



3. Übung zur Vorlesung Rechnernetze (Musterlösung)

Abgabe: 26.11.2004

Aufgabe 1: Implementierung eines einfachen Schicht7-Protokolls des OSI Protokollstacks (5 Punkte)

Im Archiv http://www-vs.informatik.uni-ulm.de/teach/ws04/rn1/OSI_Aufgabe.zip finden Sie die Java-Klasse *Layer4*, welche ein Reihe von Dienstprimitiven der Schicht 4 des ISO/OSI Modells implementiert. Des weiteren finden sie in dem Archiv zwei Java-Programme „Klient“ und „Server“. Der Server hört permanent auf ankommende Nachrichten von Klienten und gibt diese bei ihrem Eintreffen am Bildschirm aus. Der Klient sendet 10 Textnachrichten an den Server und beendet sich dann. Klient und Server benutzen Instanzen der Klasse *Layer4*, um Nachrichten zu versenden bzw. zu empfangen. Den Quellcode für die Programme finden Sie in *Client.java* und *Server.java*. Eine Beschreibung der Klasse *Layer4* finden Sie im Unterverzeichnis *Layer4_doc*.

Wenn Sie die beiden Programme starten (Server zuerst starten), werden Sie anhand der Meldungen auf dem Bildschirm allerdings feststellen, dass nicht alle vom Klienten gesendeten Nachrichten auch beim Server ankommen. Das liegt daran, dass die Klasse *Layer4* einen zufälligen Verlust von Datenpaketen simuliert. Ihre Aufgabe ist es nun, das Problem der Paketverluste auf einer höheren Protokollschicht zu beheben. Dabei sollen Sie folgendermaßen vorgehen:

- Schreiben Sie eine neue Klasse *Layer7*. Diese soll die Dienste von *Layer4* in Anspruch nehmen (Schicht 5 und 6 aus dem OSI-Modell überspringen wir einfach, da wir sie für diese Anwendung nicht benötigen). *Layer7* soll ihrerseits Dienstprimitive für die Programme Klient und Server zur Verfügung stellen. In *Client.java* und *Server.java* sollen Sie dann nur noch auf die Dienstprimitive von *Layer7* zugreifen.
- Um Paketverluste sicher zu erkennen, soll *Layer7* alle Pakete mit einem Header versehen, bevor sie an *Layer4* weitergegeben werden. Der Header soll nur ein einziges Byte mit einer Sequenznummer enthalten. Auf Empfängerseite bestätigt Schicht 7 jedes ankommende Datenpaket (UnitDataIndication) mit einem Acknowledge-Paket (UnitDataResponse). Das Acknowledge-Paket soll nur aus dem Header bestehen und dieselbe Sequenznummer tragen, wie das zu bestätigenden Paket.
- Nachdem die Schicht 7 ein Datenpaket abgeschickt hat, wartet sie maximal 500 Millisekunden lang auf eine Bestätigung. Das nächste Paket schickt sie erst dann ab, wenn sie ein Acknowledge-Paket erhalten hat (Stop-and-Wait), welches dieselbe Sequenznummer enthält wie das zuletzt abgeschickte Datenpaket. Kommt beim Absender des Datenpakets nach 500 Millisekunden noch keine korrekte Bestätigung an, dann wiederholt die Schicht 7 das Absenden dieses Pakets.

Bei jedem ankommenden Acknowledge auf Seite des Klienten sollen Sie eine entsprechende Nachricht mit der erhaltenen Sequenznummer auf dem Bildschirm ausgeben. Sollte auf Seite des Klienten ein Timeout ablaufen, dann geben Sie ebenfalls eine dementsprechende Nachricht auf dem Bildschirm aus! Geben Sie bitte den Quelltext der Klassen *Layer7*, *Client* und *Server* ausgedruckt oder per Email ab! Schicken Sie den gesamten Quelltext gezippt an: schorr@informatik.uni-ulm.de mit dem Betreff „RN1 – Übung 3“.

Lösungsvorschlag:

```
/*  
 * Rechnernetze I, WS2004/2005  
 * Uebungsblatt 3, Aufgabe 1  
 * Musterlösung, File: Layer7.java  
 */  
  
import java.io.*;  
import java.net.*;  
  
public class Layer7  
{  
    private byte    _sequenceNumber = 0;
```

```

private Layer4 _layer4Instance;

public Layer7(boolean client)
{
    _layer4Instance = new Layer4(client);
}

public int SendUnitDataRequest(String ip, byte[] buffer)
{
    // Create a buffer capable of storing the 1-byte header and the payload.
    byte[] packet = new byte[buffer.length+1];
    // Create a buffer capable of storing the incoming acknowledge packet.
    byte[] ack = new byte[1];

    // Set the sequence number of the data packet.
    packet[0] = _sequenceNumber;
    // Copy the payload into the data packet.
    for(int i = 0; i < buffer.length; i++)
    {
        packet[i+1] = buffer[i];
    }

    _layer4Instance.SendUnitDataRequest(ip, packet);

    while ((_layer4Instance.CheckForUnitDataConfirmation(500, ack) == 0) ||
        (ack[0] != _sequenceNumber))
    {
        System.out.println("Timeout occured while waiting for the ack!");
        _layer4Instance.SendUnitDataRequest(ip, packet);
    }

    // Now we can increment the sequence Number.
    _sequenceNumber++;

    return 0;
}

public int CheckForUnitDataIndication(int timeout, byte[] buffer)
{
    // Create a buffer capable of storing the incoming data packet
    // (we simply assume that 1024 bytes is enough!)
    byte packet[] = new byte[1024];
    // Create a buffer capable of storing the acknowledge packet.
    byte ack[] = new byte[1];
    int packetLength;

    // Now we wait for incoming packets.
    packetLength = _layer4Instance.CheckForUnitDataIndication(0, packet);

    // Now check, whether the sequence number is correct.
    while (packet[0] != _sequenceNumber)
    {
        // Although the sequence number was wrong, we still send an acknowledge.
        // Otherwise, the sender would try to transmit the same packet again.
        ack[0] = packet[0];
        _layer4Instance.SendUnitDataResponse(ack);
        System.out.println("- Layer7: Sent acknowledge for packet number: " + ack[0]);

        // Now we wait again for the next packet.
        packetLength = _layer4Instance.CheckForUnitDataIndication(0, packet);
    }

    // Now we construct an acknowledge packet. The acknowledge shall bear the
    // same sequence number as the last packet received.
    ack[0] = _sequenceNumber;
    _sequenceNumber++;

    // Now send the acknowledge packet to the other peer.

```

```

_layer4Instance.SendUnitDataResponse(ack);
System.out.println("- Layer7: Sent acknowledge for packet number: " + ack[0]);

for(int i = 1; i < packetLength; i++)
{
    buffer[i-1] = packet[i];
}

return packetLength - 1;
}
}

```

Aufgabe 2: Prüfpolynome und CRC-Berechnung (3 Punkte, 1 + 1 + 1)

- Eine Station empfängt die Bitsequenz 1100110101110. Überprüfen Sie, ob diese Bitsequenz mit hoher Wahrscheinlichkeit fehlerfrei übertragen wurde. Nehmen Sie dabei an, dass die Prüfsumme mittels CRC und dem Generatorpolynom $G(x) = x^3 + x + 1$ berechnet wurde (vollständige Rechnung angeben)!
- Gehen Sie nun einmal davon aus, dass im Falle von Teilaufgabe a) keine Übertragungsfehler aufgetreten sind. Wie hat dann die tatsächlich zu übertragende Nachricht ausgesehen, und zu welchem Zweck wurden an diese Nachricht noch eine bestimmte Anzahl Bits angehängt?
- Wie müsste ein Übertragungsfehler beschaffen sein, damit er durch das CRC-Verfahren nicht erkannt wird? Gehen Sie davon aus, dass die Bitsequenz aus Teilaufgabe a) korrekt ist. Verändern sie ihn so, dass der Empfänger Ihren eingebauten Fehler nicht erkennt!

a) Lösungsvorschlag:

1	1	0	0	1	1	0	1	0	1	1	1	0
1	0		1									
0	1	1	1	1								
	1	0	1	1								
	0	1	0	0	1							
		1	0	1	1							
		0	0	1	0	0						
			0	0	0	0						
			0	1	0	0	1					
				1	0	1	1					
				0	0	1	0	0				
					0	0	0	0				
					0	1	0	0	1			
						1	0	1	1			
						0	0	1	0	1		
							0	0	0	0		
							0	1	0	1	1	
								1	0	1	1	
								0	0	0	0	0
									0	0	0	0
									0	0	0	0

→ Der Bitstring wurde fehlerfrei übertragen!

b) Lösungsvorschlag:

Es wurden 3 Bits angehängt, da der Grad des Generatorpolynoms 3 ist. Die tatsächliche Nachricht war also: 1100110101. Die 3 zusätzlichen Bits wurden angehängt, um die zusammengesetzte Zahl nach Anwendung des CRC-Verfahrens auf jeden Fall durch das Generatorpolynom teilbar zu machen ohne dabei die Originalnachricht zu verändern.

c) Lösungsvorschlag:

Das den umgeklappten Bits entsprechende Fehlerpolynom müsste durch das Generatorpolynom teilbar sein. Dann käme bei der Division wie im fehlerfreien Fall ein Rest von 0 heraus. Das wäre z.B. der Fall, wenn wir das Generatorpolynom selber als Fehlerpolynom verwenden. Durch Umklappen von Bit 0, 1 und 3 (von rechts) erhalten wir: 1100110100101. Wir überprüfen mittels Division, ob wir den Fehler erkennen würden:

1	1	0	0	1	1	0	1	0	0	1	0	1			
1	0	1	1												
0	1	1	1	1											
	1	0	1	1											
	0	1	0	0	1										
		1	0	1	1										
		0	0	1	0	0									
			0	0	0	0									
			0	1	0	0	1								
				1	0	1	1								
				0	0	1	0	0							
					0	0	0	0							
						0	1	0	0	0					
							1	0	1	1					
							0	0	1	1	1				
								0	0	0	0				
									0	1	1	0			
										1	0	1			
											0	1			
												1			
													1		
														0	
															0

→ Nein, der Fehler würde in diesem Fall nicht erkannt werden.

Aufgabe 3: Effizienz von Stop-And-Wait – ohne Übertragungsfehler (2 Punkte)

In der Übung haben wir eine Formel zur Berechnung der Effizienz von Stop-and-Wait kennen gelernt. Ein Kanal besitzt eine Datenrate von 56kbps und eine Signallaufzeit von 25ms. Für welche Paketgrößen ergibt sich eine Effizienz des Stop-and-Wait Protokolls von mindestens 80%? (Die Länge der Acknowledgements sei vernachlässigbar und es treten keine Paketfehler auf).

Lösungsvorschlag:

Sei L die Paketlänge

$$U = \frac{TRANS_P}{TRANS_P + 2 * PROP} \quad TRANS_P = \frac{L}{56000}$$

$$0,8 \leq \frac{\frac{L}{56000}}{\frac{L}{56000} + 2 * 0,025} \Leftrightarrow 0,8 * (\frac{L}{56000} + 0,05) \leq \frac{L}{56000} \Leftrightarrow 0,04 \leq 0,2 * \frac{L}{56000} \Leftrightarrow \frac{0,04 * 56000}{0,2} \leq L$$

$$\Leftrightarrow L \geq 11200$$

Bei einer Paketlänge von mindestens 11.200 Bit erreichen wir eine Effizienz von 80%.