Grundlagen: Rechnernetze und verteilte Systeme

Fünfte Woche: 14./18. Mai 2018

Medienzugriffsverfahren, Bit-Stuffing, Prüfsummen

Leo Glavinić
netze@eo.gl
eo.gl/netze

Inhalt

1. Schnelle Fragen zu Medienzugriffsverfahren

2. Bit-Stuffing

3. CRC

1. Medienzugriffsverfahren

a. Prinzip von ALOHA

Station sendet, sobald Daten vorliegen. Bestätigung von Übertragungen out-of-band (andere Frequenz)

1. Medienzugriffsverfahren

b. Kollisionserkennung in ALOHA

Ausbleiben der out-of-band Bestätigung

1. Medienzugriffsverfahren

c. Prinzip von Slotted ALOHA

Station sendet zu Beginn des nächsten Zeitslots; egal, ob bereits Übertragung stattfindet

1. Medienzugriffsverfahren

d. Vorteil von Slotted ALOHA

Verringerte Wahrscheinlichkeit für Kollisionen, da Stationen nicht mehr zu beliebigem Zeitpunkt mit Übertragung beginnen können

Kollisionsmöglichkeit nur zu Beginn eines Zeitslots; andernfalls sendet garantiert höchstens eine Station

1. Medienzugriffsverfahren

e. Prinzip von CSMA

Abhören des Mediums vor dem Senden; bei Freiheit im aktuellen Zeitslot kann im nächsten gesendet werden

1. Medienzugriffsverfahren

f. Ergänzung von CSMA/CD ggü. reinem CSMA

Kollisionserkennung, erneute Übertragung betroffener Rahmen

1. Medienzugriffsverfahren

g. Erkennung erfolgreicher Übertragungen bei CSMA/CD

Annahme einer Übertragung als erfolgreich, falls keine Kollision festgestellt (kein Jam-Signal) empfangen wurde

1. Medienzugriffsverfahren

h. Ergänzung von CSMA/CA ggü. reinem CSMA

keine Kollisionserkennung (i.A.); Verringerung der Auftrittswahrscheinlichkeit durch Randomisierung des Sendebeginns

1. Medienzugriffsverfahren

i. Binary Exponential Backoff

Alle CSMA-Varianten warten nach Kollision oder erfolgloser Übertragung eine zufällige Anzahl von Slotzeiten, die aus dem Backoff-Window zufällig und gleichverteilt gezogen wird; Verdopplung dieses Fensters nach jeder Kollision oder erfolgloser Übertragung (binary exponential), bis Maximalwert erreicht ist; Zurücksetzen des Fensters nach erfolgreicher Übertragung

1. Medienzugriffsverfahren

Relevanz für uns: Medienzugriffsverfahren kontrollieren, wann ein Rechner senden darf

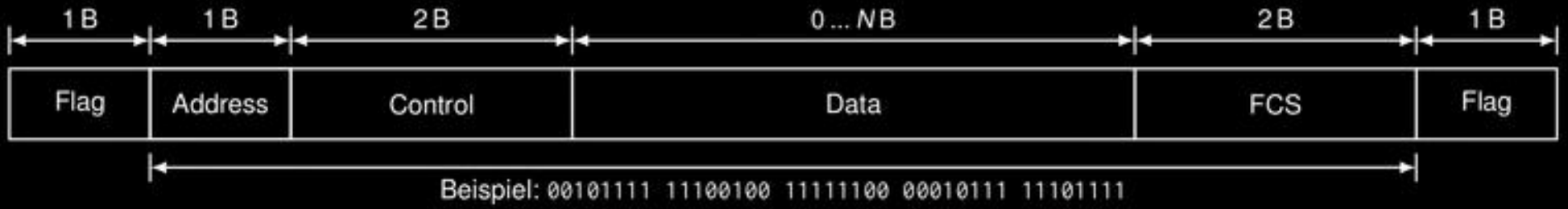
Wichtig: Umgang mit Wahrscheinlichkeiten (für Kollisionen, erfolgreiche Übertragungen usw.)

2. Bit-Stuffing

Problem auf Schicht 2: Erkennung von Rahmengrenzen → verschiedene Lösungsansätze! Steuerzeichen dürfen nicht zufällig durch die Konkatenation von Codewörtern auftreten

HDLC: High Level Data Link Control (Schicht-2-Protokoll für serielle Schnittstellen zwischen Cisco-Routern)

2. Bit-Stuffing



Spezielles Begrenzungsfeld (Flag) mit dem Wert 0x7e

konstante Länge der Headerfelder; Länge der Nutzdaten als ganzzahliges Vielfaches von 8 bit

2. Bit-Stuffing

a. zu übertragende Bitsequenz (auf dem Blatt) ohne Begrenzungsfelder: effizientes Stuffing-Verfahren, sodass 0x7e nicht in den Daten vorkommt

Einfügen einer Null nach jeweils fünf konsekutiven Einsen → leicht umkehrbar und effizient

2. Bit-Stuffing

b. Gesamte übertragene Bitsequenz nach Aufg. a

01111110001011111011001001111101000001011111010111101111110

2. Bit-Stuffing

c. Modifikation des Headers für Längenangaben anstatt von Bit-Stuffing; maximale Nutzdatengröße N=255 B

Feld für „Length“ (1 B) nach erstem Flag-Feld für Angabe der Nutzdatenlänge in B

2. Bit-Stuffing

Relevanz für uns: immer bedenken, dass Rahmengrenzen irgendwie eindeutig gemacht werden müssen!

Wer bei der Umwandlung von Hexadezimalzahlen in Binärzahlen Probleme hatte, sollte dies so bald wie möglich in den Griff bekommen ;)

3. Cyclic Redundancy Check (CRC)

Prüfsummenverfahren, mit dem der Empfänger einen Rahmen auf Bitfehler überprüfen kann

Gegeben: Reduktionspolynom *r(x)* in Binärdarstellung (Umwandlung von Graden der einzelnen Teilolynome in Stellenwerte der Binärzahl)

Anhängen von so vielen Nullen an die Nachricht wie der Grad von *r(x)* („höchster Exponent")

3. Cyclic Redundancy Check (CRC)

Division der erweiterten Nachricht durch *r(x)* (analog zur Grundschule; schriftliche Division binärer Zahlen: bitweise XOR-Operationen!)

Checksumme: Rest der Division, wird an ursprüngliche Nachricht angehängt

Division der empfangenen Nachricht durch *r(x)* beim Empfänger: kein Bitfehler, falls kein Rest bleibt

Fehlermuster: 1 für geflipptes Bit, 0 sonst

3. Cyclic Redundancy Check (CRC)

Übertragung von 10101100 mit $r(x)=x^3+1$

a. Länge der Checksumme

Gleich dem Grad von *r(x)*, hier 3 bit

3. Cyclic Redundancy Check (CRC)

b. Checksumme für gegebene Nachricht

Anhängen dreier Nullen: 10101100000

3. Cyclic Redundancy Check (CRC)

Division durch *r(x)*:

3. Cyclic Redundancy Check (CRC)

c. Versandte Bitfolge

10101100011

3. Cyclic Redundancy Check (CRC)

d. Empfangene Bitfolge bei Fehlermuster 00100000000

XOR aus versandter Bitfolge und Fehlermuster:
10001100011 (drittes Bit von links ist geflippt)

3. Cyclic Redundancy Check (CRC)

e. Erkennung des Übertragungsfehlers

Division der empfangenen Bitfolge durch *r(x)*:
Rest 100 (statt 0)

3. Cyclic Redundancy Check (CRC)

f. Fehlermuster, die nicht erkannt werden können

Vielfache des Reduktionspolynoms, hier z.B. 10010000000

Einfach ausprobieren ;)

3. Cyclic Redundancy Check (CRC)

g. Begründung, wieso in diesem Beispiel keine Bitfehler durch CRC korrigiert werden können

Übertragene Nachricht hat 11 bit → elf mögliche Bitfehler. Checksumme hat aber nur 3 bit → es können max. sieben Bitfehler unterschieden werden. Keine eindeutige Zuordnung von Rest auf Bitfehler!

Fehlermuster 00001000000 und 00000001000 liefern z.B. beide den Rest 001

3. Cyclic Redundancy Check (CRC)

Relevanz für uns: CRC korrigiert keine Fehler!

Wehe, jemand behauptet das Gegenteil in der Klausur...