Inhaltsverzeichnis

1 Formelsammlung

Entfernung	d
Bitrate	B oder R
Daten auf der Leitung	$R \cdot \frac{d}{c}$
Paketgröße	L
Übertragungsverzögerung (Paketsendezeit)	$d_{trans} = \frac{L}{R}$
Ausbreitungsverzögerung	$D = d_{prop} = \frac{d}{v}$
Ausbreitungsgeschwindigkeit	v
Durchschn. Paketankunftsrate?	a
Verkehrsintensität	$\frac{L \cdot a}{R}$
Kanalpuffergröße (in Paketen)	$a = \frac{R \cdot D}{L}$
Normierter Durchsatz	S

2 Grundlagen

Schichtenmodell

Schicht	Datenformat	Protokoll
Anwendung (application)	Nachrichten (messages)	HTTP, FTP
Transport (transport)	Segmente (segments)	TCP, UDP
Netzwerk (network)	Datagramme (packets/dataframes)	IP, routing
Verbindung (data link)	Rahmen (frames)	Ethernet
Physikalisch (physical)	Bits	

Dienstgüte (QOS) Antwortzeit, Durchsatz, Verlust-/Fehlerrate, Verfügbarkeit

Verzögerungen

- Verarbeitung auf dem Knoten d_{proc}
- Warteschlangenverzögerung d_{queue} auf dem ausgehenden Link
- Übertragungsverzögerung $d_{trans} = \frac{L}{R}$
- Ausbreitungsverzögerung $d_{prop} = \frac{d}{\sim 2 \cdot 10^8 \frac{m}{s}}$
- Gesamtverzögerung $d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$

3 Anwendungsschicht

3.1 HTTP

Skript Seite 8ff

2 Nachrichtentypen: Anfrage und Antwort

Anfrage

```
[Request line:] method URL version
[Header lines:] header_field_name: value
[Blank line]
[Entity body:]
Beispiel:
GET /abc/def.html HTTP/1.1
Host: www.example.com
User-agent: Mozilla/4.0
Connection: close
Accep-language: de
```

Methoden: GET (Abruf einer Seite), HEAD (Metadaten) und POST (nur hier Rumpf!)

Antwort

```
[Status line:] version status_code phrase
[Header lines:] header_field_name: value
[Blank line]
[Entity body:]

Beispiel:
HTTP/1.1 200 OK
Connection: close
Date: Mon, 06-09-11, 12:34 UTC
Server: Apache/1.3.0 (Unix)
Last-Modified: Sat, 06-09-09, 18:25 UTC
Content-Length: 6821
Content-Type: text/html

[Daten]
```

Codes: 200 OK, 301 Moved, 400 Bad Request, 404 Not Found, 505 HTTP Version Not Supported

Versionen Version 1.0: nicht-persistent. Verbindung auf - Daten - Verbindung zu

Version 1.1: persistent. Verbindung auf - Daten* - Verbindung zu. Erweiterung Pipelining: eine Anfrage für alle eingebetteten Objekte (Zeit für Aufbau und Anfrage gespart)

3.2 Netzwerk-Management

Überwachung und Verwaltung von

- Leistung (Auslastung, Durchsatz, Antwortzeiten...)
- Fehlern (Monitoring, Doku, Reaktion)
- Konfigurationen (Übersicht über Geräte und deren Config)
- Zugang (Festlegung, Kontrolle, Doku)
- Sicherheit (Monitoring, Kontrolle, Schlüsselverwaltung, Firewalls, Intrusion Detection...)

3.2.1 SNMP

- Managing Entity: Prozess auf Management Station
- Managed Device: Gerät
- Managed Object: Teil des Geräts
- Management Agent: Prozess auf Device, der lokale Aktionen ausführt
- Protokoll: GET/SET, GET-NEXT/GET-BULK, TRAP

3.3 Socket-Programmierung

Skript Seite 35ff, Buch S. 140ff

Alle Routinen in java.net.*

3.3.1 TCP

Client

```
import java.net.*;
Socket mySock = new Socket(''host'', Port);
DataOutputStream out = new DataOutputStream(mySock.getOutputStream());
BufferedReader in =
    new BufferedReader(new InputStreamReader(mySock.getInputStream()));
out.writeBytes(~); ~ = in.readLine();
mySock.close();
```

Server

```
import java.net.*;
ServerSocket accSock = new ServerSocket(Port);
while (true) {
    Socket conSock = accSock.accept();
    BufferedReader in =
        new BufferedReader(new InputStreamReader(conSock.getInputStream()));
    DataOutputStream out = new DataOutputStream(conSock.getOutputStream());
    ~ = in.readLine(); out.writeBytes(~);
    conSock.close();
}
```

3.3.2 UDP

Client

```
DatagramSocket sock = new DatagramSocket();
InetAddress addr = InetAddress.getByName(''host'');
byte[] data = new byte[size];
byte[] rcvData = new byte[size];
data = ~;
DatagramPacket sendPacket = new DatagramPacket(data, data.length, addr, PORT);
sock.send(sendPacket);
DatagramPacket rcvPacket = new DatagramPacket(rcvdata, rcvdata.length);
sock.receive(rcvPacket);
~ = rcvData;
sock.close();
```

Server

```
DatagramSocket sock = new DatagramSocket(PORT);
byte[] rcvData = new byte[size]; byte[] sendData = new byte[size];
while (true) {
    DatagramPacket rcvPacket = new DatagramPacket(rcvData, rcvData.length);
    sock.receive(rcvPacket);
    ~ = rcvPacket.getData();
    InetAddress addr = rcvPacket.getAddress();
    int port = rcvPacket.getPort();
    DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
        addr, port);
    sock.send(sendPacket);
}
```

3.3.3 Web-Server

Siehe Buch S. 154!

4 Transportschicht

Prinzip des (De-)Multiplexen: Ports identifizieren Anwendungsprozesse

Der Pseudo-Header (UDP, TCP) enthält zusätzlich Quell-/Ziel-IP, Protokollnummer und Segmentlänge für die Prüfsummenberechnung.

4.1 UDP

Segmentaufbau

```
source port #, dest port #
length of segment, checksum
message
```

Jedes Header-Feld ist 16 Bit groß

4.2 Fehlerkontrolle

4.2.1 Stop-and-Wait

Skript Seite 9

Sender

1. Sende Paket (Daten, SQN, CRC)
2. Starte Timer
3. ACK ohne Bitfehler + mit korrekter SQN? -> goto 1
4. Timeout? -> Nochmal senden, goto 3

Empfänger

1. Paket ohne Bitfehler + mit erwarteter SQN? -> ACK(SQN)
2. Sonst: ACK(SQN - 1)

4.2.2 Go-Back-N

Sender Fenster (Puffer) der Größe W

1. Daten zu senden + Platz im Fenster? -> sende Paket, nextSQN++. Wenn erstes Paket im Fenster, starte Timer.
2. ACK ohne Bitfehler + SQN im Fenster? -> Fensterbeginn = ACK.SQN. Wenn Fenster leer, stoppe Timer, sonst restart.
3. Timeout? -> alle unbestätigten Pakete senden, Timer restart

Empfänger

1. Paket ohne Bitfehler + erwartete SQN? -> ACK(SQN)
2. Sonst: letztes ACK noch mal senden

Gut an Go-Back-N: Kummulative ACKs (Verluste durch ACKs!), nur 1 Timer, kein Empfangspuffer

4.2.3 Selective Repeat

Sender

1. Daten zu senden + Platz im Fenster? -> sendet Paket, Timer(SQN) starten, nextSQN++
2. ACK ohne Bitfehler & SQN im Fenster? -> Paket als bestätigt markieren, Fenster bis zur nächsten Lücke verschieben
3. Timeout(SQN): Paket(SQN) nochmal senden, Timer(SQN) restart

Empfänger

1. Paket ohne Bitfehler + SQN im Fenster? -> ACK(SQN), Paket puffern, Fenster bis zur nächsten Lücke
2. Sonst: Letztes ACK erneut senden

Gut an Selective Repeat: Weniger Paket-Wiederholungen!

4.2.4 Leistungsanalyse

Skript Seite 43ff

Stop-and-Wait ohne Fehler: Durchsatz $= \frac{L}{\frac{L}{R}+2D}$, $S = \frac{1}{1+2a}$ ist schlecht für große a! Mit Fehlern $S = \frac{1-p}{1+2a}$

Schiebefenster allgemein ohne Fehler: $S = \begin{cases} 1 & W > 1 + 2a \\ \frac{W}{1+2a} & W < 1 + 2a \end{cases}$ => Ab $a = \frac{W-1}{2}$ sinkt der Durchsatz!

Selective Repeat mit Fehlern $S = \begin{cases} 1 - p & W \geq 1 + 2a \\ \frac{W \cdot (1-p)}{1+2a} & W < 1 + 2a \end{cases}$

=> Selective Repeat ist bei großen Fenstern, hohen Fehlerraten oder großem a besser!

Go-Back-N mit Fehlern $S = \begin{cases} \frac{1-p}{1+2a \cdot p} & W \geq 1 + 2a \\ \frac{W(1-p)}{(1-p+W \cdot p)(1+2a)} & W < 1 + 2a \end{cases}$

4.3 TCP

Skript Seite 54ff

Features: Reihenfolge bewahrend, fensterbasierte Fehlerkontrolle,
vollduplex, verbindungsorientiert, Flußkontrolle, Überlastkontrolle

Segmentformat

```
source port #, dest port #
sequence number
acknowledgement number
data offset, reserved, flags, advertized window
checksum, urgent data pointer
options
data
```

Flags

URG Urgent Data Pointer gültig

ACK acknowledgement number gültig

PSH Push Segment (ohne Puffer sofort an Anwendung)

RST Connection Reset

SYN Synchronisation von SQN beim Verbindungsaufbau

FIN Keine Daten mehr vom Sender

4.3.1 Fehlerkontrolle

Mischung aus Selective Repeat & Go-Back-N:

- Puffer auf Sender- und Empfangsseite
- Nur ein Timer, kummulative ACKs
- SQN: Position des ersten Bytes des Segments im Bytestrom
- ACK: Position des NÄCHSTEN erwarteten Bytes im Bytestrom

Sender

1. Daten zu senden + Platz im Fenster? -> Segment mit nextSQN erstellen und senden, nextSQN += sizeof(segment). Erstes Segment im Fenster? -> Timer starten
2. ACK mit SQN im Fenster? -> Fenster an Stelle SQN schieben. Fenster leer? -> Timer stoppen. Sonst Timer restart
3. Timeout? -> Erstes unbestätigtes Paket schicken, Timer restart

Empfänger

1. SQN = Fensteranfang + alle vorherigen Segmente bestätigt? -> Fensteranfang auf nächste Position, delayed ACK
2. SQN = Fensteranfang + unbestätigtes vorheriges Segment? -> Fensteranfang auf nächste Position, ACK(Fensteranfang)
3. SQN > Fensteranfang? -> im Puffer ablegen, ACK(Fensteranfang)
4. Paket ist Lückenfüller? -> im Puffer ablegen, Fensteranfang verschieben, ACK(Fensteranfang)

Modifikationen

Fast Retransmit Bei 4 ACKs mit der gleichen SQN wird das Segment mit dieser SQN erneut gesendet, Timer egal

Selective_ACKs Zusätzlich im Optionsfeld

Zeitstempel im Optionsfeld SQN-Raum zu klein für hohe Datenraten

4.3.2 Verbindungsauf- / Abbau

Aufbau aktiver Client (connect), passiver Server (listen/accept)

- SYN-Segment: SYN-Flag = 1, zufällige SQN, keine Daten
- SYN/ACK-Segment: SYN-Flag = 1, ACK-Flag = 1, zufällige SQN, ACK = client_SQN + 1, keine Daten
- ACK-Segment: ACK-Flag = 1, SQN = client_SQN + 1, ACK = server_SQN, Daten

Abbau

- FIN-Flag = 1
- ACK-Flag = 1
- Timed-Wait für 2 MSS

4.3.3 Timeout-Schätzung

Zeitstempel im Optionsfeld liefert aktuelle RTT (SampleRTT)

$\text{EstimatedRTT} = (1-\alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}$, $\alpha = 0{,}125$ typisch
$\text{DevRTT} = (1-\beta) \cdot \text{DevRTT} + \beta \cdot |\text{SampleRTT} - \text{EstimatedRTT}|$, $\beta = 0{,}25$ typisch
$\text{Timeout} = \text{EstimatedRTT} + 4 \cdot \text{DevRTT}$

Bei jedem Timeout den Wert verdoppeln, bis wieder Pakete eintreffen.

4.3.4 Flusskontrolle

AdvertizedWindow: Empfänger teilt freie Puffergröße mit.

EffectiveWindow: Noch frei vermuteter Platz beim Empfänger (Senderseite).

4.3.5 Überlastkontrolle

CongestionWindow bestimmt Sendefenster.

1. Slow Start: CongestionWindow = MSS, exponentielle Erhöhung bis 3 dupACKs eintreffen
2. AIMD (Additive Increase, Multiplicative Decrease): CW halbieren, linear erhöhen bis 3 dupACKs eintreffen
3. Bei Timeout: Slow Start bis $\frac{1}{2}$CW

4.3.6 Leistungsanalyse

Skript Seite 78-83

geringe Bitraten: Übertragungszeit ˜ Antwortzeit => persistente Verbindung nur geringfügig besser

hohe RTT: Slow-Start-Zeit ˜ Antwortzeit => persistente Verbindung top

Legende:

S	MSS in Bit
O	Objektgröße in Bit
R	Bitrate
RTT	Round Trip Time
W	Fenstergröße in MSS
K	# Fenster, um Objekt zu übertragen
Q	# Wartezeiten für unendliche großes Objekt
P	# Wartezeiten für Objekt der Größe O
$delay$	Gesamtverzögerung

Allgemein gilt: $delay = 2 \cdot RTT + \frac{O}{R} + \text{SlowStart}$

SlowStart-Wartezeit im k-ten Fenster bei Verdopplung der Fenstergröße:

Sendezeit im k-ten Fenster: $2^{k-1} \cdot \frac{S}{R}$, Zeit Sendebeginn - ACK: $\frac{S}{R} + \text{RTT}$ => $\frac{S}{R} + \text{RTT} - 2^{k-1}\frac{S}{R}$ ist Wartezeit.

Damit folgt: $delay = \frac{O}{R} + 2 \cdot \text{RTT} + \sum_{k=1}^{P} \left[\frac{S}{R} + \text{RTT} - 2^{k-1}\frac{S}{R}\right] = \frac{O}{R} + 2 \cdot RTT + P\left[\text{RTT} + \frac{S}{R}\right] - \left(2^P - 1\right)\frac{S}{R}$ wegen der geometrischen Reihe $s_n = \sum_{k=0}^{n} q^k = \frac{q^{n+1}-1}{q-1}$

Anzahl Slow Start Wartezeiten P

$K = \min\left\{k : 2^0 + 2^1 + \cdots + 2^{k-1} \geq \frac{O}{S}\right\} = \min\left\{k : 2^k - 1 \geq \frac{O}{S}\right\} = \min\left\{k : k \geq \log_2\left(\frac{O}{S} + 1\right)\right\} = \left\lceil \log_2\left(\frac{O}{S} + 1\right)\right\rceil$

$Q = \max\left\{k : \frac{S}{R} + \text{RTT} - 2^{k-1}\frac{S}{R} \geq 0\right\} = \max\left\{k : 2^{k-1} \leq 1 + \frac{\text{RTT}}{\frac{S}{R}}\right\} = \left\lfloor \log_2\left(1 + \frac{\text{RTT}}{\frac{S}{R}}\right)\right\rfloor + 1$

=> $P = \min(Q, K - 1)$

5 Netzwerkschicht

Kommunikation zwischen Hosts über verschiedene Netze: Forwarding und Routing.

- Entweder Datagram-basierte Paketvermittlung: jedes Datagram trägt globale Adresse -> IP. => Keine Fehlerkontrolle, Reihenfolge, Verbindungen, Fluss-/Überlastkontrolle, Dienstgüte!
- Oder virtuelle Leitungsvermittlung: Dateneinheiten tragen lokale Kennungen, bei jeder Vermittlungseinheit wird diese verändert -> X.25, ATM, MPLS... => virtuelle Leitung, Bereitstellung von Merkmalen (s.o.)

5.1 IP

Datagram-Format

```
version, IP header length, type of service, length
identifier, flags, fragment offset [<- Fragmentierungs-Infos!]
TTL, upper layer, checksum
source IP
dest IP
options
DATA
```

Klassenbasierte Adressierung (Netzwerkteil + Hostteil)

Pro: selbst identifizierende Adressen. Contra: feste Zuordnung Adresse - Netzwerk (Laptop!), zwischen Classs B und C zu großer Abstand -> Verschwendung

Netze

Klasse	Aufbau	Bereich	Größe Hostteil
A	0NH	1.0.0.0 - 127.255.255.255	24 Bit
B	10NH	128.0.0.0 - 191.255.255.255	16 Bit
C	110NH	192.0.0.0 - 223.255.255.255	8 Bit

Subnets #H - #Hosts = #Subnet
Beispiel: Class B => #H = 16, 100 Hosts/Subnet => 7 Bit nötig => #Subnet = 16 - 7 = 9.

#N + #H = Maske
Beispiel: 16 + 9 = 25 => 150.100.12.128/25

Fragmentierung Wenn unterwegs kleinere MTU auftaucht: Datagramm fragmentieren -> Header-Felder setzen. Reassemblierung erst am Ziel!

ICMP

```
typ, code, checksum, daten
```

Wird in IP-Datagrammen befördert.

5.2 Routing

Aufbau Router

1. Eingangsport: Line termination, data link processing (decapsulation), PUFFER
2. Switching fabric: Speicher (2x Bus benutzen!), Bus (kleine bis mittlere Router) oder Crossbar (quadrat. Aufwand!)
3. Ausgangsport: PUFFER, data link processing, line termination

5.2.1 Intra-Domain-Routing

Link-State-Routing (Skript Seite 28) Jeder Router berechnet kürzeste Pfade zu **allen** anderen Routern, z.B. OSPF. => Informationen über Netz-Topologie müssen bekannt gemacht werden.

=> Fluten: Link-State-Advertisements (LSAs) mit Kosten zu allen Nachbarknoten an Nachbarn schicken, dort weiter verteilen.

Dijkstra

1. Knotenmenge N, Endmenge N' = **W**urzelknoten.
2. Kosten zu Nachbarknoten von W berechnen (c, 0, ∞)
3. Knoten mit kleinsten Kosten zu N' hinzufügen
4. Kosten zu Nachbarknoten von N' berechnen
5. goto 3 until done

Forward-Search (Dijkstra in der Routing-Praxis)

- 2 Listen: bestätigt und vorläufig
- Werte der Form (Ziel, Knoten, nextHop)
- => OSPF: periodisches Fluten, verschiedene Metriken

Distanzvektor-Routing (Skript Seite 35) Router kennen Kosten zu direkten Nachbarn und die davon aus erreichbaren Ziele, z.B. RIP.

Bellman-Ford-Verfahren

1. Kosten $\forall$Nachbarn eintragen: (Ziel, kosten, nextHop), wobei kosten aus $\{0, c, \infty\}$
2. Tabelle an alle Nachbarn schicken
3. Tabelle aktualisieren und an alle Nachbarn schicken
4. goto 3 until stable

Problem: Vergrößerung der Kosten kann zyklischen Pfad erzeugen! (bad new travel slowly)

=> RIP: Advertisements, Kostenmetrik = #Hops

Vergleich Link-State + Distanzvektor (Skript Seite 45) LS: gut

DV: nicht so gut

5.2.2 Interdomain-Routing

Pfadbasiertes Routing (Skript Seite 46)

- Austausch von ganzen Pfaden - ohne Angabe von Knoten
- Nur Pfade bekanntgeben, die andere Router auch nutzen sollen
- Router sucht nach individuellen Regeln aus

Border Gateway Protocol

- Austausch von Advertisements (Pfade: Liste von Netzwerken in einem erreichbaren AS, AS-Nummern die da hin führen, IP des Senders)
- Beim Weiterleiten eines Pfads Anfügen des eigenen AS und Eintragen als nextHop -> Pfade mit sich selbst verwerfen

MPLS Skript Seite 49

6 Verbindungsschicht

- Datagramme in Rahmen einpacken
- Adressierung über physikalische Adresse
- Datensicherung, Medienzugriff, evtl. Flusskontrolle

6.1 Adressierung

MAC-Adresse: 48 Bit $\widehat{=}$ 12 Hex-Ziffern

Die physikalische Adressen in den Rahmen wechseln bei Netz-Übertritten während die IPs in den Headern immer gleich bleiben!

ARP ARP-Tabelle pro Knoten: (IP-Addr, MAC, TTL)

ARP-Anfragen an Broadcast-Addr. FF*

6.2 Datensicherung (Skript Seite 6)

Rauschen, Einstrahlung -> häufig gebündelte Fehler

- Fehlererkennung (Parität, CRC)
- Fehlerkorrektur (Redundanz), nur bei stark gestörten Kanälen

CRC
Generatorpolynom G mit g Bit und Nutzdaten D -> Prüfdaten R mit $r = g - 1$ Bit.

$R = \text{remainder}\left(\frac{D \cdot 2^r}{G}\right) \Rightarrow (D, R) = D \cdot 2^r + R$

Bemerkung: $D \cdot 2^r$ entspricht r Nullen an D anhängen (shift-Operation!)

Wenn beim Empfänger $\frac{(D,R)}{G} = 0$ dann kein Fehler.

6.3 Medienzugriff

Koordinierung nur bei Mehrfachzugriff auf das Medium:

- feste Kanalteilung: Frequenz-, Zeit-, Codemultiplex (Chipping-Code erzeugt Signal mit höerer Frequenz)
- Zufallszugriff: ALOHA, CSMA
- Zyklische Zuteilung: Token Ring, FDDI, USB...

Zufälliger Medienzugriff (ALOHA, CSMA)

- Wenn Daten zu senden -> senden!
- Kollisionen durch Sendewiederholung beheben. Bei schwacher Last Kollisionen selten.

6.3.1 ALOHA

- Ein Kanal
- Rahmen sofort senden
- Fehlerfreier Empfang -> ACK
- Timeout ohne ACK: Backoff (zufällige Zeit) und Sendewiederholung

-> Sehr einfaches Protokoll

Binärer exponentieller Backoff Nach jeder Kollision Variable m erhöhen. Backoff-Zeit $= k \cdot \tau$, wobei k zufällig aus $\{0, 1, 2, \ldots, 2^m - 1\}$

=> Backoff-Zeit an aktuelle Last angepasst. Wenn m bestimmten Wert überschreitet aufgeben und Fehlermeldung.

Leistungsanalyse (Skript Seite 15) $S = N \cdot p\,(1-p)^{2(N-1)}$mit N Stationen und Sendewahrscheinlichkeit p.

=> Schlecht! Max. 18% Auslastung!

Slotted ALOHA Sendebeginn nur zu Beginn eines Slots => $S_{max} = 37\%$

Reservation ALOHA (Skript Seite 25)

6.3.2 CSMA

Listen Before Talking!

Geht nur wenn Ausbreitungsverzögerung < Sendezeit!

1. Daten zu senden? -> Listen on medium
2. Medium frei? -> senden. Sonst: warten
3. Fehlerfreier Empfang: ACK
4. Timeout ohne ACK: Backoff (zufällige Zeit), Sendewiederholung

Varianten

- 1-persistent: Medium belegt -> warten bis frei, sofort senden. => geringe Wartezeit, hohe Kollisionsgefahr wenn mehrere warten
- nicht-persistent: Medium belegt -> Backoff => hohe Wartezeiten, wenig Kollisionen
- p-persistent: Mit Wsk. p senden, mit Wsk. p-1 warten

CSMA/CD Listen While Talking!

- Kollisionserkennung: Sendeabbruch, Jamming-Signal
- keine ACKs

=> Leistung kritisch von Kanalpuffergröße $a = \frac{R \cdot D}{L}$ abhängig.

6.3.3 Zyklische Zuteilung

- Token Ring:
  - Listen Mode, Transmit Mode
  - Zirkulierendes Bitmuster mit 2 Zuständen (frei, belegt)
  - Sender setzt belegt, Empfänger nimmt entgegen, Sender (!) entfernt Daten wieder und setzt frei
- FDDI $\hat{=}$Multi-Token-Ring: mehrere Tokens auf Ring, Sender sendet nach Daten freies Token

6.4 Ethernet

Rahmenformat

```
Präambel [7 Bytes alternierende Bits Synchronisation, 1 Byte Schluss]
Zieladresse
Quelladresse
2 Byte Typ [Protokoll der Nutzdaten (IP, IPX, AppleTalk...)]
Nutzdaten [46-1500 Bytes]
CRC [4 Bytes/32 Bit]
```

=> Min. 64 Bytes, max. 1518 (Präambel zählt nicht)

Medienzugriff

- 1-persistentes CSMA/CD mit 48 Bit Jam-Signal
- Binärer exponentieller Backoff mit max. 16 Wiederholungen und $k \cdot 512$ Bitzeiten
- verbindungslos -> kein Handshake
- unzuverlässig -> keine ACKs

Varianten

10Base2 10 MBps, Max. 2,5 km Segmentlänge, $S_{max} = 16\%$(64 Byte-Rahmen) - 95% (6400 Byte Rahmen)

10BaseT Max. Distanz Hub-Knoten: 100m. Hub = Repeater mit vielen Ports. Switch = Bridge mit Pufferung -> keine Kollisionen bei den Knoten!

100BaseT Fast Ethernet. 100 MBps. Max. 100m Hub-Knoten, bei Glasfaser (100BaseTX) 2 km

1000BaseT Gigabit Ethernet (Skript Seite 10). 1 GBps. Min. Rahmengröße 512 Bytes. Distanz 100m - 5km

Switches

- Selbst-lernen: eingehende Pakete definieren phys. Adressen des Knotens am Port, ausgehende werden entsprechend geleitet (Fluten bei unbekannter Adresse)
- Aufspannender Baum: Zyklische Strukturen im Netz -> kein selbst-lernen möglich! -> Verteilter Algorithmus

6.5 WLAN

- Long-range-fading: Signalstärke sinkt quadratisch zum Abstand
- Interferenzen
- Short-range-fading: Auslöschung durch phasenverschobene, reflektierte Wellen

=> Hohe Fehlerrate, v.a. Fehler-Bursts

Hidden-Terminal-Problem: Zwei Knoten wissen nichts von Kollisionen miteinander wenn sie sich gegenseitig nicht hören!

6.5.1 IEEE 802.11

- 4 physikalische Adressen für APs
- Medienzugriff CSMA/CA (weil CD 2. Antenne benötigen würde) oder Basic Access
- Stationen (STA) = Knoten
- Basic Service Set (BSS): Stationen, die denselben Kanal verwenden
- Access Points (APs)
- Distribution System: Verbindung mehrerer Zellen

Rahmenformat (Skript Seite 15/16)

6.5.2 Basic Access

- Listen before talking für Zeit DIFS (Distribution Interframe Space). -> frei: senden, sonst Backoff
- Empfänger: fehlerfrei erhalten? -> Warte SIFS (Short Interframe Space), sende ACK
- Timeout ohne ACK? -> Backoff
- Binary exponentially truncated backoff: Wie sonst, aber nur solange Medium frei ist Zähler reduzieren!

6.5.3 RTC/CTS

Skript Seite 20

Request-to-send, Clear-to-send vor eigentlicher Übertragung.

Contra: Overhead! Pro: kurze Kollisionen, Hidden-Terminal-Problem gelöst

7 Physikalische Schicht

Bandbreitenbeschränkung: Skript Seite 8-10

Nyquist-Theorem / Shannon-Theorem: Skript Seite 13

Modulation: Skript Seite 14

Signalübertragung über Leitungen: Skript Seite 21