Lehrstuhl für Betriebssysteme
Fakultät für Informatik
Technische Universität München

Grundlagen Rechnernetze und Verteilte Systeme

IN0010, SoSe 2018

Übungsblatt 5

14. Mai – 18. Mai 2018

Hinweis: Mit * gekennzeichnete Teilaufgaben sind ohne Lösung vorhergehender Teilaufgaben lösbar.

Aufgabe 1 Medienzugriffsverfahren

a)* Erläutern Sie kurz das Prinzip von *ALOHA*.

Eine Station sendet, sobald Daten anliegen. Übertragungen werden Out-of-Band bestätigt (andere Frequenz).

b) Wie werden Kollisionen in *ALOHA* erkannt?

Nicht direkt, sondern über das ausbleiben der Out-of-Band Bestätigung.

c) Erläutern Sie kurz das Prinzip von ***Slotted** ALOHA*.

Station beginnt im nächsten Zeitslot zu senden, ungeachtet dessen, ob bereits eine Übertragung stattfindet.

d) Worin besteht der Vorteil von *Slotted ALOHA* gegenüber normalem *ALOHA*?

Durch die Aufteilung in Zeitslots verringert sich die Wahrscheinlichkeit für Kollisionen, da Stationen nicht mehr zu jedem beliebigen Zeitpunkt mit einer Übertragung beginnen können.

Sofern die Zeitslots der Übertragungsdauer einer kompletten Nachricht entsprechen und die Knoten ausreichend genau miteinander synchronisiert sind (was bei derart langen Zeitslots möglich ist), tritt eine Kollision entweder zu Beginn eines Zeitslots auf oder es ist garantiert, dass höchstens eine Stations sendet. (siehe Vorlesung)

e)* Erläutern Sie kurz das Prinzip von *CSMA*.

Medium wird vor dem Senden abgehört. Wenn das Medium im aktuellen Zeitslot frei ist, kann im nächsten begonnen werden zu senden.

f) Erläutern Sie kurz, welche Ergänzungen *CSMA/**CD*** gegenüber reinem *CSMA* hat.

Kollisionen werden erkannt und betroffene Rahmen erneut übertragen.

g) Wie werden erfolgreiche Übertragungen bei *CSMA/CD* erkannt?

Eine Übertragung wird als erfolgreich angenommen, wenn während der Übertragung keine Kollision festgestellt wurde bzw. kein JAM-Signal empfangen wurde.

h) Erläutern Sie kurz, welche Ergänzungen *CSMA/**CA*** gegenüber reinem *CSMA* hat.

Kollisionen können i. A. nicht erkannt werden. Stattdessen wird deren Auftrittswahrscheinlichkeit durch Randomisierung des Sendebeginns verringert.

i)* Was versteht man unter *Binary Exponential Backoff*?

CSMA (sowohl CD als auch CA) warten nach einer Kollision bzw. nicht erfolgreichen Übertragung eine zufällige Anzahl an Slotzeiten. Diese Anzahl wird aus dem Backoff-Window zufällig und gleichverteilt gezogen. Mit jeder Kollision bzw. nicht erfolgreichen Übertraung wird dieses Fenster verdoppelt (binary exponential) bis ein bestimmter Maximalwert erreicht ist. Nach einer erfolgreichen Übertragung wird das Fenster zurückgesetzt.

Aufgabe 2 Bit-Stuffing

In der Vorlesung wurden Verfahren zum Erkennen von Rahmengrenzen vorgestellt. Bei FastEthernet beispielsweise werden Rahmengrenzen mit Hilfe von Steuerzeichen erkannt, was durch die Verwendung der 4B5B-Kodierung möglich wird. Dabei stellt der 4B5B-Code bereits sicher, dass diese Steuerzeichen nie zufällig

durch die Konkatenation von Codewörtern auftreten können.

Im Folgenden wollen wir untersuchen, wie das *High Level Data Link Control (HDLC) Protocol* diesem Problem begegnet. Bei HDLC handelt es sich um ein alternatives Schicht-2 Protokoll, welches beispielsweise auf seriellen Schnittstellen zwischen Cisco-Routern eingesetzt wird. Ein Beispiel für einen HDLC-Rahmen ist in Abbildung 1 dargestellt (vgl. Ethernet-Rahmen in den Vorlesungsfolien).

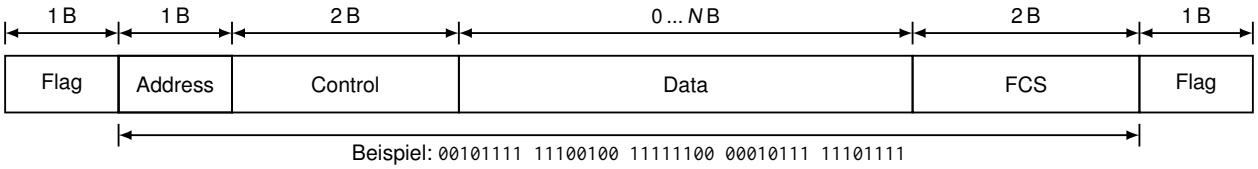


Abbildung 1: Aufbau eines HDLC-Rahmens

HDLC sieht am Anfang und Ende eines jeden Rahmens ein spezielles Begrenzungsfeld (Flag) mit dem Wert 0x7E vor. Es ist also nicht auf die Verwendung zusätzlicher Codes zur Erkennung von Rahmengrenzen angewiesen. Allerdings muss nun sichergestellt werden, dass das Begrenzungsfeld nicht zufällig in einem Rahmen auftaucht. Alle Headerfelder seien von konstanter[1] Länge. Die Länge der Nutzdaten sei stets ein ganzzahliges[2] Vielfaches von 8 bit.

a)* In Abbildung 1 ist eine beispielhafte Bitsequenz für den Rahmen ohne Begrenzungsfelder abgebildet. Diese soll nun mittels HDLC übertragen werden. Um zu verhindern, dass das Begrenzungsfeld 0x7E in den Daten auftaucht, soll Bit-Stuffing verwendet werden. Überlegen Sie sich ein möglichst effizientes Verfahren. Beschreiben Sie dieses in Worten.

Nach jeweils fünf konsekutiven Einsen wird eine Null eingefügt. Dies kann vom Empfänger leicht rückgängig gemacht werden, indem er nach jeweils fünf konsekutiven Einsen die darauf folgende Null wieder entfernt.

Würde erst nach sechs Einsen eine Null eingefügt, so würde die Bitfolge `0111111` nach dem Stuffing genau das Begrenzungsfeld ergeben. Bereits nach vier oder weniger Einsen zu stopfen erhält zwar die Codetransparenz, ist jedoch weniger effizient.

Anmerkung: Warum nicht nach jeder sechsten Eins eine Siebte stopfen? Aus Sicht der Codetransparenz spricht nichts dagegen. Für HDLC wurde aber die oben beschriebene Variante gewählt. Der einzige Vorteil der HDLC-Version besteht in meinen Augen darin, dass lange Eins-Folgen verhindert werden. Je nach Leitungscode kann dies die Taktrückgewinnung ermöglichen, z. B. bei Einsatz einer „inversen“ MLT-3 Kodierung, bei der eine logische Null durch einen Pegelwechsel und eine logische Eins durch einen ausbleibenden Pegelwechsel dargestellt wird. Bei anderen Leitungscodes, wie z. B. NRZ, wird die Taktrückgewinnung nicht ermöglicht, da nach wie vor beliebig lange Nullfolgen auftreten können.

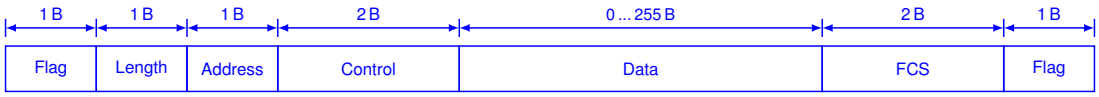
b) Geben Sie die gesamte zu übertragende Bitsequenz gemäß dem in Teilaufgabe a) entwickelten Verfahren an. Markieren Sie die eingefügten Bitstellen.

`01111110 00101111101100100111111010000010111111010111 01111110`

c)* Modifizieren Sie den in Abbildung 1 dargestellten HDLC-Header, so dass anstelle von Bit-Stuffing Längenangaben verwendet werden können. Nehmen Sie dabei an, dass maximal $N = 255\,\mathrm{B}$ Daten pro Frame übertragen werden können.

Es reicht, zwischen Flag- und Address-Feld ein neues Feld „Length“ einzufügen. Dieses habe eine Länge von 1 B und ermöglicht damit die Angabe der Werte $0, \ldots, 255$. Das Datenfeld trägt stets ein ganzzahliges Vielfaches von 8 bit. Damit werden Daten bis zu 255 B möglich. Die übrigen Headerfelder haben (da nicht anders angegeben) feste Längen.

[1] Das ist eine Vereinfachung. Bei HDLC können einige Header-Felder unterschiedliche Längen haben.
[2] Obwohl dies eine übliche Konvention ist, handelt es sich auch hier um eine vereinfachende Annahme.



Aufgabe 3 Cyclic Redundancy Check (CRC)

Die Nachricht `10101100` werde mittels CRC, wie in der Vorlesung eingeführt, gesichert. Als Reduktionspolynom sei $r(x) = x^3 + 1$ gegeben.

a)* Wie lang ist die Checksumme?

Die Länge der Checksumme in Bit entspricht dem Grad des Reduktionspolynoms, hier also grad($r(x)$) = 3 bit.

b) Bestimmen Sie die Checksumme für die gegebene Nachricht.

Zunächst werden an die Nachricht grad($r(x)$) = 3 Nullen angehängt: `10101100`**`000`**. Anschließend wird durch $r(x)$ dividiert:

```
10101100000 : 1001 = 10111011  Rest 011
1001|||||||
----|||||||
001111|||||
  1001|||||
  ----|||||
  01100||||
   1001||||
   ----||||
   01010|||
    1001|||
    ----|||
    001100|
      1001|
      ----|
      01010
       1001
       ----
       0011
```

c) Geben Sie die übertragene Bitfolge an.

Die übertragene Bitfolge besteht aus der ursprünglichen Nachricht konkateniert mit der eben berechneten Prüfsumme: `10101100 011`.

Bei der Übertragung trete nun das Fehlermuster `00100000000` auf.

d)* Wie lautet die empfangene Bitfolge?

Die empfangene Bitfolge ist das XOR aus der übertragenen Bitfolge und dem Fehlermuster:

```
     10101100011
XOR  00100000000
---------------
     10001100011
```

e) Zeigen Sie, dass der Übertragungsfehler erkannt wird.

Die empfangene Bitfolge wird wieder durch $r(x)$ dividiert:

```
10001100011 : 1001 = 10011111 Rest 100
1001|||||||
----|||||||
```

```
0001110||||
   1001||||
   ----||||
   01110|||
    1001|||
    ----|||
    01110||
     1001||
     ----||
     01111|
      1001|
      ----|
      01101
       1001
       ----
       0100
```

Es bleibt der Rest `100`. Bei einer fehlerfreien Übertragung hätte hingegen kein Rest bleiben dürfen.

f)* Geben Sie ein Fehlermuster an, welches nicht erkannt werden kann.

Vielfache des Reduktionspolynoms können nicht erkannt werden, z. B. `10010000000`:

```
     10101100011
XOR  10010000000
---------------
     00111100011
```

Division durch $r(x)$ ergibt:

```
                (war: 00110111 Rest 000)
00111100011 : 1001 = 00111011 Rest 000
001001|||||
  ----|||||
  01100||||
   1001||||
   ----||||
   01010|||
    1001|||
    ----|||
    001101|
      1001|
      ----|
      01001
       1001
       ----
       0000
```

g) CRC wurde in der Vorlesung ausdrücklich als fehlererkennender, nicht aber als fehlerkorrigierender Code eingeführt. Zeigen Sie, dass mittels CRC selbst 1 bit-Fehler im konkreten Beispiel dieser Aufgabe nicht korrigierbar sind.

Argumentativ Die übertragene Nachricht ist 11 bit lang, d. h. es gibt insgesamt elf mögliche 1 bit-Fehler. Die Checksumme ist jedoch nur 3 bit lang, d. h. es könnten maximal sieben Bitfehler unterschieden werden, da nur sieben von Null verschiedene Reste existieren. Damit ist eine eindeutige Zuordnung von einem Rest auf einen konkreten Bitfehler nicht möglich.

Beweis durch Gegenbeispiel Es reicht, zwei unterschiedliche Fehlermuster zu finden, die denselben Rest erzeugen, denn dann kann von diesem Rest nicht eindeutig auf einen der beiden Fehler geschlossen werden. Die Fehlermuster `00001000000` und `00000001000` liefern beide den Rest `001`.

Diskussion: Was ist mit längeren Checksummen?

Ein Ethernet-Rahmen hat eine Größe von bis zu 1518 B[3] inkl. Checksumme. Dies entspricht $1518 \cdot 8 = 12144$ möglichen 1 bit-Fehlern. Die Checksumme ist 32 bit lang, womit sich $2^{32} - 1$ von Null verschiedene Reste ergeben. Da nun $2^{32} - 1 > 12118$ ist, könnte nach obiger Darstellung eine Korrektur möglich sein. Allerdings muss dazu etwas mehr Mathematik betrachtet werden, da die Anzahl möglicher Reste und deren eindeutige Zuordbarkeit zu 1 bit-Fehlern vom Aufbau des Reduktionspolynoms abhängen.

Tatsächlich ist es nun so, dass mittels des bei Ethernet verwendeten Polynoms 1 bit-Fehler wirklich zu eindeutigen Resten führen. Eine Korrektur wäre damit zwar möglich, wird in der Praxis bei Ethernet aber nicht verwendet.

Grundsätzlich hängt es also von

- der Wahl des Reduktionspolynoms und
- dem Größenverhältnis zwischen Nutzdaten und Checksumme

ab, ob 1 bit-Fehler mittels CRC korrigierbar sind.

[3] Jumbo-Frames werden in GRNVS nicht betrachtet