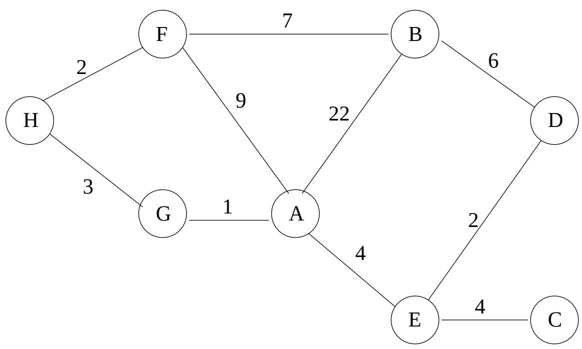
Übung 12 zur Vorlesung
“Vernetzte Systeme”
WS 1999/2000

Prof. Dr. F. Mattern

Ausgabedatum: 26. Jan. 2000
Abgabedatum: 2. Feb. 2000

43. Aufgabe (Source Routing)

(4 Punkte) Im folgenden wird das Beispielnetz aus Aufgabe 42 wiederverwendet. Die Knoten symbolisieren hier Brücken. Knoten C sendet einen *discovery frame* an Knoten H. Geben Sie an, welche Pakete C schliesslich zurück erhält und wie dort jeweils die RD-Felder gesetzt sind.

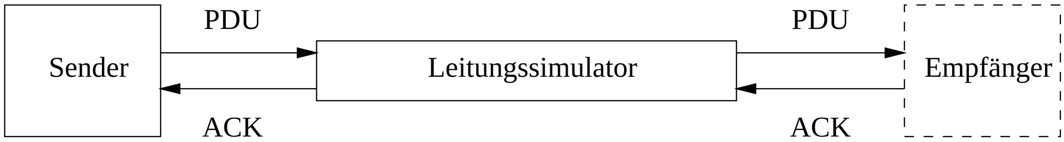


44. Aufgabe (Sliding-Window-Protokoll)

In Aufgabe 36 haben Sie bereits den Empfangspuffer für das Sliding-Window-Protokoll implementiert. Ihre Aufgabe ist es nun, unter Verwendung dieses Puffers die Empfängerseite des Sliding-Window-Protokolls zu implementieren und in einer Simulationsumgebung zu testen.

Simulationsumgebung

Die Simulationsumgebung besteht wie unten skizziert aus einem Empfänger, der die Senderseite des Sliding-Window-Protokolls implementiert sowie einem Leitungssimulator, der zwischen Empfänger und den von Ihnen zu entwickelnden Empfänger geschaltet wird und der bestimmte Eigenschaften der Verbindung (wie Delay und Verlustrate) simulieren kann.



Sender und Leitungssimulator bestehen aus einer Reihe von Java-Klassen, die Sie sich von der bereits bekannten Webseite `http://www.inf.ethz.ch/department/IS/vs/education/WS9900/VS/` herunterladen können. Übersetzen können Sie diese mittels des Kommandos `javac *.java`.

Empfänger

Bevor Sie die Simulationsumgebung nutzen können, müssen Sie zunächst Ihre Empfänger-Klasse in Java implementieren. Dazu können Sie die Klasse `ServerSocket` wie folgt verwenden, um auf eine Verbindung von einem Sender auf Port `<port>` zu warten und einen `Socket` für diese Verbindung zu erzeugen:

```
ServerSocket serverSocket = new ServerSocket(<port>);
Socket clientSocket = serverSocket.accept();
```

Verwenden Sie bei der Lösung der Aufgabe Port 5555, da sich der Leitungssimulator defaultmässig an diesen Port verbindet.

Um die Pakete zu lesen bzw. zu schreiben können Sie (wie bereits aus vorangehenden Programmieraufgaben bekannt), die Klassen `BufferedReader` bzw. `PrintWriter` verwenden:

```
PrintWriter out =
    new PrintWriter(clientSocket.getOutputStream(), true);
BufferedReader in =
    new BufferedReader(
        new InputStreamReader(clientSocket.getInputStream()));
```

Mittels der Methoden `in.readLine()` bzw. `out.println()` können Sie dann ganze Zeilen lesen bzw. schreiben. Der Datenaustausch zwischen Sender und Empfänger erfolgt über jeweils einzelne Text-Zeilen (Strings), welche entweder ein PDU oder ein ACK repräsentieren. Das Format dieser Nachrichten ist im nächsten Abschnitt beschrieben. Statt dieses Format selbst zu erzeugen bzw. zu parsen sollten Sie die existierende Hilfsklasse `Packet` verwenden (wie weiter unten beschrieben), welches es Ihnen erlaubt, am Socket empfangene Strings in PDU's zu übersetzen[1] und zu sendende Paket-Bestätigungen (bzw. deren Nummer) vor der Übertragung in ACK's.

Sobald der Empfänger Pakete aus dem Ringpuffer auslesen kann, geben Sie diese Bitte an STDOUT aus, z.B. mittels

```
System.out.println(data);
```

Protokoll

Es wird das aus der Vorlesung bekannte Sliding-Window-Protokoll verwendet. Vom Sender zum Empfänger werden Datenpakete (PDU's) übertragen, wobei jede PDU aus einem einzeiligen String besteht, der eine fünfstellige Sequenznummer, gefolgt von einem Komma, gefolgt von den Daten enthält. In umgekehrter Richtung werden Bestätigungen (ACK's) übertragen, die lediglich aus einer ebenfalls fünfstelligen Sequenznummer bestehen. Ein Beispiel für ein gültiges Datenpaket ist "`00001,foobarbaz`", ein Beispiel für eine gültige Bestätigung ist "`00132`".

[1] also in eine Paketnummer und den Paketinhalt.

Hilfsklassen

Für die Implementierung des Empfängers stellen wir Ihnen zwei Hilfsklassen zur Verfügung. Zum einen die Klasse `RecvBuffer` (eine Beispiel-Implementierung des Empfangspuffers aus Aufgabe 36, alternativ können Sie auch Ihre eigene Lösung verwenden) und zum anderen die Klasse `Packet` (erleichtert das Parsen und Erzeugen von Protokollnachrichten). Die Schnittstelle der Klasse RecvBuffer kennen Sie bereits aus Aufgabe 36. Die Klasse `Packet` verfügt u.a. über folgenden für die Empfängerseite relevante Methoden:

- `Packet(String inputLine)`
  Konstruktor, der aus einem PDU-String (in dem in Abschnitt "Protokoll" beschriebenen Format) ein `Packet` erzeugt.
- `int id()`
  Liefert die Sequenznummer des Paketes.
- `String data()`
  Liefert die Daten des Paketes.
- `String formatACK(int id)`
  Liefert einen ACK-String mit Sequenznummer `id` gemäss dem im Abschnitt "Protokoll" beschriebenen Format.

In dieser Aufgabe müssen Sie das erste mal die Empfängerseite einer Socket-Verbindung implementieren.

Benutzung der Klassen

Starten Sie zuerst Ihren Empfänger mittels des Kommandos `java SlidingWindowReceiver`[2], dann den Leitungssimulator mittels `java Pipe` in einer zweiten Shell und zum Schluss den Sender mittels `java SlidingWindowSender` in einer dritten Shell.

Der Sender erwartet nun die Eingabe eines Textes über STDIN, welchen er dann versucht, durch den Leitungssimulator hindurch zum Empfänger zu übertragen. Damit Sie nicht selbst ständig Texte eingeben müssen, können sie auf der (Unix-)Kommandozeile den Inhalt existierender Dateien wie folgt an das Sendeprogramm umleiten:

```
cat myTextFile.txt | java SlidingWindowSender
```

Auf der Kommandozeile des Senders können sie optional zusätzliche Parameter angeben, um die Grösse des Sendefenster (`<bufsize>`), den Timeout (`<timeout>`, in Millisekunden) und die maximale Paketgrösse (`<packetSize>`, in Bytes) anzugeben oder den Debugmodus (`debug`) einzustellen. Die vollständige Kommandozeilensyntax des Senders lautet damit:

```
java SlidingWindowSender <bufsize> <timeout> <packetSize> debug
```

Ebenso kann man bei der Pipe die Verlustrate (`<BER>`, Bit-Error-Rate in Fehler pro Bits), das minimale (`<minDelay>`, in Millisekunden) und maximale (`<maxDelay>`, in Millisekunden) Delay über optionale Kommandozeilenparameter festlegen und den Debugmodus (`debug`) einstellen:

```
java Pipe <BER> <minDelay> <maxDelay> debug
```

[2] Den Namen der Klasse, die den Server implementiert, können Sie selbst wählen.

Beim Aufruf dieser beiden Klassen mittels `java Pipe -h` bzw. `java SlidingWindowSender -h` werden die Aufrufparameter sowie ihre voreingestellten Default-Werte angezeigt. Verwenden Sie den Debugmodus zur Fehlersuche.

a) (12 Punkte) Implementieren sie ihren Empfänger wie oben beschrieben und testen Sie ihn mittels folgender Prozedur:

1. Starten Sie Ihren Empfänger und leiten Sie dessen Ausgabe in eine Datei um, z.B.:
   `java SlidingWindowReceiver > OutFile.txt`
2. Starten Sie (in einem 2. Fenster) die Leitungssimulation: `java Pipe`
3. Starten Sie (in einem 3. Fenster) den Sender und leiten Sie den Inhalt einer (hinreichend grossen) Text-Datei[3] dahin um, z.B:
   `cat InFile.txt | java SlidingWindowSender`
4. Nachdem die Datei übertragen wurde, sollten `InFile.txt` und `OutFile.txt` identisch sein.

Senden Sie den Quellcode (`*.java`) ihrer Empfänger-Implementation an Ihren Tutor.

b) (4 Punkte) Bestimmen Sie den Durchsatz[4] für das Senden der Text-Datei[5] `TestFile.txt` mit folgenden Parametern:

- `java Pipe <BER> 5 10 debug`
  Mit BER={3.5, 3.75, 4.0, 5.0}
- `cat InFile.txt | java SlidingWindowSender 10 1000 <PacketSize>`
  Mit PacketSize={10,50,500}

Beobachten Sie die Ausgabe der Leitungssimulation – wenn so gut wie jedes Paket verloren geht, brechen Sie den Sendeprozess ab (ermitteln also 0 Bytes Durchsatz).

[3] eine Beispieldatei findet sich auf der Web-Seite der Vorlesung.
[4] dieser wird am Ende einer Übertragung vom `SlidingWindowSender` ausgegeben.
[5] diese findet sich auf der Web-Seite der Vorlesung.