Paketverzögerung

1. Ausbreitungsverzögerung $D = \frac{L}{v}$
2. Kanalpuffergröße $a = \frac{R \cdot D}{L} = \frac{\frac{l}{v}}{\frac{L}{R}}$
3. Übertragung in Paketen über mehrere Links: $d_{trans} = n_{Links} \cdot \frac{L}{R} + (n_{pakete} - 1) \cdot \frac{L}{R}$ (wenn leitungsvermittelt: $n_{links} = 1$)
4. mittlere Warteschlangenverzögerung $d = \frac{d_{ges}}{N}$, $d_{ges} = \sum_{i=1}^{N}(i-1) \cdot \frac{L}{R} = \frac{N \cdot (N-1)}{2} \cdot \frac{L}{R}$
5. Wahrscheinlichkeit, dass n Nutzer gleichzeitig senden: $P_n = \binom{\max}{n} \cdot (P_{user})^n \cdot (1 - P_{user})^{\max - n}$

TCP-Leistungsanalyse

1. Verbindungs**AUF**bau: $2 \cdot RTT$ für:
   (a) $C \to S$: `SYN` (seqnum: client_isn)
   (b) $S \to C$: `SYN` + `ACK` (seqnum: server_isn, acknum: client_isn + 1)
   (c) $C \to S$: `ACK` + Request (seqnum: client_isn + 1, acknum: server_isn + 1)
2. Verbindungs**AB**bau:
   (a) $C \to S$: `FIN` (seqnum: client_sqn)
   (b) $S \to C$: `ACK` (seqnum: server_sqn, acknum: client_sqn + 1)
   (c) $S \to C$: `FIN` (seqnum: server_sqn)
   (d) $C \to S$: `ACK` (seqnum: client_sqn + 1, acknum: server_sqn + 1)
   (e) C wartet noch 2 Segementlebensdauern auf mögliche alte Segmente ("time wait"), S ist nach dem Erhalt des letzten ACKs fertig.
3. Bis zu welchem Fenster k treten bei einem unendlich großen Objekt Wartezeiten auf?
   $Q = \max\left\{k : \frac{L}{R} + RTT - 2^{k-1}\frac{L}{R} \geq 0\right\} = \left\lfloor \log_2\left(1 + \frac{RTT}{\frac{L}{R}}\right)\right\rfloor + 1$
4. Wie viele Fenster K werden benötigt, um ein Objekt der Größe O zu übertragen?
   $K = \min\left\{k : \sum_{i=1}^{k} 2^{i-1} L \geq 0\right\} = \lceil \log_2\left(\frac{O}{L} + 1\right)\rceil$
5. Wie viele Wartezeiten P treten bei einem endlich großen Objekt der Größe O auf?
   $P = \min\{Q\ ;\ K - 1\}$
6. Delay $= \underbrace{2RTT}_{\text{Verbindungsaufbau}} + \underbrace{\frac{O}{R}}_{\text{Übertragung des reinen Objekts}} + \underbrace{P \cdot \left(RTT + \frac{L}{R}\right) - \underbrace{\left(\sum_{k=1}^{P} 2^{k-1}\right)}_{=2^P - 1} \cdot \frac{L}{R}}_{\text{Slow-Start Wartezeit}}$
7. mehrere Links T: $d = 2 \cdot RTT_T + \frac{O}{R} + (T-1) \cdot \frac{L}{R} + P_T \cdot (RTT_T + \frac{TL}{R}) - (2^{P_T} - 1) \cdot \frac{L}{R}$
   $Q_T = \lfloor \log_2\left(T + \frac{RTT}{\frac{L}{R}}\right)\rfloor + 1$
   K = wie normal
8. Verzögerung `HTTP` mit M Objekten und 1 Basisseite:
   (a) nicht-persistent (Für jedes Objekt eine extra Verbindung): $(M+1)\left(2 \cdot RTT + \frac{O}{R} + \text{SSW}\right)$
   (b) persistent mit Pipelining: Q und K konstant (Eine Verbindung für alle Objekte (evtl. mit Pipelining für parallele Übertragung der Objekte)): $3\ RTT + (M+1) \cdot \frac{O}{R} + \text{gem. SSW}$
   gem. SSW = SSW + RTT − $\max\left\{\frac{L}{R} + RTT - \left(2^{K-1} \cdot \frac{L}{R}\right); 0\right\}$
   (c) nicht-persistent mit X parallelen Verbindungen ($\frac{M}{X}$ ganze Zahl):
   in $SSW_{parallel}$ R durch $\frac{R}{X}$ ersetzen $\left(\frac{M}{X} + 1\right) \cdot 2 \cdot RTT + (M+1) \cdot \frac{O}{R} + SSW_{normal} + SSW_{parallel}$
9. Flusskontrolle: Empfänger steuert Sender um Überlastung zu verhindern, aber Geschwindigkeit zu maximieren
   Falls mehr als 16 Bit für Fenstergröße benötigt werden: Scaling Factor $F \leq 14$, dann gilt window = advertised window $\cdot\ 2^F$
10. Überlastkontrolle: Sender wird davon abgehalten, das Netz zu überlasten

Slow Start Setze CongestionWindow ← MSS. Nach Erhalt eines ACKs setzte

$$\text{CongestionWindow} \leftarrow \text{CongestionWindow} + \text{MSS}.$$

Wenn der Threshold erreicht ist mache mit AMID weiter, zu Beginn ist dieser jedoch unendlich.

AIMD Nach drei doppelten ACKs kommt es zum AIMD. Hier kommt es zu einem *multiplicative decrease*, bei dem der Threshold und das CongestionWindow auf

$$\text{Threshold} = \frac{\text{CongestionWindow}}{2} \quad \text{und} \quad \text{CongestionWindow} = \frac{\text{CongestionWindow}}{2}$$

gesetzt werden. Bei Erhalt eines ACKs wird dann das CongestionWindow auf

$$\text{CongestionWindow} = \text{CongestionWindow} + \text{MSS} \cdot \frac{\text{MSS}}{\text{CongestionWindow}}$$

gesetzt. Hierdurch wird ein Wachstum um circa ein MSS pro RTT realisiert.

Timeout Hier wird der Threshold und das CongestionWindow auf

$$\text{Threshold} = \frac{\text{CongestionWindow}}{2} \quad \text{und} \quad \text{CongestionWindow} = \text{MSS}$$

gesetzt. Anschließend Slow-Start

Fehlerkontrolle

1. Stop-And-Wait:
   (a) (Leitungs-)Durchsatz: $\frac{L}{N \cdot \left(\frac{L}{R} + 2D\right)}$
   (b) Normierter Durchsatz: $S = \frac{1}{N(1+2a)}$
   (c) Falls keine Fehler: $N = 1$
   (d) Mittlere Anzahl der Sendeversuche pro Paket bei Fehlerwahrscheinlichkeit p: $N = \frac{1}{1-p} \Rightarrow S = \frac{1-p}{1+2a}$
2. Selective-Repeat:
   (a) $W > 1 + 2a$:
      i. ohne Fehler: $S = \frac{W \cdot L}{W \cdot \frac{L}{R}} \cdot \frac{1}{R} = 1$
      ii. mit Fehlerwahrscheinlichkeit p: $S = 1 - p$
   (b) $W < 1 + 2a$:
      i. ohne Fehler: $S = \frac{W}{1+2a}$
      ii. mit Fehlerwahrscheinlichkeit p: $S = \frac{W \cdot (1-p)}{1+2a}$
3. Go-Back-N (ohne Fehler wie Selective-Repeat) mit Fehlerwahrscheinlichkeit p:
   (a) Mittlere Anzahl Sendeversuche pro Paket: $N = \frac{1-p+K \cdot p}{1-p}$
   (b) $W \geq 1 + 2a\ (K = 1 + 2a)$: $S = \frac{1-p}{1+2ap}$
   (c) $W < 1 + 2a\ (K = W)$: $S = \frac{W \cdot (1-p)}{(1-p+W \cdot p) \cdot (1+2a)}$

Anzahl Sequenznummern m $(= 2^n)$

1. Falls Empfangsfenstergröße = 1: $W < m$ hinreichend
2. Falls Sendefenstergröße = Empfangsfenstergröße > 1: $W < \frac{m+1}{2}$

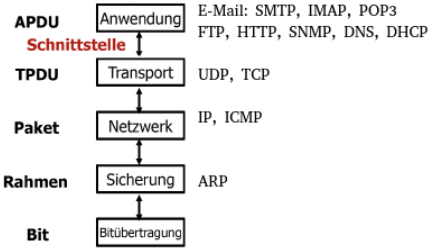
Leistungsanalyse Medienzugriff

1. (Slotted-)ALOHA:
   (a) Wahrscheinlichkeit für Senden ohne Kollision bei Sendewahrscheinlichkeit p: $p \cdot (1-p)^{2N-2}$
   (b) Normalisierter Durchsatz: $S = N \cdot p(1-p)^{2(N-1)}$
   (c) Sendeversuche pro Slot: $G = N \cdot p$
2. CSMA (listen before talking)
   (a) 1-persistent: Bei belegtem Medium warten bis frei, dann sofort senden
   (b) nicht-persistent: Bei belegtem Medium Backoff
   (c) p-persistent: Wenn Medium wieder frei, Senden mit Wahrscheinlichkeit p oder einen weiteren Slot abwarten $(1-p)$
3. CSMA/CD (listen while talking)
   (a) Normierter Durchsatz: $\frac{1}{1+4,4a}$
   (b) Minimale Rahmengröße L: $\frac{L}{R} > 2 \cdot D$
4. Ethernet: Präambel 8 Byte, minimal Header 64 Byte, 1-persistentes CSMA/CD, unzuverlässig da keine ACKs

Cyclic Redundancy Check

1. Subtraktion = Addition = XOR
2. Generatorpolynom G = r + 1 bits ($x^2 + 1 = 101_2, r = 2$)
3. $(D << r) \div G = X$& Rest R
4. $(D << r) + Rest$ = Endergebnis

Eigenschaften bei der Fehlererkennung
- alle Einzelbitfehler, wenn Koeffizienten von x^r und x^0 gleich Eins
- alle Doppelbitfehler, falls G unzerlegbaren Faktor mit mindestens 3 Termen enthält
- jede ungerade Bitfehlerzahl, falls G den Faktor (x+1) enthält
- jeder Fehlerburst, der kürzer als r Bits ist (die meisten längeren Fehlerbursts können ebenfalls erkannt werden)



FTP

1. Active Mode: Server baut Verbindung zu Port des Clients auf
2. Passive Mode: Client baut Verbindung zu Port des Servers auf → Vorteile: Server hinter NAT, Keine Portfreigabe in Firewall des Clients
3. Out-of-Band Control: Getrennte Verbindungen für Steuerbefehle und Dateien

[Verzögerungszeiten an einem Knoten]

$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

- d_{proc} = Verarbeitungsverzögerung (processing delay)
  - typischerweise wenige Mikrosekunden oder noch weniger
- d_{queue} = Warteschlangenverzögerung (queuing delay)
  - lastabhängig
- d_{trans} = Übertragungsverzögerung (transmission delay)
  - = L/R, bei langsamen Verbindungen ein signifikanter Anteil
- d_{prop} = Ausbreitungsverzögerung (propagation delay)
  - Wenige Mikrosekunden bis hunderte Millisekunden

Latenzzeitenberechnung

d_{prop} = Ausbreitungsverzögerung und d_{trans} = Übertragungsverzögerung

Cut-Through
Vorteil: Sehr schnell
Nachteil: Fehler können auftreten
→ Zeit s um ein Paket zu verschicken:

$$s = d_{prop} + d_{trans}$$

$E \cdot (\frac{H_2}{R} + d_{prop}) + (L + \frac{\text{anderer Header}}{R})$

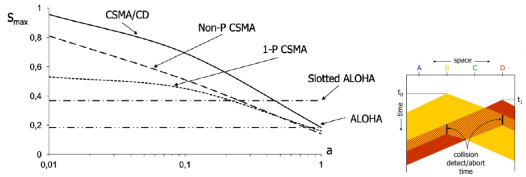
Store-And-Forward
Vorteil: keine Fehler
Nachteil: Langsam
→ Zeit s um ein Paket zu verschicken (E = Links):

$$s = E \cdot d_{trans} + d_{prop}$$

$E \cdot (\frac{L}{R} + \frac{\text{Header}}{R} + d_{prop})$

Statistisches Multiplexen: Aufteilung der Leitung je nach Bedarf, da die Reihenfolge der Pakete keinem Muster folgt

Vorteil von Paketvermittlung zu Leitungsvermittlung: Ressourcen werden während Inaktivität durch andere genutzt, dadurch wird die Bitrate effizienter aufgeteilt



throws Exception!!!
Befehle mit `startsWith("xy")` überprüfen

TCP-Client

```
Socket sock = new Socket("hostname", 6789);
BufferedWriter out = new BufferedWriter(new
↪  OutputStreamWriter(sock.getOutputStream(),
↪  "UTF-8"));
BufferedReader in = new BufferedReader(new
↪  InputStreamReader(sock.getInputStream(),
↪  "UTF-8"));
String sentence = in.readLine();
out.write("text\r\n");
out.flush();
sock.close();
```

TCP-Server

```
ServerSocket server = new ServerSocket(6789);
while (true) {
    Socket sock = server.accept();
    BufferedWriter out = new BufferedWriter(new
    ↪  OutputStreamWriter(sock.getOutputStream(),
    ↪  "UTF-8"));
    BufferedReader in = new BufferedReader(new
    ↪  InputStreamReader(sock.getInputStream(),
    ↪  "UTF-8"));
    String sentence = in.readLine();
    out.write("antwort\r\n");
    out.flush();
    sock.close();
}
```

1. SQN (des Servers) = Anfang des Pakets
2. ACK (des Clients) = Anfang des nächsten erwarteten Pakets
3. RTT-Abschätzung für möglichst gutes Timeout
4. TCP: Overhead → Langsamer, dafür kommen Daten sicher und in richtiger Reihenfolge an
5. UDP: Schnell, Daten können verloren gehen oder in falscher Reihenfolge ankommen

UDP-Client

```
BufferedReader in = new BufferedReader(new
↪  InputStreamReader(System.in));
DatagramSocket sock = new DatagramSocket();
InetAddress ipAddress =
↪  InetAddress.getByName("hostname");
byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];
String sentence = in.readLine();
sendData = sentence.getBytes("UTF-8");
DatagramPacket sendPacket = new
↪  DatagramPacket(sendData, sendData.length,
↪  ipAddress, 9876);
sock.send(sendPacket);
DatagramPacket receivePacket = new
↪  DatagramPacket(receiveData, receiveData.length);
sock.receive(receivePacket);
int rcvLen = receivePacket.getLength();
String modifiedSentence = new
↪  String(receivePacket.getData(), 0, rcvLen,
↪  "UTF-8");
System.out.println("FROM SERVER: " +
↪  modifiedSentence);
sock.close();
```

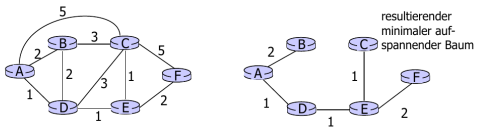
UDP-Server

```
DatagramSocket sock = new DatagramSocket(9876);
byte[] sendData = new byte[1024];
byte[] receiveData = new byte[1024];
while (true) {
    DatagramPacket receivePacket = new
    ↪  DatagramPacket(receiveData,
    ↪  receiveData.length);
    sock.receive(receivePacket);
    int rcvLen = receivePacket.getLength();
    String sentence = new
    ↪  String(receivePacket.getData(), 0, rcvLen,
    ↪  "UTF-8");
    InetAddress ipAddress =
    ↪  receivePacket.getAddress();
    int port = receivePacket.getPort();
    String capitalizedSentence =
    ↪  sentence.toUpperCase();
    sendData =
    ↪  capitalizedSentence.getBytes("UTF-8");
    DatagramPacket sendPacket = new
    ↪  DatagramPacket(sendData, sendData.length,
    ↪  ipAddress, port);
    sock.send(sendPacket);
}
```

Beispiel für den Ablauf des Dijkstra-Verfahrens:

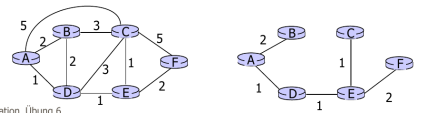
		jeweils D(·), p(·)				
Schritt	V´	B	C	D	E	F
0	A	2,A	5,A	1,A	∞,-	∞,-
1	A,D	2,A	4,D		2,D	∞,-
2	A,D,B		4,D		2,D	∞,-
3	A,D,B,E				3,E	4,E
4	A,D,B,E,C					4,E
5	A,D,B,E,C,F					

Notation! Sobald Knoten in V´, jeweilige Zelle frei lassen, da sich ihr Inhalt nicht mehr ändert



Routing: Link-State-Routing Forward-Search

Schritt	bestätigteListe (Ziel, Distanz, nexthop)	vorläufigeListe
0		(A,0,-)
1	(A,0,-)	(B,2,B), (C,5,C), (D,1,D)
2	(A,0,-), (D,1,D)	(B,2,B), (C,4,D), (E,2,D)
3	(A,0,-), (D,1,D), (B,2,B)	(C,4,D), (E,2,D)
4	(A,0,-), (D,1,D), (B,2,B), (E,2,D)	(C,3,D), (F,4,D)
5	(A,0,-), (D,1,D), (B,2,B), (E,2,D), (C,3,D)	(F,4,D)
6	(A,0,-), (D,1,D), (B,2,B), (E,2,D), (C,3,D), (F,4,D)	



- **Link-State-Routing**
  - zentrales Verfahren
  - Skalierbarkeit beschränkt durch Komplexität des Dijkstra-Verfahrens bei n Knoten: $O(n^2)$, effiziente Implementierungen schaffen $O(n \log n)$
  - Nachrichtenaustausch: O(ne) bei e Kanten
  - Robustheit: Router können fehlerhafte **Kanten**-Kosten weiter geben
- **Distanzvektor-Routing**
  - verteilter Algorithmus
  - Skalierbarkeit beschränkt durch Konvergenzprobleme bei Zyklen
  - Nachrichtenaustausch beschränkt auf direkte Nachbarn, aber mehrere Iterationen
  - Robustheit: Router können fehlerhafte **Pfad**-Distanzen bekannt geben, Fehlerfortpflanzung möglich
- **allgemein gilt**
  - (von der aktuellen Netzlast abhängige) dynamische Kostenmetriken führen zu instabilem Verhalten und haben sich nicht bewährt

Beispiel für exterior Routing: BGP

4.1.7 IPv6

IPv6 — ursprünglich *Internet Protocol Next Generation (IPng)* — ist initiiert worden wegen des Adressraumproblems von IPv4. Dabei hat man gleich noch weitere Probleme gelöst:

- Header haben nun feste Länge; dies ermöglicht ein schnelles Weiterleiten
- Es gibt keine Fragmentierung mehr, sondern Pakete werden einfach verworfen falls sie größer als die MTU sind.
- Es gibt keine Prüfsumme mehr, die Fehlererkennung erfolgt in höheren Schichten.
- Man kann zusätzliche Optionen außerhalb über Headerketten angeben.
- Die Dienstgüte wird nun unterstützt.
- Informationssicherheit

DHCP *server discovery* von `0.0.0.0` an `255.255.255.255` (Broadcast)

DHCP *server offers* von der IP-Adresse des DHCP-Servers an `255.255.255.255` (Broadcast)

DHCP *request* von `0.0.0.0` an die IP-Adresse des ausgewählten DHCP-Servers (Unicast)

DHCP *ACK* vom ausgewählten DHCP-Server an `255.255.255.255`, enthält die IP-Adresse `yiaddr` und gegebenenfalls weitere Parameter.

Vor- und Nachteile einer klassenbasierten Adressierung **Vorteil:** Es gibt **selbstidentifizierende Adressen**, also Adressen bei denen an den ersten Bits erkannt wird, um welche Klasse es sich handelt.
Vorteil: Weiterleitungstabellen benötigen nur einen Netzwerkteil der Adresse und können somit klein gehalten werden.
Nachteil: Es gibt eine feste Zuordnung von Netzwerken. Wenn ein Rechner also „umzieht", so muss seine IP-Adresse angepasst werden.
Nachteil: C-Netze erlauben nur wenige Hosts, B-Netze hingegen sehr viele Hosts (8 zu 16 Bit). Dies führt unweigerlich zu einer Verschwendung. Größere Organisationen bemühen sich um B-Netze, nutzen diese dann aber oftmals nicht aus.
Nachteil: Der Adressraum ist zu klein. 2011 hat ICANN die letzten verbleibende IPv4-Adressen ausgegeben. Die Lösung hierzu ist IPv6, ein *„Pflaster"* ist NAT.

Pro und Contra von NAT

Nachteil: Das Schichtenprinzip wird hierbei verletzt, da sich der Router mit Hosts beschäftigt und Ports für Dienste zwischen Transport- und Anwendungsschicht gedacht sind.
Nachteil: Eingriff in die Ende-zu-Ende-Verbindung
Nachteil: Hosts hinter NAT können nicht als Server auftreten (NAT-Traversal benötigt Tricks)
Vorteil: Interne Änderungen sind ohne externe Auswirkungen möglich und eine bessere Abschirmung wird erzielt.

Berechnung der Subnetzadresse:

→ Gegeben: 32 bit Adresse a.b.c.d / n
⇒n gibt die Anzahl der wichtigen Bits von links an
⇒alle Bits hinter n werden zu 0
⇒168.128.20.4 / 20 ⇒ 168.128.16.0

Subnetzmaske:

→ Gegeben: 32 bit Adresse a.b.c.d / n
⇒die ersten n Bits werden zu 1, Rest wird zu 0
⇒168.128.20.176 / 25 ⇒ 255.255.255.128
⇒Ergebnis kommt meist so in Weiterleitungstabelle

DNS

- **Anfragearten**
  - iterativ
    - Antwort: anderer Server, der Namen evtl. auflösen kann (oder keine Antwort)
    - NS- und A-Datensatz
    - Antwort wird sofort geliefert, es muss keine Information gespeichert werden, gut für hochfrequentierte Server
  - rekursiv
    - Antwort: Auflösung des Namens, die u.U. von anderen Servern geholt wird
    - A-Datensatz
    - bei Anfrage an einen anderen Server muss die Information gespeichert werden
- **Cross-Layer Optimierung**
  - die saubere Trennung in Schichten wird in der Realität oft nicht eingehalten, z.B. werden zur Steigerung der Effizienz Mechanismen der Bitübertragungs- und der Sicherungsschicht gekoppelt

E-Mail: From, To, Subject nochmal getrennt im Rumpf, der eig Inhalt folgt nach einer Leerzeile

Vorteile Go-Back-N: Kumulative ACKs, Sender nur einen Timer, Empfänger keinen Puffer
Vorteile Selective Repeat: Weniger Sendewiederholungen

Content Delivery Network (CDN):

1. Engpässe auf der Strecke zum Nutzer werden minimiert
2. keine mehrfache Übertragung desselben Inhalts über eine Strecke
3. Kein Single Point of Failure

Warum werden beim Alternating-Bit-Protokoll keine NAK0- und NAK1-Nachrichten benötigt?

Bei fehlerhaftem Empfang wird das letzte ACK zurückgesendet, damit weiß der Sender, dass das letzte Paket fehlerhaft übertragen wurde. Dieses „doppelte" ACK übernimmt hier implizit die Aufgabe der NAK-Nachrichten.

Pseudo-Header

- es ist in Wirklichkeit noch ein bisschen komplizierter ...
- Pseudo-Header enthält Quell- und Ziel-IP-Adresse, Protokollnummer (17 für UDP) und Segmentlänge
- UDP des Senders schreibt zunächst 0 in das Checksum-Feld, erstellt einen Pseudo-Header und berechnet die Prüfsumme zusammen für das UDP-Segment und den Pseudo-Header
- diese Prüfsumme wird in das Checksum-Feld geschrieben
- dann wird das Segment und der Pseudo-Header an IP weitergereicht
- UDP des Empfängers erhält von IP das UDP-Segment und den Pseudo-Header, schreibt 0 in das Checksum-Feld und berechnet die Prüfsumme für Segment und Pseudo-Header
- Vorteil: die Kontrolle der Prüfsumme erkennt auch Fehler in den IP-Adressen, z.B. fehlgeleitete Segmente
- Nachteil: Verletzung des Schichtenprinzips (aber nur auf Endsystem)

UDP Prüfsummenrechnung:
Sender:
→ 1. Prüfsummenfeld wird mit 16 0er vorinizialisiert
→ 2. UDP-Paket-Daten werden in 16-Bit-Blöcke aufgeteilt
⇒ Falls letzter Block < 16 Bit, mit 0er auffüllen
→ 3. Addiere nun alle 16-Bit Blöcke mit Übertrag auf
⇒ Übertrag ergibt einfach ein Prüfsumme + 1
→ 4. Addiere dieses Ergebnis auf das Prüfsummenfeld
→ 5. Pseudeheader in 16-Bit-Blöcke und Summe bilden
→ 6. Pseudoheader Summe auf Prüfsummenfeld addieren
→ 7. Prüfsumme invertieren, falls nicht nur 1er bzw. 0er
Empfänger:
→ 1. Selben Algorithmus wie Sender durchführen
⇒ Keine Invertierung der Prüfsumme vornehmen!
→ 2. Empfänger und Sender Summen addieren
⇒ Falls Ergebnis der Summe 0: alle richtig
⇒ Ansonsten neues versenden anfordern (unbemerkt)