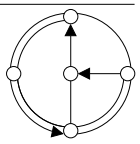
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



SS 2007 Prof. R. Wattenhofer / Michael Kuhn / Remo Meier

Vernetzte Systeme
Lösungsvorschlag 4

1 Vorbemerkung

Die vollständigen Quellcode-Dateien der Lösungsvorschläge können von unserer Homepage heruntergeladen werden. Auf diesem Lösungszettel finden Sie Ausschnitte und ein paar Anmerkungen dazu.

2 Instant Messenger

a) Sie müssen ein `DatagramPacket` erstellen, das über den vorhandenen `DatagramSocket` versendet wird. Im Konstruktor übergeben Sie dem Paket ein Byte-Array, das die zu verschickenden Daten beinhaltet. Zusätzlich spezifizieren Sie, an welche Adresse (ip:port) das Paket gesendet werden soll. Das Zusammenfügen des Byte-Arrays sollte weitestgehend selbsterklärend sein. Um Byte-Arrays zu kopieren, können Sie die von Java bereitgestellte Methode `System.arraycopy` verwenden.

```
// Get some necessary information...
int messageLength = messageText.length();
byte[] message = new byte[3 + messageLength];
int port = user.getPort();
InetAddress ip = InetAddress.getByName(user.showIP());

// Create the message (as byte array)
message[0] = NetworkUtilities.MESSAGE;
NetworkUtilities.put16(messageLength,message,1);
System.arraycopy(messageText.getBytes(),0,message,3,messageLength);

// Create and send the UDP packet
DatagramPacket packet = new DatagramPacket(message,message.length,ip,port);
socket.send(packet);
```

b) Sie müssen ein neues `DatagramPacket` anlegen, dem ein Byte-Array als Puffer für die eingehenden Daten übergeben wird. Mittels der Methode `receive` des bereitgestellten UDP-Sockets empfangen Sie das *komplette* Paket. Die Methode ist blockierend. Die Ausführung des Programms wird daher so lange gestoppt, bis ein Paket eintrifft[1]! Im Unterschied zu einer TCP-Verbindung erhalten Sie also alle gesendeten Daten auf einmal. Danach können Sie aus dem Byte-Array die übermittelten Parameter extrahieren und die Nachricht über die GUI des Instant Messengers anzeigen.

```
/* get the actual packet */
DatagramPacket packet = new DatagramPacket(message,message.length);
socket.receive(packet); // blocking, until packet received!

/* parse the message */
byte type = message[0];
int messageLength = NetworkUtilities.get16(message,1);
String messageText = new String(message,3,messageLength);

/* display it to the user */
User user = InstantMessenger.IM.getUser(packet.getAddress().getAddress(),packet.getPort());
InstantMessenger.IM.displayMessage(user.getName() + ": " + messageText);
```

[1] Mittels der Methode `setSoTimeout` eines `DatagramSockets` (UDP) kann eine Timeout-Zeit eingestellt werden. Nach Ablauf dieser Zeit wirft ein `receive` eine `java.net.SocketTimeoutException`, sofern keine Daten empfangen wurden.

3 Flusssteuerung

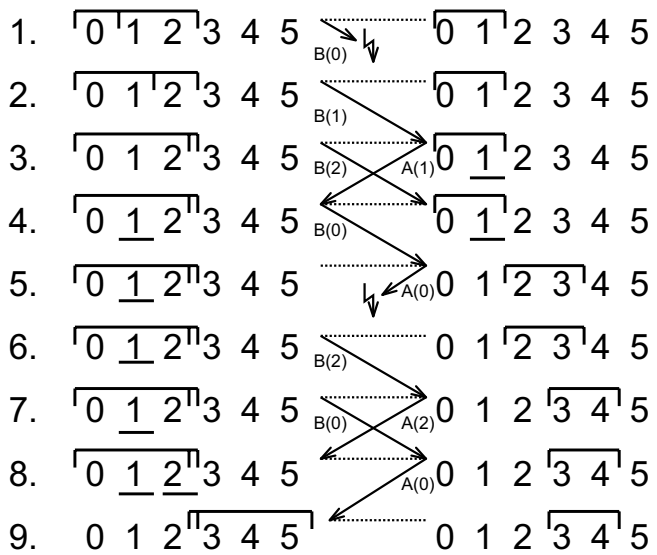
a) Das Produkt aus (Ausbreitungs-)Verzögerung V und Bandbreite R ist die (maximale) Datenmenge, die sich auf dem Weg vom Sender zum Empfänger "in der Leitung" befindet. Nachdem der Sender aufgehört hat zu senden, dauert es noch V, bis das letzte vom Sender geschickte Signal den Empfänger erreicht hat. Wenn keine Daten auf Empfängerseite weggeworfen werden sollen, muss der Puffer so dimensioniert sein, dass er diese nicht mehr zu stoppenden Daten noch aufnehmen kann.

b) Wenn der Empfänger eine nahende Überlast bemerkt und eine Halt-Nachricht abschickt, sind $V \cdot R$ Daten unterwegs. Es dauert mindestens die Verzögerung V, bis die Halt-Nachricht den Sender erreicht (da die Halt-Nachricht sehr klein ist, vernachlässigen wir die Übertragungszeit). In dieser Zeit hat der Sender nochmals $V \cdot R$ neue Daten verschickt. Insgesamt muss der Puffer auf Empfängerseite also mindestens $2 \cdot V \cdot R$ gross sein.

c) Treffen Datenpakete an einer Zwischenstation schneller ein, als diese weiterverarbeitet werden können (in diesem Fall bedeutet weiterverarbeiten das Weiterleiten der Pakete zur nächsten Zwischenstation), müssen Datenpakete weggeworfen werden, oder es muss eine aufwändige Flusssteuerung auch auf den Teilstücken einer Verbindung realisiert werden. Pufferspeicher bieten hier keine prinzipielle Lösung: sie können unterschiedliche Bandbreiten von Teilstrecken einer Verbindung nicht ausgleichen, sondern nur Schwankungen von Übertragungsraten.

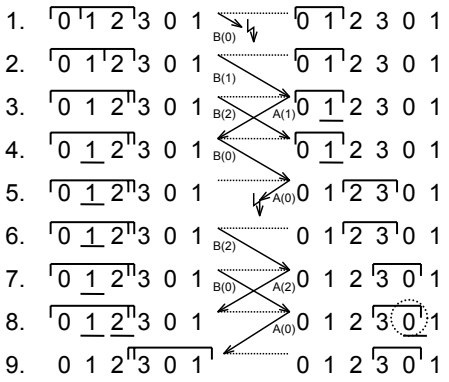
4 Schiebefenster und Sequenznummern

a) Sliding-Window-Protokoll, korrekter Ablauf



Hinweis: Der auf Folie 3/39 beschriebene Algorithmus geht davon aus, dass Sende- und Empfangsfenster gleich gross sind (N). In diesem Beispiel ist dies jedoch nicht gegeben. Hier ist Sendefenstergrösse M=3 und Empfangsfenstergrösse N=2. Entsprechend muss der Algorithmus angepasst werden: Beim Empfänger muss es in Zeile 8 'pkt n in [rcvbase−M,rcvbase−1]' und beim Sender in Zeile 6 'ACK(n) in [sendbase,sendbase+M-1]' heissen. Somit wird in Zeile 8 obiger Abbildung der wiederholte Empfang von B(0) bestätigt.

b) Sliding-Window-Protokoll, Protokollfehler



In Zeile 8 wird hier der zweite Empfang von B(0) nicht als Wiederholung erkannt, sondern als neues Paket aufgefasst. Dies führt dazu, dass das gleiche Paket zweimal empfangen und an die Anwendung ausgeliefert wird. Somit versagt das Protokoll in diesem Fall.

c) Um ein Szenario wie auf Folie 3/41 zu vermeiden, müssen wir verhindern, dass die obere Kante des Empfangsfensters (d.h. mit der "höchsten"[2] Sequenznummer) die Menge der Sequenznummern überspringt und mit der unteren Kante (d.h. mit der "tiefsten" Sequenznummer) des Sendefensters überlappt. Die Menge der Sequenznummern muss also so gross sein, dass das Sende- und das Empfangsfenster ohne zu überlappen hineinpassen. Folglich müssen wir herausfinden, wie gross der mögliche Sequenznummernbereich des Sende- und Empfangsfensters zu einer bestimmten Zeit sein kann.

Nehmen wir an, w sei die Fenstergrösse und m die tiefste Sequenznummer, auf die der Empfänger wartet. In diesem Fall umfasst beim Empfänger das Fenster die Sequenznummern $[m, m + w - 1]$, und er hat das Paket $m - 1$ und die $w - 1$ Pakete davor empfangen (und bestätigt). Wenn der Sender noch keine dieser w Bestätigungen (acknowledgments, ACKs) empfangen hat, dann können Bestätigungen mit den Sequenznummern $[m-w, m-1]$ immer noch unterwegs sein. Wenn der Sender keine dieser Bestätigungen empfängt (weil sie alle verloren gingen), dann wäre das Sendefenster $[m - w, m - 1]$.

Folglich ist die untere Kante des Sendefensters $m - w$ und die obere Kante des Empfangsfensters $m + w - 1$. Damit also die obere Kante des Empfangsfensters nicht mit der unteren Kante des Sendefensters überlappt, muss die Menge der Sequenznummern mindestens $2 \cdot w$ Sequenznummern umfassen, also $k \geq 2 \cdot w$.

[2] Die Begriffe "höchste" und "tiefste" Sequenznummer haben hier eine spezielle Bedeutung. Bedenken Sie, dass die Sequenznummern immer wieder verwendet werden, und stellen Sie sich den Bereich am besten als je einen im Uhrzeigersinn durchnummerierten Kreis beim Sender und Empfänger vor, auf welchem sich die Schiebefenster bewegen. Eine Sequenznummer a ist tiefer als eine Sequenznummer b, wenn a sich im Uhrzeigersinn nach b verschieben lässt, ohne dabei die Bereichsgrenzen zu überschreiten.