Sie in Ihren Routinen Methoden aufrufen, die von der Emulationsumgebung zur Verfügung gestellt werden (z.B. um Ihre Daten an darunterliegende Protokollebenen zu übergeben und über das emulierte Netzwerk zu versenden). Die Gesamtstruktur der Umgebung ist in folgendem Bild veranschaulicht:



Die Dateneinheit, die zwischen den darüberliegenden Schichten und Ihrem Protokoll ausgetauscht wird, ist eine *message* (`NWEmuMsg`), die `NWEmu.PAYSIZE = 20` Bytes Nutzdaten enthält, und wurde in unserem Code wie folgt deklariert:

```
class NWEmuMsg {
    byte data[] = new byte[NWEmu.PAYSIZE];
}
```

Diese Deklaration und alle anderen Datenstrukturen, Emulationsroutinen und vordefinierte leere Methoden (die Sie überschreiben müssen) sind im Package `fau.cs7.nwemu` in der Datei `NWEmu.jar` enthalten, auf die wir später näher eingehen wollen. Ihre Senderseite wird also 20 Byte große Stücke mit Daten von Schicht 5 bekommen, Ihre Empfänger-Seite soll korrekt empfangene Daten wiederum in 20 Byte großen Stücken an die Schicht 5 übergeben.

Zum Datenaustausch zwischen Ihren eigenen Routinen und der Netzwerkschicht wird ein *packet* (`NWEmuPkt`) verwendet, das wie folgt deklariert ist:

```
class NWEmuPkt {
    int seqnum;              // vgl. Vorlesung
    int acknum;              // vgl. Vorlesung
    int checksum;            // vgl. Vorlesung
    int flags;               // fuer Experimente

    /* Nutzdaten */
    byte payload[] = new byte[NWEmu.PAYSIZE];
}
```

Ihre Routinen müssen das `payload`-Feld mit den Daten füllen, die von oben aus Schicht 5 als *message* übergeben wurden. Die anderen Felder sind zur Realisierung einer zuverlässigen Auslieferung durch Ihr Protokoll vorgesehen, wie es in der Vorlesung gezeigt wurde (das `flags`-Feld dient eigenen Experimenten bzw. Erweiterungen). Weitere Erläuterungen zur Verwendung von `seqnum` und `acknum` finden Sie in den Hinweisen am Ende des Aufgabenblattes.

Nun zu den Details der von Ihnen zu implementierenden Methoden: Diese Methoden sind in der Klasse `AbstractHost` als leere Methoden deklariert. Sie müssen also Unterklassen von `AbstractHost` sowohl für die Sender-Seite als auch für die Empfängerseite ableiten, in denen Sie die vorgegebene Implementierung aller von Ihnen benötigten Methoden überschreiben. Wie aus obigem Diagramm ersichtlich ist, sind nicht alle Methoden auf beiden Seiten nötig, so braucht die Empfänger-Seite beispielsweise keine `output()`-Methode. Solche Prozeduren sind im Normalfall Teil des Betriebssystems und würden von anderen Betriebssystem-Prozeduren aufgerufen. **Die folgenden Methoden werden also von der Emulationsumgebung aufgerufen (Sie selbst dürfen diese nicht aufrufen!):**

- **output(NWEmuMsg message)**

  `message` ist ein Objekt der Klasse `NWEmuMsg`, das die Daten enthält, die an die Gegenseite versendet werden sollen. Diese Methode wird immer dann aufgerufen, wenn die oberen Schichten auf der Sender-Seite (A) Nachrichten zu verschicken haben. Es ist Aufgabe Ihres Protokolls sicherzustellen, dass die Nachrichten in richtiger Reihenfolge und korrekt an die oberen Schichten auf der Empfänger-Seite ausgeliefert werden. Wurde ein Paket erfolgreich verschickt, wird `true` zurückgegeben. Bei einem Rückgabewert `false` wird Ihnen die Nachricht zu einem späteren Zeitpunkt nochmals übergeben.

- **input(NWEmuPkt packet)**

  `packet` ist ein Objekt der Klasse `NWEmuPkt`. Diese Methode wird immer dann aufgerufen, wenn ein Paket, das von der Gegenseite versendet wurde, auf dieser Seite eintrifft. `packet` ist das (möglicherweise fehlerhafte) Paket, das von der anderen Seite über das emulierte Netzwerk versendet wurde.

- **timerInterrupt()**

  Diese Methode wird aufgerufen, wenn der Timer dieser Seite abläuft (d.h. ein Timer-Interrupt generiert wird). Möglicherweise ist dies nützlich, um eine Sendewiederholung anzustoßen. Die unten beschriebenen Methoden `startTimer()` und `stopTimer()` dienen zum Starten und Stoppen des Timers.

- **init()**

  Diese Methode wird einmal aufgerufen, wenn die Emulation gestartet wird. Sie kann dazu verwendet werden, nötige Initialisierungen vorzunehmen.

Software-Interface

Die oben beschriebenen Methoden sind von Ihnen zu implementieren, dürfen aber wie erwähnt nicht von Ihnen aufgerufen werden. Hingegen werden die folgenden Methoden der Klasse `AbstractHost` von uns zur Verfügung gestellt und können in Ihren eigenen Methoden verwendet werden. Diese Methoden sollten bei der Ableitung von Unterklassen nicht überladen werden.

- **startTimer(double delay)**

  dient zum Starten des Timers dieser Seite. `delay` ist ein `double`-Wert, der angibt, wieviel Zeit von nun ab vergehen soll, bevor der Timer-Interrupt ausgelöst wird. Die Timer einer Seite sollen nur von Methoden der jeweiligen Seite gestartet und gestoppt werden. **Als Anhaltspunkt für einen sinnvollen Timeout-Wert kann folgende Bemerkung dienen**: Ein Paket, das auf das Netzwerk geschickt wird, benötigt im Mittel 5 Zeiteinheiten, um auf der anderen Seite anzukommen, wenn keine weiteren Nachrichten das Medium belegen.

- **stopTimer()**

  hält den Timer dieser Seite an. Zum Neustarten des Timers rufen Sie also `stopTimer()` gefolgt von `startTimer()` auf (nicht einfach nur `startTimer()`).

- **toLayer3(NWEmuPkt packet)**

  `packet` ist ein Objekt der Klasse `NWEmuPkt`. Mit dem Aufruf dieser Methode wird ein Paket von dieser Seite mit der anderen Seite als Ziel auf das emulierte Netzwerk geschickt.

- **toLayer5(NWEmuMsg message)**

  `message` ist ein Objekt der Klasse `NWEmuMsg`. Durch den Aufruf dieser Methode werden Daten an die Schicht 5 dieses Hosts nach oben gereicht.

Die Netzwerk-Emulationsumgebung

Ein Aufruf von `toLayer3()` sendet Pakete auf das Medium (d.h. die Netzwerk-Schicht). Ihre Methode `input()` wird aufgerufen, wenn ein Paket vom Medium an das Protokoll auf dem jeweiligen Host ausgeliefert werden soll. Das Medium kann den Inhalt von Paketen verändern (d.h. Fehler hervorrufen) und Pakete verlieren. Es ordnet Pakete jedoch nicht um, d.h. die Reihenfolge bleibt erhalten.

Die Emulation des Netzwerkes wird durch die Erzeugung einer Instanz der Klasse `NWEmu` initialisiert. Dem Konstruktor der Klasse `NWEmu` müssen dabei zwei Objekte der Klasse `AbstractHost` (oder von Subklassen dieser Klasse) als Argumente übergeben werden, wobei das erste Argument die Implementierung des Senders und das zweite Argument die Implementierung des Empfängers enthalten muss:

```
public NWEmu(AbstractHost sender, AbstractHost receiver)
```

Nach Erzeugung eines Objektes der Klasse `NWEmu` können Sie die Methode `emulate()` aufrufen, um die Emulation zu starten. Die Methode hat folgende Parameter:

```
public void emulate(int nsimmax,
                    double lossprob,
                    double corruptprob,
                    double lambda,
                    int trace)
```

- **nsimmax**

  Anzahl von Nachrichten, die in der Emulation generiert werden sollen. Der Emulator und Ihr Code werden beendet, sobald diese Anzahl an Nachrichten beim Empfänger an Schicht 5 nach oben gereicht wurde.

- **lossprob**

  Paketverlustwahrscheinlichkeit. Ein Wert von 0,1 bedeutet, dass im Mittel eines von zehn Paketen verloren wird.

- **corruptprob**

  Paketfehlerwahrscheinlichkeit. Ein Wert von 0,2 bedeutet, dass im Mittel eines von fünf Paketen fehlerhaft übertragen wird. Beachten Sie bitte, dass die Felder `payload`, `seqnum`, `acknum` oder `checksum` bei der Übertragung zerstört werden können. Daher sollte Ihre Prüfsumme die `payload`, `seqnum` und `acknum` umfassen.

- **lambda**

  Durchschnittliche Zeit zwischen aufeinanderfolgenden Nachrichten von Schicht 5 des Senders. Dieser Wert kann auf jeden Wert größer als Null gesetzt werden. Je kleiner dieser Wert ist, desto schneller kommen die sendebereiten Nachrichten beim Sender an.

- **trace**

  Wird Trace auf einen Wert größer als Null gesetzt, so gibt der Emulator nützliche Informationen über die internen Abläufe (z.b. über Pakete und Timer) aus. Ein Wert von 1 bewirkt nur die Ausgabe von Informationen über die an Schicht 5 übergebenen Pakete, während 5 sehr viele interne Details ausgibt. Ein Trace-Wert von 2 könnte sich als nützlich zum Debugging Ihres Codes erweisen. Bedenken Sie, dass diese sinnvollen Informationen bei der Entwicklung einer echten Implementierung nicht zur Verfügung stehen.

Aufgabe 5.1: Das Alternating-Bit-Protokoll (10 Punkte)

In dieser Aufgabe sind die Methoden `output()`, `input()`, `timerInterrupt()` und `init()` der Sender-Seite (Host A) und die Methoden `input()` und `init()` der Empfänger-Seite (Host B) zu implementieren, die zusammen einen unidirektionalen Datentransfer im Stop-and-Wait-Modus von Host A zu Host B realisieren. Sie sollen hier den Spezialfall des Alternating-Bit-Protokolls umsetzen, d.h. die Sequenz-/ACK-Nummern beim Stop-and-Wait-Mechanismus nehmen lediglich die Werte 0 und 1 an.

Sie sollten einen sehr hohen Wert für die mittlere Zeit zwischen den Nachrichten von Layer 5 der Sender-Seite wählen, so dass die `output()`-Methode des Senders nie aufgerufen wird, solange der Sender versucht, ein noch ausstehendes unbestätigtes Paket an den Empfänger zu senden. Ein Wert von 1000 für `lambda` wäre hier sinnvoll. Weiterhin sollten Sie eine Überprüfung in Ihren Sender einbauen, in der getestet wird, ob noch ein Paket in Übertragung begriffen ist, während `output()` aufgerufen wird. Ist dies der Fall, kann die an die `output()`-Methode übergebene Nachricht einfach verworfen und `false` zurückgegeben werden.

Führen Sie die Emulation so lange durch, bis etwa 10 Nachrichten korrekt vom Empfänger bestätigt wurden. Verwenden Sie eine Verlustwahrscheinlichkeit von 0,1, eine Fehlerwahrscheinlichkeit von 0,3 und setzen Sie den Trace-Wert auf 2.

Sie benötigen die Datei `NWEmu.jar`, die die Emulationsumgebung und die vordefinierten leeren Methoden enthält. Diese Datei ist über StudOn abrufbar. **Lesen Sie den Abschnitt *Hilfreiche Hinweise*** in dieser Aufgabe (nach dem Abschnitt *Go-Back-N*) gründlich. Die Benutzung von `NWEmu.jar` wird dort nochmals näher erklärt.

Aufgabe 5.2: Go-Back-N (13 Punkte)

In diesem Aufgabenteil sollen die Methoden `output()`, `input()`, `timerInterrupt()` und `init()` der Sender-Seite (Host A) und die Methoden `input()` und `init()` der Empfänger-Seite (Host B) implementiert werden, die gemeinsam einen unidirektionalen Datentransfer von Host A zu Host B mittels Go-Back-N mit einer Fenstergröße von 8 implementieren.

Es wird **dringend** empfohlen, zuerst den einfacheren Teil der Aufgabe (Alternating-Bit-Protokoll) zu implementieren und Ihre Implementierung dann um das schwierigere Go-Back-N-Protokoll zu erweitern. Bei der Implementierung von Go-Back-N sind jedoch einige zusätzliche Aspekte zu berücksichtigen, die beim Alternating-Bit-Protokoll keine Rolle spielten:

- **output(NWEmuMsg message)**

  Wie aus Vorlesung und Übung bekannt, bedeutet die Fenstergröße von 8, dass bis zu 8 Pakete ohne Erhalt einer Bestätigung gesendet werden können. Die gesendeten Pakete müssen allerdings gepuffert werden, um für etwaige Sendewiederholungen zur Verfügung zu stehen. Für die Pufferung erscheint uns eine `LinkedList` geeignet. Die Größe des Puffers muss dabei nicht beschränkt werden, d.h. sie müssen bereits bestätigte Pakete nicht entfernen. Dennoch muss natürlich beachtet werden, dass die Fenstergröße auf 8 beschränkt ist. Kommt bei bereits vollem Fenster eine weitere Nachricht von der Anwendungsschicht, so wird diese abgewiesen, d.h. `false` zurückgegeben.

  `https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html`

  Initialisieren einer `LinkedList` mit `NWEmuPkt`-Elementen:
  `LinkedList<NWEmuPkt> buffer;`

  Einfügen eines `NWEmuPkt`-Elementes am Ende:
  `buffer.add(packet);`

- **timerInterrupt()**

  Diese Methode wird auf der Sender-Seite aufgerufen, wenn deren Timer abläuft (und damit ein Timer-Interrupt generiert wird). **Sie haben nur einen einzigen Timer pro Seite** und möglicherweise viele noch nicht bestätigte Pakete auf dem Medium, daher sollten Sie sich überlegen, wie Sie mit einem einzigen Timer auskommen können.

  Hinweis: Sollte die Übertragung der Pakete beim Testen Ihrer Implementierung nicht zum Ende kommen bzw. nur sehr langsam voranschreiten, kann dies an einem zu klein gewählten Timeout liegen.

Führen Sie die Emulation solange aus, bis 20 Nachrichten erfolgreich vom Sender zum Empfänger übertragen wurden. Dabei soll eine Verlustwahrscheinlichkeit von 0,2, eine Fehlerwahrscheinlichkeit von 0,2, ein Trace-Wert von 2 und eine mittlere Zeit zwischen zwei Paketankünften von 10 Zeiteinheiten verwendet werden.

Die Verwendung von `NWEmu.jar` wird im folgenden Abschnitt *Hilfreiche Hinweise* beschrieben.

Hilfreiche Hinweise

Verwendung von NWEmu.jar

- Um den vorgegebenen Emulationscode zu verwenden, müssen Ihre Programme die Importanweisung `import fau.cs7.nwemu.*;` enthalten.
- Zum Kompilieren einer eigenen Java-Quelldatei `yourown.java` unter Verwendung der Klassen aus `NWEmu.jar` sollten Sie in das Verzeichnis wechseln, in dem sich Ihr Code befindet, die Datei `NWEmu.jar` ins aktuelle Verzeichnis kopieren und folgende Kommandozeile verwenden:

  ```
  javac -classpath NWEmu.jar:. yourown.java
  ```

- Zum Ausführen Ihres Programms müssen Sie folgende Befehlszeile verwenden:

  ```
  java -classpath NWEmu.jar:. yourown
  ```

  (Unter Windows muss für beide Befehle die Option `-classpath NWEmu.jar;.` verwendet werden.)

Prüfsummenbildung

- Sie können jeden beliebigen Ansatz zur Berechnung der Prüfsumme verwenden. Beachten Sie dabei, dass auch die Sequenznummer und die ACK-Nummer fehlerhaft sein können. Ein Möglichkeit ist eine TCP-ähnliche Prüfsumme, die aus der Summe der Integer-Werte des Sequenz- und ACK-Nummern-Feldes und der Byte-weisen Summe des Nutzdatenfeldes besteht (d.h. alle Bytes wie 8 bit Ganzzahlen betrachten und aufsummieren).
- Beachten Sie, dass Methoden und Variablen nur von einer Seite (Sender oder Empfänger) genutzt werden dürfen, denn in der Realität können die Systeme auch nur über einen Kommunikationskanal und nicht über gemeinsame Variablen kommunizieren.

Sequenz- und ACK-Nummern

- Die Variablen `seqnum` und `acknum` der `NWEmuPkt`-Objekte sind entsprechend ihrer Bezeichnung und des jeweiligen Protokolls zu verwenden. Der Sender setzt die jeweilige `seqnum` des zu übertragenden Pakets und lässt die `acknum` unberührt. Der **Empfänger hingegen setzt in seinen Bestätigungspaketen die `acknum` auf die zu bestätigende Sequenznummer und lässt das `seqnum` Feld unbenutzt**.
- Wir gehen hier vereinfachend davon aus, dass der Wertebereich eines Integers in Java ausreicht, sodass ein etwaiger Überlauf der Sequenz-/ACK-Nummern nicht behandelt werden muss.

Vorgegebene leere Implementierung der Methoden

- Viele der Methoden von `AbstractHost` müssen überladen werden, um ein sinnvolles Ergebnis zu erreichen. Einige der Methoden müssen bei Sender und Empfänger unterschiedlich sein, manche müssen beim Empfänger überhaupt nicht implementiert werden. Ein Beispiel, wie Subklassen von `AbstractHost` gebildet werden und Methoden überladen werden, ist in der Datei *Beispiel.java* (siehe StudOn) aufgezeigt.
- Eine Sammlung der Methoden, die von `AbstractHost` und der Emulationsumgebung zur Verfügung gestellt werden, ist in der Datei *documentation.txt* verfügbar (siehe StudOn).

- Zur Implementierung können folgende vorgegebene Methoden hilfreich sein:
  - Die Methode `currTime()` liefert die aktuelle Emulationszeit zurück. Die Zeit ist nützlich zum Debugging und zur Behandlung des Timer-Interrupts.

- Ebenfalls für Debugging ist die Methode `sysLog(int level, String str)` vorgesehen. Diese Methode gibt `str` auf der Standard-Ausgabe aus, wenn `level` kleiner oder gleich dem `trace`-Wert ist, der bei Aufruf der Methode `NWEmu.emulate()` verwendet wurde.
- Es ist mitunter sinnvoll, den Trace-Wert auf 3 zu setzen und **viele** `sysLog()` in den Code einzubauen.
- Die Methode `randTimer()` von Objekten der Klasse `NWEmu` kann genutzt werden, um die Zufallszahlengeneratoren mit der aktuellen Systemzeit zu initialisieren. Wenn Sie diese Methode vor dem Beginn der Emulation aufrufen, liefert jeder Emulationslauf ein unterschiedliches Ergebnis.

Beginnen Sie mit einer einfachen Implementierung

- Bei der Implementierung der beiden Protokolle ist es sehr hilfreich, **sich an den entsprechenden Statecharts aus den Vorlesungsfolien zu orientieren**. Die jeweiligen Aktionen lassen sich gut auf die hier zu implementierenden Methoden übertragen.
- Setzen Sie die Paketverlust- und die Paketfehlerwahrscheinlichkeit zunächst auf 0 und testen Sie Ihre Implementierung. Eventuell ist es sinnvoll, die Methoden zunächst für fehler- und verlustfreien Fall zu entwickeln. Dann können Sie die Methoden um den Fall erweitern, dass eine der beiden Wahrscheinlichkeiten nicht 0 ist und erst dann auf den Fall, dass beide größer als 0 sind.