
Technische Informatik III:
Betriebssysteme und Rechnernetze WS 2007/08
Musterlösung zum Übungsblatt Nr. 7

Aufgabe 1: Begriffe

7 Punkte

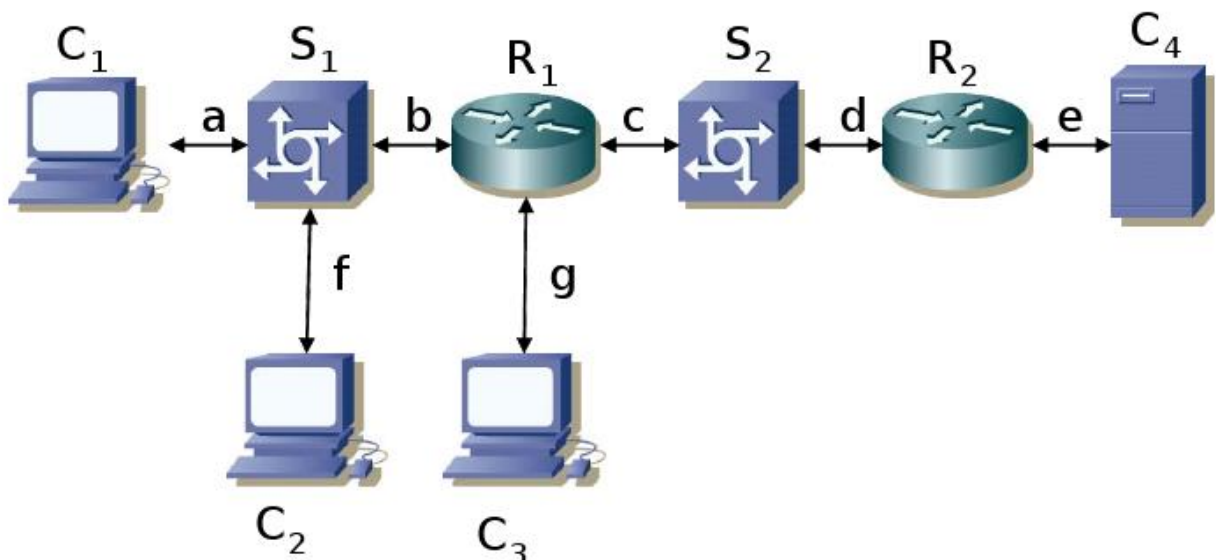
Beschreiben Sie jeden der folgenden Begriffe durch maximal zwei Sätze: Port, Cipher, Public Key Infrastructure, Bastion Host, Proxy, NAT, IPsec

- Port, zusammen mit der IP-Adresse die Adressierung im TCP und UDP-Protokoll, damit können direkt Prozesse adressiert werden.
- Cipher bezeichnet Verschlüsselungsverfahren, die Inhalte von Nachrichten verschleiern.
- Public Key Infrastructure bezeichnet ein System im welchen digitale Zertifikate erstellt, verteilt und überprüft werden können um Entitäten zu authentifizieren.
- Bastion Host ist ein Server, der an der Schnittstelle im öffentlichen Netz steht (Mailserver, Firewall, etc.) und deswegen besonderen Schutz braucht.
- Proxy, Vermittlungsschnittstelle, die für interne Klienten mit externen Servern kommuniziert.
- NAT, Network Address Translation, ist ein Verfahren um Adressen (IP-Adresse und Port) bei den Übergang in andere Netze auszutauschen.
- IPsec, Internet Protocol Security, ist ein Verschlüsselungsprotokoll, das auf Schicht 3 arbeitet.

Aufgabe 2: Zwischensysteme

9 Punkte

Für die gesamte Aufgabe soll von folgendem einfachen Firmennetz ausgegangen werden:



Alle Verbindungen a, b, ..., g sollen vom Typ 100 Mbit/s-Ethernet sein. Die MAC-Adressen wurden vereinfacht, um unnötiges Schreiben zu vermeiden. Ansonsten wird eine ganz normale Internet-Protokollwelt mit IPv4 angenommen. Switches arbeiten wie transparente Brücken und können noch mehr als die gezeigten Ports besitzen. Ebenso wurden bei den Routern nur die für die Aufgabe benötigten Ports gezeigt. C₃ sei allen Komponenten als DNS-Server bekannt, ebenso die Default-Gateways. Ein Management-Netz wurde der Übersicht halber weggelassen.

1. Welche Adressklassen gibt es im Internet und welche davon werden hier verwendet? Welche Adressbereiche werden für das öffentliche und welche für das private Netz eingesetzt? (3 Punkte)

Für das IP-Protokoll gelten folgende Klassen:

- Class A: 1.0.0.0 – 127.255.255.255
- Class B: 128.0.0.0 – 191.255.255.255
- Class C: 192.0.0.0 – 223.255.255.255
- Class D: 224.0.0.0 – 239.255.255.255
- Class E: 240.0.0.0 – 255.255.255.255

Im diesem Firmnetzwerk wird für die Switches und Router (10.1.1.* – privates Netz) die Klasse A verwendet und für die Klienten (129.13.*.* – öffentliches Netz) die Klasse B.

2. Gehen Sie davon aus, dass noch kein Rechner C_i mit einem anderen Rechner kommuniziert hat. Ein Anwender klickt im Web-Browser auf C_1 auf einen URL, welcher auf eine Datei verweist, die auf C_4 gespeichert sei. Welche Art Pakete sind die ersten, die jeweils über Teilstrecke a , b und e übertragen werden? Warum werden diese Pakete benötigt? (4 Punkte)

Über die Teilstrecken a und b wird zuerst ein DNS-Paket laufen, da die URI zuerst aufgelöst werden muss und der registrierte DNS-Server C_3 ist und an R_1 hängt. Nachdem die URL aufgelöst worden ist, wird dann eine HTTP-Anfrage direkt an C_4 gesendet wird, das dann auch das erste Paket ist, das über e läuft.

3. Nach jahrelangem Dienst fällt S_1 aus. Der Switch wird umgehend durch einen neuen ersetzt und die angeschlossenen Komponenten sollen möglichst nichts davon bemerken. Nach Einschalten des neuen S_1 sendet C_2 , wie gewöhnlich und unabhängig von der Existenz eines neuen Switches, eine DNS-Anfrage an C_3 . Geben Sie den jeweiligen Paketkopf (nur Quell-/Zieladresse) auf Schicht 2 von Teilstrecken a , b , c , f und g an. (2 Punkte)

Da der Switch neu im Netzwerk ist, hat er noch keine Einträge in seiner Source-Address-Table (SAT), also sendet er das ankommende Paket auf allen Ausgangsleitungen weiter. Weiterhin trägt er die MAC-Adresse von C_2 in seine SAT ein, so dass er das nächste Mal, wenn ein Paket für C_2 kommt, weiß auf welche Leitung es weiter gesendet werden muss.

| Teilstrecke | Paket | |
|-------------|---|------|
| | Von | Bis |
| a | 1F3D | 3A24 |
| b | 1F3D | 3A24 |
| c | Kein Paket, da R_1 die DNS-Anfrage nur über Port g weiterleitet | |
| f | 1F3D | 3A24 |
| g | 7F29 | 4E37 |

Aufgabe 3: Sicherheitsziele

8 Punkte

Weisen Sie den Sicherheitszielen Vertraulichkeit, Erhalt der Datenintegrität, Verantwortlichkeit und Serviceerreichbarkeit jeweils einen typischen Angriff zu, der versucht diese zu unterlaufen. Erläutern sie diese in maximal zwei Sätzen.

Vertraulichkeit: Durch den *Man-In-The-Middle*-Angriff kann ein Angreifer eine Kommunikation mitlauschen indem er logisch in der Verbindung sitzt und kann so Passwörter, etc mitlauschen. Er vermittelt den Kommunikationspartnern den Eindruck, dass er jeweils der andere Kommunikationspartner ist und leitet die Anfragen weiter an den echten Kommunikationspartner weiter. Siehe auch *Pufferüberlauf* und *Spoofing*.

Datenintegrität: Der Angreifer versucht einen *Pufferüberlauf* zu erzeugen, der durch fehlende Überprüfung beim Dienst auftreten kann und überschreibt so Daten im Speicher des Dienstes und kann eventuell auch eigene Programme starten. Siehe auch *Man-In-The-Middle*-Angriff und *Spoofing*.

Verantwortlichkeit: Beim *Spoofing* werden die Adressen von Daten-Paketeten so gefälscht, dass der Empfänger denkt, sie kommen von einem vertrauenswürdigen Host, so dass der Angreifer Daten unter dem Namen vom vertrauenswürdigen Host senden kann. Siehe auch *Man-In-The-Middle*-Angriff und *Pufferüberlauf*.

Serviceerreichbarkeit: Ein *Denial of Service*-Angriff untergräbt die Serviceerreichbarkeit indem er den angegriffenen Dienst überlastet, so dass der Dienst seine Tätigkeit nicht mehr wahrnehmen kann. Wenn der Angriff von vielen verschiedenen Rechnern erfolgt, spricht man von einem *verteilten Denial of Service* (DDoS).

Aufgabe 4: Fifty/Fifty

8 Punkte

Entscheiden Sie, ob folgenden Aussagen zutreffend oder nicht zutreffend sind, und begründen Sie Ihre Entscheidung:

- **Eine Denial-of-Service-Attacke ist schon dann gegeben, wenn absichtlich so viele erlaubte Anfragen an einen Server gestellt werden, dass die Anfragen anderer Benutzer erst nach langer Zeit ausgeführt werden können.**
Richtig, da sich die anderen Benutzer den Services unter Umständen dann schon nicht richtig verwenden können.
- **Mit dem Aufkommen von asymmetrischen Codes sind symmetrische Codes überflüssig geworden, da diese einen vorherigen Schlüsselaustausch benötigen.**
Falsch, da symmetrische Schlüssel einen großen Geschwindigkeitsvorteil bieten. Außerdem kann man den Schlüsselaustausch mit einem asymmetrischen Verfahren bewerkstelligen.
- **UDP wird für Multimedia-Dienste mit Echtzeit-Anforderungen eingesetzt.**
Richtig, da UDP einen Geschwindigkeitsvorteil bietet, da es verbindungslos ist.

Aufgabe 5: Port-Scanner

8 Punkte

Implementieren Sie in C einen Port-Scanner, der als Kommandozeilenparameter einen Rechnernamen und zwei Portnummern erhält. Das Programm soll nun versuchen, zu allen Ports, die sich in dem von den zwei weiteren Parametern gegebenen Intervall befinden, eine Verbindung aufzubauen. Die Ausgabe des Port-Scanners ist eine Liste der offenen Ports und die jeweilige Ausgabe des Servers auf diesem Port, falls es denn eine gibt. Welche Rückschlüsse über den gescannten Rechner lassen sich aus einem Scan der untersten 1024 Ports ziehen?

Für die ersten 1024 Ports sind spezielle Services definiert und lassen Serverfunktionen eines Rechners ziehen.

Testlauf:

```
chris>./port_scanner localhost 1 1024
Scan host localhost:
Port 22 open: SSH-2.0-OpenSSH_4.6p1 Debian-5ubuntu0.1

Port 111 open: No answer
Port 139 open: No answer
Port 445 open: No answer
Port 631 open: No answer
chris>./port_scanner 192.168.0.1 1 1024
Scan host 192.168.0.1:
Port 23 open: ??????!????
Port 80 open: No answer
```

port_scanner.c basicstyle

```
1  /**
2  * @file port_scanner.c
3  *
4  * TI III: Betriebssysteme und Rechnernetze
5  * WS 2007/08
6  * Übungsblatt Nr. 7,
7  * Aufgabe 5: Port-Scanner
8  *
9  * port_scanner
10 *
11 *
12 * @author Christian Grümme
13 * @see http://cst.mi.fu-berlin.de/teaching/WS0708/19513-V-TI-III/index.html
```

```

14 * @date 2008-01-28
15 */
16 #include <stdlib.h>
17 #include <stdio.h>
18 #include <unistd.h> /* Für close(socket) */
19 #include <sys/types.h> /* Allgemeine Systemtypen z.B. ssize_t*/
20 #include <sys/socket.h> /* Für die Sockets */
21 #include <netdb.h> /* Für gethostbyname() */
22 #include <netinet/in.h> /* Für Strukturen z.B. sockaddr_in */
23 #include <arpa/inet.h> /* Funktionen mit Network Byte Order */
24 #include <sys/time.h> /* Für struct timeval */
25
26 #define BUFFER_SIZE 1024 /* Größe des Lesepuffers */
27 #define TIMEOUT 2 /* Timeout in Sekunden */
28
29 /** @fn int my_connect(struct sockaddr_in server, int port, int* sock
30 )
31 * Verbindet übergebenen Socket mit dem übergebenen Server
32 * und dem übergebenen Port. Wenn es nicht funktioniert hat,
33 * gibt diese Funktion -1 zurück.
34 *
35 * @param server Der zu verbindene Server
36 * @param port Der zu verbindende Port
37 * @param sock Pointer auf einen Socket
38 * @return -1 bei Fehler
39 */
40 int my_connect(struct sockaddr_in server, int port, int* sock)
41 {
42     int is_connected; /* Ob Socket verbunden */
43
44     /* Konvertiere Port nach Network byte order */
45     server.sin_port = htons((unsigned short int) port);
46
47     /* Verbinde mit dem Server */
48     is_connected = connect(*sock, (struct sockaddr*) &server, sizeof(
49         server));
50
51     return is_connected;
52 }
53
54 /** @fn int my_receive(int sock)
55 * Diese Funktion empfängt Daten von einem bereitsverbundenen Socket
56 * und gibt diese auf stdout aus.
57 *
58 * @param sock Der verbundene Socket
59 * @return -1 bei einem Fehler, sonst 0
60 */
61 int my_receive(int sock)
62 {
63     char answer[BUFFER_SIZE]; /* Puffer für die Antwort */
64     ssize_t bytes; /* Anzahl empfangener Bytes */
65     struct timeval tv; /* Struktur für die Zeit */
66     int retval; /* Rückgabewert von select() */
67     fd_set rfd; /* Menge von File Descriptoren */
68
69     /* Setze Timeout */
70     tv.tv_sec = TIMEOUT;
71     tv.tv_usec = 0;

```

```

70
71  /* Inizialisiere Menge */
72  FD_ZERO(&rfd);
73  /* Füge Socket in die Menge */
74  FD_SET(sock, &rfd);
75
76  /* Warte, bis es timeout oder empfangene Daten */
77  retval = select((sock + 1), &rfd, NULL, NULL, &tv);
78
79  if(retval > 0)
80  {
81      /* Empfange */
82      bytes = recv(sock, answer, BUFFER_SIZE - 2, 0);
83      if(bytes == -1)
84      {
85          return -1;
86      }
87
88      /* Terminiere String */
89      answer[bytes] = '\0';
90      /* Antwortausgabe */
91      printf("%s\n", answer);
92
93      return 0;
94  }
95  else
96  {
97      return -1;
98  }
99  }
100
101  /** @fn int init(char* url, char* s_port, char* e_port)
102  * Diese Funktion überprüft die Eingaben und versucht sich
103  * mit der übergebenen URL mit jedem Port zu verbinden.
104  *
105  * @param url Die zu verbindene URL
106  * @param s_port Anfang der zu scannenen Ports
107  * @param e_sock Ende der zu scannenen Ports
108  * @return Fehlercode
109  */
110  int init(char* url, char* s_port, char* e_port)
111  {
112      int start_port, end_port, is_conn, i, sock;
113      struct sockaddr_in server = {0}; /* Komplette Serveradresse, auf 0
114          */
115      struct hostent *host; /* Für die Auflösung von DNS */
116      struct in_addr address; /* IP */
117
118      start_port = atoi(s_port);
119
120      /* Überprüfe Startport */
121      if(start_port < 1 || start_port > 65535)
122      {
123          fprintf(stderr, "Error: Begin port invalid: %s\n", s_port);
124
125          return EXIT_FAILURE;
126      }

```

```

127 end_port = atoi(e_port);
128
129 /* Überprüfe Endport */
130 if(start_port < 1 || start_port > 65535)
131 {
132     fprintf(stderr, "Error: End port invalid: %s\n", e_port);
133
134     return EXIT_FAILURE;
135 }
136
137 /* Tausche, wenn die Reihenfolge nicht stimmt */
138 if( start_port > end_port)
139 {
140     i = end_port;
141     end_port = start_port;
142     start_port = i;
143 }
144
145 /* Löse Namen in IP auf */
146 host = gethostbyname(url);
147 if(host == NULL)
148 {
149     fprintf(stderr, "Error: Cannot resolve host %s\n", url);
150
151     return EXIT_FAILURE;
152 }
153
154 /* Nehme IP aus dem DNS */
155 address = *(struct in_addr*) host->h_addr;
156 /* Setze IP für den Server */
157 server.sin_addr = address;
158 /* Setze Protokollfamilie */
159 server.sin_family = AF_INET;
160
161 printf("Scan host %s:\n", url);
162
163 /* Durchlaufe Ports */
164 for(i = start_port; i <= end_port; ++i)
165 {
166     /* Initalisiere Socket */
167     sock = socket(AF_INET, SOCK_STREAM, 0);
168
169     if(sock == -1)
170     {
171         fprintf(stderr, "Error: Create socket\n", url);
172         return EXIT_FAILURE;
173     }
174
175     /* Versuche aud Port i zu verbinden */
176     is_conn = my_connect(server, i, &sock);
177     if(is_conn != -1)
178     {
179         printf("Port %d ", i);
180         printf("open: ");
181         /* Überprüfe die Antwort */
182         if (my_receive(sock) == -1)
183         {
184             printf("No answer\n");

```

```

185     }
186 }
187
188     close(sock);
189 }
190
191     return EXIT_SUCCESS;
192 }
193
194 /** @fn int main(int argc, char *argv[])
195 * Main-Funktion, Einstieg in das Programm.
196 * 1. Argument Host
197 * 2. Argument Start Port
198 * 3. Argument End Port
199 *
200 * @param argc Anzahl der Argumente
201 * @param argv Host, Start, Ende
202 * @return Status der Terminierung
203 */
204 int main(int argc, char *argv[])
205 {
206     int sock;        /* Der Socket */
207     int is_connected;
208
209
210     /* Überprüfe richtige Anzahl der Argumente */
211     if(argc != 4)
212     {
213         fprintf(stderr, "usage: %s <host> <begin port> <begin port>\n",
214             argv[0]);
215         return EXIT_SUCCESS;
216     }
217
218     return init(argv[1], argv[2], argv[3]);
219 }

```

Aufgabe 6: IP-Stack

12 Punkte

Der auf dem 6. Übungsblatt in Aufgabe 5 verwendete Linux-Kernel kann keine Netzwerkverbindung zur Außenwelt herstellen, weil der Emulator in der verwendeten einfachen Konfiguration keine Netzwerkunterstützung anbietet. Den- noch bietet er Einblicke in die Implementierung eines TCP/IP Stacks.

1. Welche Ausgabe erhalten Sie verschiedene IP-Adressen mit dem Befehl *ping* auf Erreichbarkeit testen? Testen Sie das Verhalten mit der Loopback-Adresse, der eigenen IP-Adresse (zu ermitteln mit dem Befehl *ifconfig*), einer IP-Adresse in demselben Subnetz und einer IP-Adresse außerhalb des Subnetzes. (2 Punkte)

Das Loopbackdevice mit allen Adressen des Netzes 127.*.* verbunden und die virtuelle Ethernet-Karte *eth0* ist mit der Adresse 10.0.2.15 verbunden. Auf diese Adressen reagiert der *ping*, alle anderen Adressen nicht:

```

tiiii:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2500 (2.4 KiB)  TX bytes:1329 (1.2 KiB)

```

Interrupt:10 Base address:0x1000

```
lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:0
        RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

```
tiiii:~# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=2.19 ms
```

```
--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 5ms
rtt min/avg/max/mdev = 2.199/2.199/2.199/0.000 ms
```

```
tiiii:~# ping -c 1 127.0.0.2
PING 127.0.0.2 (127.0.0.2) 56(84) bytes of data.
64 bytes from 127.0.0.2: icmp_seq=1 ttl=64 time=0.569 ms
```

```
--- 127.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.569/0.569/0.569/0.000 ms
```

```
tiiii:~# ping -c 1 127.0.0.27
PING 127.0.0.27 (127.0.0.27) 56(84) bytes of data.
64 bytes from 127.0.0.27: icmp_seq=1 ttl=64 time=0.348 ms
```

```
--- 127.0.0.27 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.348/0.348/0.348/0.000 ms
```

```
tiiii:~# ping -c 1 127.5.4.7
PING 127.5.4.7 (127.5.4.7) 56(84) bytes of data.
64 bytes from 127.5.4.7: icmp_seq=1 ttl=64 time=0.590 ms
```

```
--- 127.5.4.7 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.590/0.590/0.590/0.000 ms
```

```
tiiii:~# ping -c 1 127.0.4.7
PING 127.0.4.7 (127.0.4.7) 56(84) bytes of data.
64 bytes from 127.0.4.7: icmp_seq=1 ttl=64 time=0.364 ms
```

```
--- 127.0.4.7 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.364/0.364/0.364/0.000 ms
```

```
tiiii:~# ping -c 1 126.0.4.7
PING 126.0.4.7 (126.0.4.7) 56(84) bytes of data.
```

```
--- 126.0.4.7 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

```
tiiii:~# ping -c 1 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.391 ms
```

```
--- 10.0.2.15 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
```

```
rtt min/avg/max/mdev = 0.391/0.391/0.391/0.000 ms
tiiii:~# ping -c 1 10.0.2.14
PING 10.0.2.14 (10.0.2.14) 56(84) bytes of data.
From 10.0.2.15 icmp_seq=1 Destination Host Unreachable
```

```
--- 10.0.2.14 ping statistics ---
```

```
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms
```

```
tiiii:~# ping -c 1 10.1.0.14
```

```
PING 10.1.0.14 (10.1.0.14) 56(84) bytes of data.
```

```
--- 10.1.0.14 ping statistics ---
```

```
1 packets transmitted, 0 received, 100% packet loss, time 4ms
```

```
tiiii:~#
```

2. An welcher Stelle im Kernel (Implementierung von TCP/IPv4 in net/ipv4/) werden die jeweiligen Fehlerzustände erkannt bzw. an welcher Stelle wird auf ein Timeout gewartet? Ordnen Sie die jeweiligen Stellen in der Implementierung dem ISO/OSI-Schichtenmodell zu. (4 Punkte)

[...]

3. Gegen Sie an den jeweiligen Schnittstellen zwischen den beteiligten Schichten die kompletten, von der jeweils unteren Schicht zu übertragenden Pakete im Hexadezimalformat auf der Console aus. Identifizieren Sie jeweils für ein Paket pro Schnittstelle die Felder im Paketkopf. (6 Punkte)

[...]