

# **KN I - Zusammenfassung**

Lorenz Prochaska, Jonas Pablo Stienen

Wintersemester 08/09

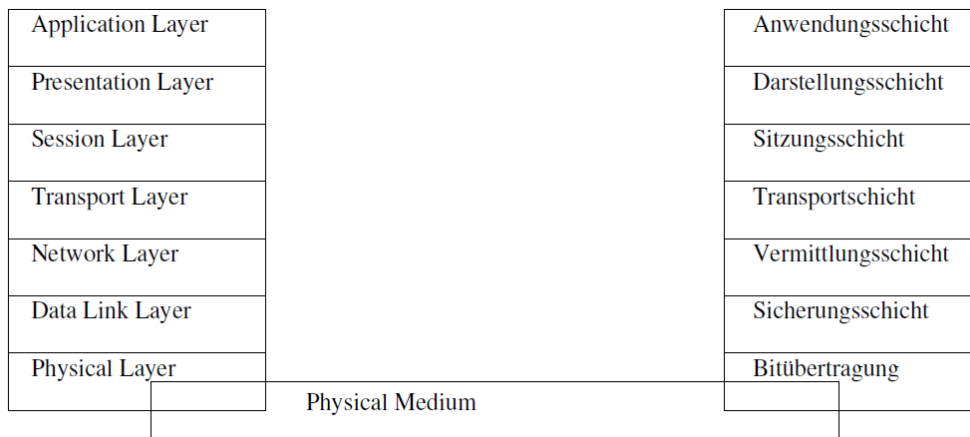
# Inhaltsverzeichnis

<b>1</b>	<b>ISO/OSI Referenzmodell Übersicht</b>	<b>4</b>
1.1	Schichten (Layer) . . . . .	4
1.2	Grundlegende Prinzipien . . . . .	4
<b>2</b>	<b>Bitübertragungsschicht (Physical Layer)</b>	<b>5</b>
2.1	Codierungsarten . . . . .	5
2.2	Kanalkapazität (channel capacity) . . . . .	6
2.2.1	Theorem (Nyquist) . . . . .	6
2.2.2	Theorem (Shannon-Hartley) . . . . .	6
2.3	4B/5B Kodierung . . . . .	6
<b>3</b>	<b>Sicherungsschicht (Data Link Layer)</b>	<b>7</b>
3.1	Byte-Stuffing . . . . .	7
3.2	Bit-Stuffing . . . . .	7
3.3	Cyclic Redundancy Check (CRC) . . . . .	7
3.4	Hamming Bedingung . . . . .	7
3.5	Codierverfahren nach Hamming . . . . .	7
3.6	Flusskontrolle (flow control) . . . . .	7
3.6.1	Sliding Window protocol . . . . .	7
3.6.2	Automatic Repeat Request (ARQ) . . . . .	8
3.6.3	ARQ Typen . . . . .	8
3.6.4	Mehrfachzugriff (multiple access) . . . . .	8
3.7	High-Level Data Link Control (HDLC) . . . . .	9
3.8	Point-to-Point protocol (PPP) . . . . .	10
3.9	Ethernet protocol . . . . .	11
3.9.1	Ethernetverkabelung . . . . .	11
3.9.2	Kanaleffizienz . . . . .	12
3.9.3	Ethernet switches . . . . .	15
3.10	Address Resolution Protocol (ARP) . . . . .	16
<b>4</b>	<b>Vermittlungsschicht (Network Layer)</b>	<b>17</b>
4.1	Art der Übertragung . . . . .	17
4.1.1	Leitungsvermittlung (Virtual Circuit Switching) . . . . .	17
4.1.2	Paketvermittlung (Packet Switching) . . . . .	17
4.2	Netzklassen und Classless Inter-Domain Routing (CIDR) . . . . .	17
4.3	Network Address Translation (NAT) . . . . .	17
4.4	Internet Protocol (IP) . . . . .	18
4.4.1	IPv4 . . . . .	18
4.4.2	IPv6 . . . . .	19
4.4.3	Adressen . . . . .	19
4.4.4	Erweiterungsheader . . . . .	20
4.5	Internet Control Message Protocol (ICMP) . . . . .	22
4.6	Routing . . . . .	22
4.6.1	Anforderungen . . . . .	22
4.6.2	Routing Verfahren . . . . .	23
4.6.3	Flooding . . . . .	23
4.6.4	Multidestination Routing . . . . .	23
4.6.5	Reverse Path Forwarding . . . . .	23
4.6.6	Bellmann-Ford Algorithmus . . . . .	24
4.6.7	Dijkstra Algorithmus . . . . .	24

4.6.8	Bewertung von Netzen . . . . .	24
4.7	Internet Group Management Protocol (IGMP) . . . . .	25
4.8	Open Shortest Path First (OSPF) . . . . .	26
<b>5</b>	<b>Transportschicht (Transport Layer)</b>	<b>27</b>
5.1	Funktionen . . . . .	27
5.1.1	Verbindungsaufbau durch three-way handshake . . . . .	27
5.1.2	Verbindungsabbau . . . . .	27
5.1.3	Pseudoheader zur Prüfsummenberechnung . . . . .	28
5.2	Transmission Control Protocol (TCP) . . . . .	29
5.2.1	Bezeichnungen . . . . .	30
5.2.2	Socket und Connection . . . . .	30
5.2.3	Verbindungsaufbau, Verbindungsabbau und Datenübertragung . . . . .	30
5.2.4	Window Management . . . . .	31
5.2.5	Reset (RST) . . . . .	32
5.2.6	TCP Optionen . . . . .	33
5.2.7	TCP Flusskontrolle . . . . .	34
5.3	User Datagram Protocol (UDP) . . . . .	36
5.3.1	UDP Rahmen . . . . .	36
<b>6</b>	<b>Anwendungsschicht (Application Layer)</b>	<b>37</b>
6.1	Domain Name System . . . . .	37
6.1.1	DNS Rahmen . . . . .	37
6.1.2	Transport . . . . .	39
6.1.3	Auflösung von Namen . . . . .	39
6.1.4	Reverse Lookup . . . . .	39
6.1.5	Load Balancing . . . . .	39
6.2	Hypertext Transfer Protocol (HTTP) . . . . .	41
6.2.1	Aufbau einer HTTP Nachricht . . . . .	41
6.2.2	Kodierung von HTTP Nachrichten . . . . .	42
6.2.3	Uniform Resource Identifier (URI) . . . . .	42
6.2.4	HTTP Header . . . . .	43
6.2.5	HTTP Proxies . . . . .	45
6.2.6	HTTP Authentication . . . . .	46
6.2.7	Cookies . . . . .	47
6.3	TLS . . . . .	48
6.3.1	RSA Verfahren . . . . .	48
6.3.2	Digitale Signatur . . . . .	49
6.3.3	Zertifikate . . . . .	49
6.4	Real-time Transport Protocol (RTP) . . . . .	50
6.4.1	RTP Rahmen . . . . .	50
6.4.2	Real-time Transport Control Protocol (RTCP) . . . . .	52
6.5	Session Description Protocol (SDP) . . . . .	57
6.6	Session Initiation Protocol (SIP) . . . . .	59
6.6.1	SIP Dialog . . . . .	59
6.6.2	Routing und Forwarding . . . . .	59
6.6.3	Verarbeitung der Response . . . . .	61
6.7	Robust Header Compression (ROHC) . . . . .	63
6.7.1	Statemachine des Unidirectional Mode . . . . .	64
6.7.2	Statemachine des Bidirectional Optimistic Mode . . . . .	64
6.7.3	Statemachine des Bidirectional Reliable Mode . . . . .	65

# 1 ISO/OSI Referenzmodell Übersicht

## 1.1 Schichten (Layer)



## 1.2 Grundlegende Prinzipien

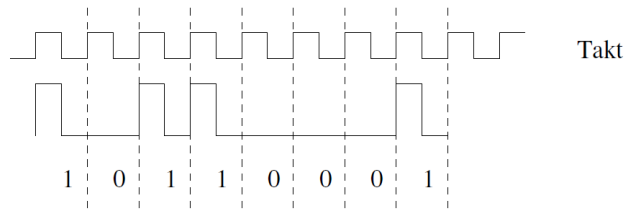
1. Dienst (service)
2. Schnittstelle (interface)
3. Protokoll (protocol)

## 2 Bitübertragungsschicht (Physical Layer)

### 2.1 Codierungsarten

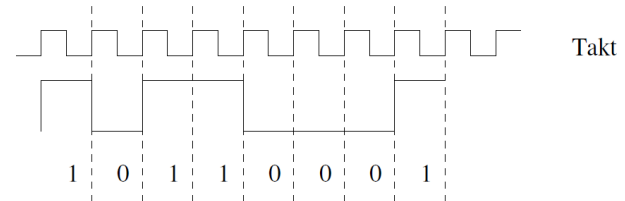
RZ (Return to Zero):

**Keine** automatische Taktrückgewinnung,  
**nicht** gleichspannungsfrei



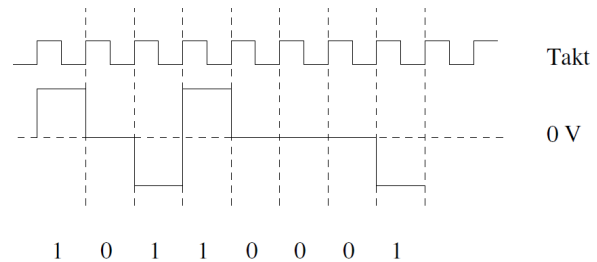
NRZ (No Return to Zero):

**Keine** automatische Taktrückgewinnung,  
**nicht** gleichspannungsfrei

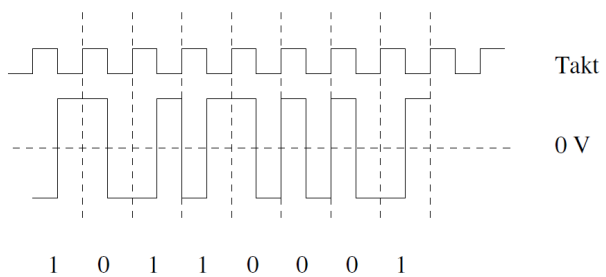


AMI (Bipolare Kodierung):

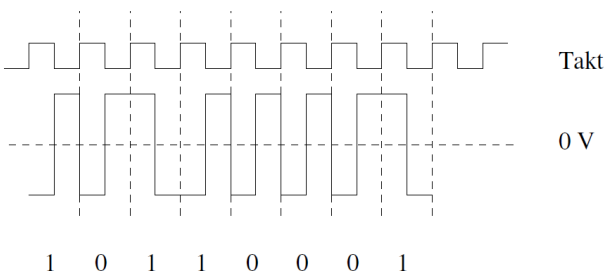
**Keine** automatische Taktrückgewinnung,  
gleichspannungsfrei



Manchester Kodierung:  
automatische Taktrückgewinnung,  
gleichspannungsfrei



Differentielle Manchester Kodierung:  
automatische Taktrückgewinnung,  
gleichspannungsfrei



## 2.2 Kanalkapazität (channel capacity)

### 2.2.1 Theorem (Nyquist)

In einem bandbegrenzten, *störungsfreien* Übertragungskanal mit Bandbreite  $B$  [Hz] und  $L$  diskreten Signalstufen ist die maximale Datenübertragungsrate  $C_N$

$$C_N = 2 \cdot B \cdot \log_2(L) \quad \left[ \frac{\text{bit}}{s} \right]$$

### 2.2.2 Theorem (Shannon-Hartley)

In einem bandbegrenzten, *gestörten* Übertragungskanal mit Bandbreite  $B$  [Hz] ist die maximale Datenübertragungsrate  $C_S$

$$C_S = B \cdot \log_2\left(1 + \frac{S}{N}\right) \quad \left[ \frac{\text{bit}}{s} \right]$$

wobei  $\frac{S}{N}$  das Signal-Rausch-Verhältnis bezeichnet.

## 2.3 4B/5B Kodierung

Um genügend Amplitudenwechsel sicherzustellen, werden 4 Bit des Eingangsstroms auf 5 Bit abgebildet.

Hex	4B	5B
0	0000	11110
1	0001	01001
2	0010	10100
3	0011	10101
4	0100	01010
5	0101	01011
6	0110	01110
7	0111	01111
8	1000	10010
9	1001	10011
A	1010	10110
B	1011	10111
C	1100	11010
D	1101	11011
E	1110	11100
F	1111	11101

### 3 Sicherungsschicht (Data Link Layer)

#### 3.1 Byte-Stuffing

Ein Byte (nicht das Flagbyte) wird als sogenanntes Escape Zeichen (ESC) gewählt. Dieses Zeichen wird nach Berechnung der Checksum vor jedem Flagbyte und vor jedem Escape Zeichen in den Datenstrom eingefügt (nicht vor den tatsächlichen Flags ; ) ).

#### 3.2 Bit-Stuffing

Auf eine beliebige Anzahl von Einsen wird senderseitig immer eine 0 eingefügt. Der Empfänger löscht jede Null, die auf die vorgeschriebene Anzahl von Einsen folgt.

#### 3.3 Cyclic Redundancy Check (CRC)

Sei  $p(x)$  die zu übertragende Information und  $g(x)$  ein Generatorpolynom mit Grad  $n$ . Die Checksum ist der Rest der Division  $\frac{p(x) \cdot x^n}{g(x)}$

#### 3.4 Hamming Bedingung

$r$  sei die Länge der Checksum (Rest der Division). Um  $k$  Bitfehler korrigieren zu können, muss gelten:

$$\sum_{i=0}^k \binom{n+r}{i} \leq 2^r$$

#### 3.5 Codierverfahren nach Hamming

Einzelfehlerkorrektur: Seien  $n = 4$  und  $r = 3$  und wir benutzen gerade Parität.

Spalte (dezimal)	1	2	3	4	5	6	7
Spalte (binär)	001	010	011	100	101	110	111
Prüfbit	↓	↓		↓			
1011	0	1	1	0	0	1	1
0110	1	1	0	0	1	1	0
0001	1	1	0	1	0	0	1

#### 3.6 Flusskontrolle (flow control)

##### 3.6.1 Sliding Window protocol

Rahmen werden mit  $\text{mod } n$  durchnummeriert. Fenster- bzw. Buffergröße bei Empfänger:  $n-1$ . Sender kennt Fenstergröße vom Empfänger und sendet  $n-1$  Rahmen ohne auf ein ACK warten zu müssen. Im ACK steht die Nummer des ersten noch nicht bearbeiteten Rahmens.

ACK( $i$ ) bedeutet, dass der Sender Pakete bis Rahmennummer  $i-2$  wieder senden darf.

### 3.6.2 Automatic Repeat Request (ARQ)

Wird kein ACK vom Empfänger gesendet, wird nach Zeitüberschreitung erneut gesendet.

Erfolgt ein NACK des Empfängers, ist das Paket fehlerhaft gewesen.

Sender kann jeden nicht quittierten Rahmen erneut senden und behält so lange eine Kopie, bis ACK erfolgt. Ein ACK( $k$ ) quittiert alle Rahmen bis  $k - 1$ .

### 3.6.3 ARQ Typen

<b>Stop-and-Wait ARQ</b>	Jedes NACK oder jedes verlorene Paket (ACK/NACK/Datenrahmen), durch Zeitüberschreitung detektiert, löst erneute Übertragung aus.
<b>Go-Back-<math>n</math> ARQ</b>	Benutzt <i>Sliding Window protocol</i> . Es werden nur fehlerfreie Rahmen in richtiger Reihenfolge akzeptiert. Jedes NACK( $k$ ) löst erneute Übertragung der Rahmen ab $k$ aus und quittiert gleichzeitig alle Rahmen bis $k - 1$ . NACK wird bei fehlerhaftem Rahmen oder bei fehlenden Rahmen gesendet. Nach Zeitüberschreitung (ACK/NACK verloren) sendet der Sender alle Pakete, die im Speicher sind noch einmal.
<b>Selective Repeat ARQ</b>	Wird ein Rahmen fehlerhaft oder nicht übertragen, kann der Empfänger mit einem NACK( $k$ ) den Rahmen erneut anfordern und quittiert gleichzeitig alle Rahmen bis $k - 1$ . Bereits gesendete Rahmen größer $k$ werden gespeichert.
<b>Bitmap ARQ</b>	Erweitert Selective Repeat ARQ, um es effizienter zu gestalten (hauptsächlich Bandbreite), siehe letzte Folie S. 43f
<b>Hybrid ARQ</b>	Hybrid ARQ kombiniert ARQ um intelligente Kanalcodierung, letzte Folie S. 45f

### 3.6.4 Mehrfachzugriff (multiple access)

**Carrier Sense Multiple Access/ Collision Detection (CSMA/CD)** wird von Ethernet benutzt. Vor dem Senden wartet der Adapter, dass die Leitung für 96 Bit frei ist. Solange gesendet wird, wird gleichzeitig detektiert, ob eine andere Station sendet. Wird erkannt, dass eine andere Station sendet, wird ein 48 Bit Störsignal abgeschickt und die Übertragung abgebrochen. Bei  $n$  Kollisionen hintereinander wartet der Sender zwischen 0 und  $2^{\min n, 10} - 1$  mal die Zeiteinheit 512 Bit ab und überträgt dann erneut (Binary Exponential Backoff Algorithm).

### 3.7 High-Level Data Link Control (HDLC)

Flag	Address	Control	Data	FCS	Flag
01111110	1 or more Byte	1 or 2 Byte	variable	2 or 4 Byte	01111110

**Address** normalerweise 8 Bit mit der Adresse des Zielterminals  
**Control** I-Frames, S-Frames und U-Frames (s.u.)  
**Information** variable Länge, 0 Bit oder mehr, i.d.R. vielfaches von 8  
**Frame Check Sequence** 16 oder 32 bit CRC Prüfsumme  
 (wird ausgehandelt oder festgelegt)

#### Control Field

7	6	5	4	3	2	1	0	
N(R) Receive sequence no.		P/F	N(S) Send sequence no.		0	1		<b>I-frame</b>
N(R) Receive sequence no.		P/F	type	0		1		<b>S-frame</b>
type		P/F	type	1		1		<b>U-frame</b>

#### Information frames

für Nutzdaten und Flusssteuerungsinformationen (huckepack)

#### Supervisory frames

zur Steuerung des Datenflusses (falls huckepack nicht möglich)

#### Unnumbered frames

zur Steuerung der Verbindung.

HDLC Datenrahmen werden **bitorientiert** übertragen

### 3.8 Point-to-Point protocol (PPP)

Flag	Address	Control	Protocol	Data	FCS	Flag
01111110	11111111	00000011	2 Byte	variable	2 or 4 Byte	01111110

<b>Protocol</b>	16 Bit Protokollspezifikationen für die oberen Schichten (z.B. IP-Datagramm, LCP, NCP)
<b>Data</b>	Variable Anzahl an Daten, mindestens 1500 Byte, wird aufgefüllt durch Padding Bytes
<b>Checksum</b>	üblicherweise 16 Bit Prüfsumme mit CRC-CCITT Generatorpolynom (manchmal 32 Bit)
<b>Bytestuffing</b>	mit Escape Zeichen 01111101, das 6. Bit des darauf folgenden Bytes (LSB first) wird invertiert (XOR 00100000)
<b>Paketgröße</b>	beträgt 1500 Bytes, falls nicht anders mittels Link Control Protocol (LCP) vereinbart.

### 3.9 Ethernet protocol

Preamble	SOF	Destination	Source	Type/Length	Data	FCS	IFG
7 Byte	1 Byte	6 Byte	6 Byte	2 Byte	46 - 1500 Byte	4 Byte	12 Byte

**Länge: 26 Byte (ohne Interframegap)**

<b>Preamble</b>	7 mal das Byte 10101010 zur Synchronisation
<b>Start of Frame Byte</b>	10101011 zur Markierung des Anfangs des Ethernetrahmens
<b>Destination</b>	MAC-Adresse der Zielschnittstelle
<b>Source</b>	MAC-Adresse der Quellschnittstelle
<b>Length/Type</b>	Typ (ab 0x05dc) oder Länge (bis 0x05dc) der Daten im Informationsfeld
<b>Padbytes</b>	Das Informationsfeld muss mindestens 46 Byte lang sein ⇒ Padding
<b>Checksum</b>	32 bit Prüfsumme

Nach dem Rahmen kommt ein 12 Byte großer **InterFrameGap**.

#### 3.9.1 Ethernetverkabelung

Name	Kabel	max. Segmentlänge	Vorteile
10Base-T	twisted pair	100m	
10Base-F	fibre optics	2000m	
100Base-T4	twisted pair	100m	Cat 3 UTP
100Base-TX	twisted pair	100m	Cat 5 UTP voll duplex
100Base-FX	fibre optics	2000m	voll duplex
1000Base-SX	fibre optics	550m	Multimodefaser (50, 62,5 $\mu\text{m}$ )
1000Base-LX	fibre optics	5000m	Single-(10 $\mu\text{m}$ ) oder Multimodefaser (50, 62,5 $\mu\text{m}$ )
1000Base-CX	twisted pair	25m	2 shielded twisted pair
1000Base-T	twisted pair	100m	4 unshielded twisted pair

Die Zahl vor 'Base' entspricht der Datenrate  $R$  der Übertragung in  $\frac{\text{Mbit}}{\text{s}}$ . Ethernet verwendet immer die Manchesterkodierung mit  $+0,85\text{V}$  highlevel und  $-0,85\text{V}$  lowlevel Spannung. Die Paketgröße  $L$  muss mindestens so groß sein, dass das Paket  $2t_{prop}$  lang gesendet wird, damit eine Störung des Signals einwandfrei festgestellt werden kann.

$$L = R \cdot 2 \cdot t_{prop} \quad \text{mit} \quad t_{prop} = \frac{d}{v}$$

$t_{prop}$  .. Dauer bis das erste Bit eines Pakets am Empfänger angekommen ist

$d$  .. Abstand vom Sender zum Empfänger

$v$  .. Ausbreitungsgeschwindigkeit auf dem Medium (Lichtgeschw. oder Kupfergeschw. (=  $\frac{2}{3}c_0$ ))

### 3.9.2 Kanaleffizienz

Die Kanaleffizienz  $U$  berechnet sich aus der Übertragungszeit  $t_{frame}$ , die ein Sender braucht, um ein Frame komplett auf die Leitung zu übertragen und der Zeit, die für die komplette Transaktion benötigt wird. Wenn kein ARQ benutzt wird, ist die Zeit der gesamten Transaktion:  $t_{frame} + t_{prop}$ . Bei Stop 'n' Wait beträgt die Zeit der Transaktion  $t_{frame} + t_{prop} + t_{ACK} + t_{prop}$ , wobei oft  $t_{ACK} = 0$  gewählt wird. Bei Selective Repeat und Go-Back-N muss beachtet werden, dass mehrere Frames geschickt werden können bevor ein ACK geschickt wird. Das heißt die Zeit, die die  $n$  Frames brauchen um auf die Leitung zu kommen beträgt  $n \cdot t_{frame}$ . Für das Verhältnis von  $t_{frame}$  und  $t_{prop}$  wird oft  $a = \frac{t_{prop}}{t_{frame}}$  eingesetzt.  $a$  gibt somit direkt an wieviele Rahmen auf die Leitung passen. Für die Übertragungszeit  $t_{frame}$  gilt:

$$t_{frame} = \frac{L}{R}$$

$L$  ist die Länge des Frames und  $R$  die Datenrate (Bandbreite). Für die Ausbreitungsverzögerung  $t_{prop}$  gilt:

$$t_{prop} = \frac{d}{v}$$

$d$  ist die Entfernung der Stationen und  $v$  ist die Ausbreitungsgeschwindigkeit auf dem Medium. Es ergibt sich für  $a$ :

$$a = \frac{R \cdot d}{L \cdot v}$$

Die Kanaleffizienz kann auch wie folgt berechnet werden:

$$U = \frac{T}{R}$$

wobei  $T$  der Durchsatz ist.

# Übersicht zu Kanaleffizienz bei fehlerlosen Übertragung ohne ARQ

$$U = \frac{t_{frame}}{t_{frame} + t_{prop}} = \frac{1}{1 + a}$$

## Stop 'n' Wait

$$U = \frac{t_{frame}}{t_{frame} + t_{prop} + t_{ACK} + t_{prop}}$$

falls  $t_{ACK} = 0$  gilt, folgt:

$$U = \frac{1}{1 + 2a}$$

## Go-Back-N

$$U = \frac{n \cdot t_{frame}}{t_{frame} + t_{prop} + t_{ACK} + t_{prop}}$$

falls  $t_{ACK} = 0$  gilt, folgt:

$$U = \frac{n}{1 + 2a}$$

Erwartungswert:

$$N_R = \frac{K \cdot P + 1 - P}{1 - P}$$

wobei  $K$  die Anzahl der Rahmen ist, die erneut übertragen werden müssen, und es gilt:  $K = 2a + 1$  falls  $W \geq 2a + 1$ , und  $K = W$  falls  $W < 2a + 1$ ,  $W$  ist die Window Size.

## Selective Repeate

$$U = \frac{n \cdot t_{frame}}{t_{frame} + t_{prop} + t_{ACK} + t_{prop}}$$

falls  $t_{ACK} = 0$  gilt, folgt:

$$U = \frac{n}{1 + 2a}$$

Erwartungswert:

$$N_R = \frac{1}{P}$$

## Kanaleffizienz bei fehlerbehafteten Übertragung

In einem Netzwerk befinden sich  $N$  Stationen. Die Wahrscheinlichkeit, dass genau  $k$  Stationen senden beträgt

$$P = \binom{N}{k} \cdot p^k \cdot (1-p)^{N-k} \quad \text{mit } p \text{ .. Sendewahrscheinlichkeit einer Station}$$

Speziell, wenn nur eine Station sendet

$$P = \binom{N}{1} \cdot p \cdot (1-p)^{N-1} = N \cdot p \cdot (1-p)^{N-1}$$

$P$  wird maximal, wenn die Sendewahrscheinlichkeit gleichverteilt ist, d.h. es muss  $p = \frac{1}{N}$  gelten. Im Grenzfall  $N \rightarrow \infty$  beträgt  $P = \frac{1}{e}$ . Die Wahrscheinlichkeit, dass nach  $i-1$  gestörten Frames ein ungestörter folgt, beträgt:

$$P \cdot (1-P)^{i-1}$$

somit ergibt sich für die mittlere Anzahl von Übertragungsversuchen ( $N_R$ ), die für eine erfolgreiche Übertragung erforderlich sind, der Erwartungswert

$$E(N_R) = \sum_{i=1}^{\infty} i \cdot P \cdot (1-P)^{i-1} = \frac{1}{P}$$

Für die mittlere Anzahl der gescheiterten Übertragungsversuche  $N_F$  gilt dann der Erwartungswert

$$E(N_F) = \sum_{i=1}^{\infty} i \cdot P \cdot (1-P)^i = \frac{1-P}{P} = \frac{1}{P} - 1$$

Allgemein berechnet sich die *Kanaleffizienz* mit Hilfe der Kanaleffizienz der fehlerlosen Übertragung  $\tilde{U}$  zu

$$U = \frac{1}{E(N_R)} \cdot \tilde{U}$$

Es folgt, dass zu einer Effizienzsteigerung die Frames länger und die Ausbreitungsgeschwindigkeit erhöht werden müssen ( $L, v$  groß) und die Bandbreite (!) und Kabellänge verringert werden muss ( $R, d$  klein).

## Länge einer Transaktion (ohne Flusststeuerung)

Die Zeit  $t_{tot}$  setzt sich zusammen aus der gesamten Übertragungszeit  $t_{trans,tot}$ , die der Sender braucht um alle Daten auf den Kanal zu übertragen, der Ausbreitungsverzögerung  $t_{prop,tot}$  und der Verarbeitungszeit der Router  $t_{proc}$ :

$$t_{trans,tot} = \frac{n \cdot L \cdot 8}{R}$$

$n$  ist die Anzahl der Rahmen und  $L$  die Größe der Rahmen. Die 8 braucht man, da die Datenrate in  $\frac{\text{bit}}{\text{s}}$  gegeben ist.

$$t_{prop,tot} = \frac{d_{tot}}{v}$$

$d_{tot}$  ist die gesamte Länge des Kanals. Die Länge der Transaktion (ohne ACKs) beträgt somit:

$$t_{tot} = t_{trans,tot} + t_{prop,tot} + t_{proc}$$

Wenn  $k$  Router bzw. Knoten in der Übertragungstrecke sind muss beachtet werden, dass ein Router bzw. Knoten erst mit der Verarbeitung der Daten anfangen kann sobald der erste Rahmen komplett angekommen ist. Das bedeutet es gilt für  $t_{tot}$ :

$$t_{tot} = t_{trans,tot} + t_{prop,tot} + t_{proc} + kt_{frame}$$

### 3.9.3 Ethernet switches

*Ethernet switches* puffern Ethernet Pakete, um sie an den relevanten Adapter (Schnittstelle) weiterzuleiten. I.d.R. werden alle Pakete bei Ankunft im RAM gespeichert und es wird geprüft, ob die Adresse zu einem der Adapter gehört. Ist dem nicht so, wird das Paket an alle Adapter verteilt, sonst direkt an den Empfänger gesendet. Dieses Design vermeidet Kollisionen gänzlich und erlaubt Vollduplexübertragung parallel für alle Ports, da jeder Port eine eigene Kollisionsdomäne darstellt. Oft werden auch Adapter an Hubs angeschlossen, hier gilt dann das übliche CSMA/CD Verfahren mit Exponential Binary Backoff Algorithmus.

### 3.10 Address Resolution Protocol (ARP)

ARP wird benötigt, wenn die Netzwerkadresse des Zielrechners bekannt ist, aber nicht die Hardwareadresse. Ziel ist es dann, die Adresse auf Sicherungsebene (i.d.R. MAC) des Zielhosts, z.B. ein Rechner oder ein Router im gleichen LAN, zu ermitteln. Dazu wird ein ARP request an alle Adapter im LAN gesendet (Broadcast). Empfängt der Host mit passender Netzwerkadresse den ARP request, antwortet er mit einem ARP reply, in dem er seine Hardwareadresse als **Sender Hardware Address** einsetzt. Wenn kein ARP reply kommt wird nach wenigen Wiederholungen abgebrochen und das Paket an eine Standard Hardwareadresse geschickt. Gratuitous ARP wird verwendet um zu prüfen, ob die eigene IP im LAN schon vorhanden ist und um sich im LAN bekannt zu machen. Dazu wird ein ARP request mit der eigenen IP gebroadcastet.

0 - 7	8 - 15	16 - 31
Hardware Type		Protocol Type
Hardware Length	Protocol Length	Operation
Sender Hardware Address		
Sender Hardware Address		Sender Protocol Address
Sender Protocol Address		Target Hardware Address
Target Hardware Address		
Target Protocol Address		

**Länge: 28 Byte**

<b>Hardware Type</b>	Art der nachgefragten Adresse, 1 für Ethernet
<b>Protocol Type</b>	Art der Netzwerkadresse. 0800 <sub>16</sub> für IP
<b>Hardware Length, Protocol Length</b>	Länge der Adressen, 6 für Ethernet, 4 für IPv4
<b>Operation</b>	Typ des Requests
<b>Sender Hardware Address</b>	Hardwareadresse des Senders, z.B. MAC (Identisch mit <i>Source</i> )
<b>Sender Protocol Address</b>	Netzwerkadresse des Senders, z.B. IP
<b>Target Hardware Address</b>	Hardwareadresse des Ziels, z.B. MAC
<b>Target Protocol Address</b>	Netzwerkadresse des Ziels, z.B. IP

ARP Rahmen werden in Ethernet-Rahmen gesendet. Der Wert im Feld Type ist 0x0806.

1. ARP Request, d.h. Netzwerk → Hardware
2. ARP Reply, Antwort
3. RARP Request, d.h. Hardware → Netzwerk (reverse ARP)
4. RARP Reply, Antwort

## 4 Vermittlungsschicht (Network Layer)

### 4.1 Art der Übertragung

#### 4.1.1 Leitungsvermittlung (Virtual Circuit Switching)

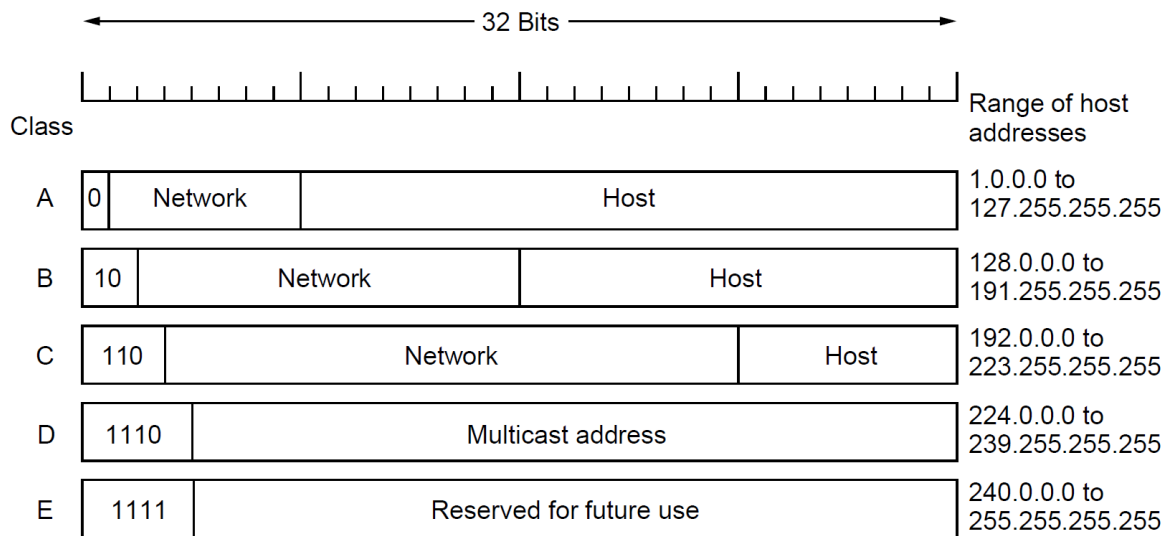
Es wird eine Verbindung aufgebaut über die alle Pakete gesendet werden. Jede bestehende Verbindung hat eine bestimmte Kennung. Am Ende wird die Verbindung abgebaut.

#### 4.1.2 Paketvermittlung (Packet Switching)

Es gibt kein Verbindungsauf- oder abbau. Alle Pakete erhalten die Adresse des Ziels und werden von jedem Router mittels einer Routingtabelle weitergeleitet. So können unterschiedliche Pfade entstehen.

### 4.2 Netzklassen und Classless Inter-Domain Routing (CIDR)

Als das Internet noch klein war und noch nicht so viele Netzwerke existierten, wurde der IP-Adressbereich in Klassen eingeteilt.



Als die IP Adressen knapp wurden, entschloss man sich die Klassen durch Subnetze zu ersetzen. Zu jeder IP Adresse wird eine Subnetzmaske angegeben. Das zugehörige Netz ergibt sich aus der logischen Und-Verknüpfung der IP Adresse und der Subnetzmaske.

### 4.3 Network Address Translation (NAT)

Oft befindet sich hinter einer IP Adresse ein kleines oder auch großes privates Netzwerk. Damit alle Hosts des privaten Netzwerks "ins Internet" können, muss der Router Adressen im IP Header ändern können.

## 4.4 Internet Protocol (IP)

### 4.4.1 IPv4

0 - 3	4 - 7	8 - 15	16 - 18	19 - 31
Version	Header Length	Type of Service	Total Length (in bytes)	
Identification			0 <i>D M</i> <i>F F</i>	Fragment Offset
Time to Live	Protocol		Header Checksum	
Source IP Address				
Destination IP Address				
Options (if any)				
Data				

**Länge: 20 Byte (ohne Optionen)**

<b>Version</b>	4 für IPv4, 6 für IPv6
<b>Header Length</b>	Länge des IP Headers inklusive Optionen gezählt in 4Byte Worten, d.h. mindestens 5 (für 20 Bytes). max. Länge des IP Headers ist 60Bytes ( $(2^4 - 1) \cdot 4$ )
<b>Type of Service</b>	Ursprünglich: Unterscheidung von Klassen von Services (Geschwindigkeit, Verlässlichkeit, Sicherheit) Heute: 2 bit explicit congestion notification, 6 bit differentiated Services
<b>Total Length</b>	Größe des gesamten Rahmens in Byte
<b>Identification</b>	Alle Fragmente eines Datagramms besitzen die selbe Identifikationsnummer
<b>DF</b>	Dont't Fragment
<b>MF</b>	More Fragments
<b>Fragment Offset</b>	Offset eines Fragments im Gesamtpaket, in <b>8Byte</b> Einheiten, D.h., das Datenfeld aller Fragmente (bis auf das letzte) muss durch 8 teilbar sein
<b>Time to Live</b>	Anzahl der max. Hops
<b>Protocol</b>	Welches Protokoll vom höheren Layer verwendet wird
<b>Header Checksum</b>	Checksum des <b>Headers</b> Der Header wird in 16 bit lange Blöcke geteilt, welche untereinander geschrieben werden (bereits vorhandene Prüfsummen werden zu 0 gesetzt) Dann wird für jede Spalte gerade Parität gebildet. Das Ergebnis ist die Prüfsumme
<b>Source IP Address</b>	IP Adresse des Senders
<b>Destination IP Address</b>	IP Adresse des Empfängers
<b>Options</b>	Protokolloptionen, Länge des Feldes wird durch die Header Length festgelegt Jede Option beginnt mit einem Byte zur Identifikation Bei manchen Optionen folgt ein Byte für die Länge der Option dann kommen die Daten der Option
<b>Data</b>	Daten

#### 4.4.2 IPv6

0 - 3	4 - 11	12 - 15	16 - 23	24 - 31
version	traffic class	flow lable		
payload length		next header	hop limit	
source address 16 Byte				
destination address 16 Byte				

**Länge: 40 Byte (ohne Extensions)**

**Länge: 48 Byte für Fragmentation (wg. Fragmentation Header)**

<b>version</b>	4 für IPv4, 6 für IPv6
<b>traffic class</b>	Unterscheidung verschiedener Klassen bzgl. zulässiger Verzögerung
<b>flow lable</b>	ermöglicht Pseudoverbindung Eine Verbindung ist eindeutig durch Flownummer, Quell- und Zieladresse bestimmt
<b>payload length</b>	Größe des Datenfeldes in Byte (ohne den Header)
<b>next header</b>	Gibt an welcher Erweiterungsheader als nächstes kommt Wenn kein Erweiterungsheader mehr folgt, steht in dem Feld das verwendete Protokoll der nächst höheren Schicht
<b>hop limit</b>	maximale Anzahl an Knoten die passiert werden dürfen
<b>source IP address</b>	IP Adresse des Senders
<b>destination IP address</b>	IP Adresse des Empfängers

#### 4.4.3 Adressen

8 Segmente mit je 2 Byte (128 bit), getrennt durch Doppelpunkt. Führende Nullen in einem Segment können weggelassen werden, genau **eine** Gruppe von leeren, aufeinanderfolgenden Segmenten kann durch 2 Doppelpunkte abgekürzt werden. Die letzten 4 Byte können als **“Dotted Notation”** geschrieben werden, d.h. wie eine IPv4 Adresse. Netzwerkpräfixe werden in CIDR Notation (Adresse/Präfix) geschrieben. Segmente, die außerhalb der Maske liegen, brauchen nicht aufgeführt werden. IPv4 Adressen können in IPv6 gemappt werden. Anwendungen (Schnittstellen) bekommen eine IPv6 Adresse bei der die letzten 4 Byte mit der IPv4 Adresse übereinstimmen, geführt von 2 Byte mit *FFFF*. Bei Routern/Geräten werden die 2 Byte *FFFF* wegelassen.

#### 4.4.4 Erweiterungsheader

Die Länge (in Bytes) jedes Erweiterungsheaders ist durch 8 teilbar, d.h. Erweiterungsheader beginnen auf 8 Byte Grenzen. Die Reihenfolge, wie die Header angehängt werden, ist festgelegt.

Type	Name
0	Hop-By-Hop Option
60	Destination Option
43	Routing Header
44	Fragment Header
51	Authentication Header
50	Encapsulating Security Payload
59	No Next Header
60	Destination Option
135	Mobility Header

#### Hop-By-Hop Option:

Muss von jedem Router ausgewertet werden. Der Header hat immer die Felder

0 - 7	8 - 15	16 - 23	24 - 31
next header	header length	option type	option length
option data			

<b>next header</b>	Gibt an, welcher Header als nächstes kommt
<b>header length</b>	Länge des Headers in 8-Byte-Einheiten, aber ohne die ersten 8 Bytes.
<b>option</b>	Für Option spezifische Angaben

Hop-by-Hop Optionen sind Typ/Länge/Wert kodierte (Tuple), nur die Typ 0 Option ist nicht so kodiert, Typ 0 und 1 dienen als Füller. Typ 194 (Jumbo Payload) signalisiert Pakete bis 4 GByte.

#### Routing Option:

Gibt an welche Router das Paket passieren muss.

0 - 7	8 - 15	16 - 23	24 - 31
next header	header length	routing type	segments left
type-specific data			

<b>next header</b>	Gibt an, welcher Header als nächstes kommt
<b>header length</b>	Länge des Headers in 8-Byte-Einheiten, aber ohne die ersten 8 Bytes.
<b>routing type</b>	Typ des Routing Headers
<b>segments left</b>	Anzahl der in der Option aufgeführten Hops
<b>type-specific data</b>	Für Typ spezifische Angaben

### Fragment Option:

Falls ein Paket zu groß für die MTU eines Routers ist. **ACHTUNG:** Diese Option muss gesetzt werden, wenn fragmentiert wird, d.h. der maximal mögliche Payload verringert sich um die Länge des Fragmentation Headers (8 Byte).

0 - 7	8 - 15	16 - 28	29,30	31
next header	reserved	fragment offset	res	M
identification				

<b>next header</b>	Gibt an, welcher Header als nächstes kommt
<b>fragment offset</b>	Offset eines Fragments im Gesamtpaket, in <b>8 Byte</b> Einheiten
<b>M-Flag</b>	Gibt an ob noch mehr Fragmente folgen
<b>identification</b>	Alle Fragmente eines Datagramms besitzen die selbe Identifikationsnummer

In IPv6 darf kein Router selbst fragmentieren, die MTU muss vorher vom Quellrouter ermittelt werden. Router senden ICMP Fehler, wenn ein Paket zu groß für die Verarbeitung ist.

### Destination Option:

Ist von jedem Router auszuwerten, sofern sie vor dem Routing Header auftritt, sonst nur vom Zielhost. Der Aufbau entspricht den Hop-By-Hop Headern.

0 - 7	8 - 15	16 - 23	24 - 31
next header	header length	option type	option length
option data			

<b>next header</b>	Gibt an welcher Header als nächstes kommt
<b>header length</b>	Länge des Headers in 8-Byte-Einheiten ohne die ersten 8 Bytes.
<b>option</b>	Für Option spezifische Angaben

Weitere Header sind der Authentication Header (s. Folie 5 S.34), der Encapsulating Security Payload (s. Folie 5 S.35) und der Mobility Header (s. Folie 5 S.36).

## 4.5 Internet Control Message Protocol (ICMP)

Das ICMP wird benutzt, um das Internet zu testen und Router zu überwachen. ICMP Pakete werden in IP Paketen gekapselt (Protocol ID 1 bei IPv4).

Type	Code	Checksum	Info
8 Bit	8 Bit	16 Bit	variabel

Alle Felder hängen vom Typ des ICMP Paketes ab. Das Info Feld enthält wenn nötig auch den Header des betroffenen IP Paketes inklusive der ersten 8 Byte des Payloads (z.B. bei time exceeded oder destination unreachable).

### ICMP Typen:

Nummer	Typ
0	destination unreachable
1	time exceeded
2	parameter problem
3	source quench
4	redirect
5	echo request
6	echo reply
7	timestamp request
8	timestamp reply

Beispiele: Folie 4 S. 41ff

Tanenbaum: Kapitel 5.6.3

## 4.6 Routing

Routing ist das Verfahren, welches sich um die Ausführung und Optimierung der Paketvermittlung in (intra) oder zwischen (inter) verschiedenen **autonomen Systemen (AS)** bemüht. Router, die autonome Systeme verbinden nennt man *Gateway Router*.

### 4.6.1 Anforderungen

- Skalierbarkeit** Größe der Systeme darf keine Rolle spielen
- Leistungsfähigkeit** Schnelle Reaktion auf Änderungen, gute Routen generieren, administrative Vorgaben einsetzbar
- Flexibilität** Besonders Reaktionen auf inter-AS Routing muss schnell möglich sein

## 4.6.2 Routing Verfahren

<b>Distance Vector Routing</b>	Distributed Bellman-Ford Algorithm (Kommunikation nur mit unmittelbaren Nachbarn, Problem in großen Netzen)
<b>Link State Routing</b>	Gewichteter Graph des gesamten Netzes, Shortest Path Algorithm (Probleme in großen Netzen, hohe Netz-/CPU-Last)
<b>Path Vector Routing</b>	Ähnlich Distance Vector Routing, aber nur ausgewählte Hosts (Speaker Nodes) werden einbezogen (BGP)
<b>Broadcast Routing</b>	Pakete sollen an alle Router gesendet werden (z.B. Flooding, Reverse Path Forwarding)
<b>Multicast Routing</b>	Pakete sollen an eine Gruppe von Routern gesendet werden (Multidestination Routing zum Einrichten einer Multicastgruppe)

## 4.6.3 Flooding

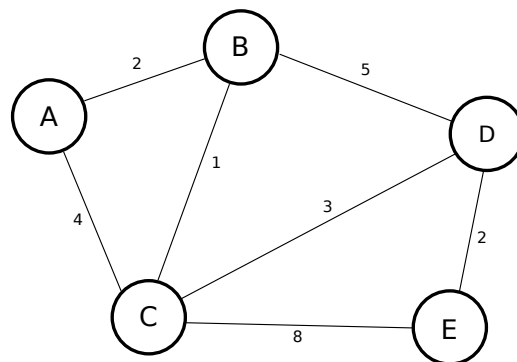
Jeder Router sendet ein ankommendes Paket an alle Leitungen außer an die Quelleitung. Da sehr viele Duplikate entstehen, müssen Maßnahmen zur Begrenzung getroffen. Der Vorteil ist, dass immer der kürzeste Weg benutzt wird.

## 4.6.4 Multidestination Routing

Wird verwendet um Multicastgruppen einzurichten. Es wird ein Broadcast geschickt mit einer Liste in der alle Router stehen, die der neuen Gruppe angehören. Wenn ein Router die Liste erhält und selber drin steht, weiß er, dass er zu der neuen Gruppe gehört und modifiziert die Liste.

## 4.6.5 Reverse Path Forwarding

Bekommt ein Router ein Paket, prüft er, ob es auf der Leitung angekommen ist, auf der er üblicherweise zu der Quelladresse senden würde. Ist dem so, nimmt der Router an, dass das Paket auf dem schnellsten Weg angekommen ist und leitet es an alle Nachbarn weiter.



#### 4.6.6 Bellmann-Ford Algorithmus

Netzwerk wird als ungerichteter gewichteter Graph betrachtet. Jeder Router erzeugt seine eigene Routing Tabelle nach BF.

Algorithmus: Bestimme Routingtabelle von jedem Router einzeln für jeden einzelnen Schritt. Im ersten Schritt sind nur die direkten Nachbarn erreichbar. Im n-ten Schritt sind alle Router erreichbar die mit n Hops erreicht werden können. Bsp: Router A:

Schritt	A	B	C	D	E
1	0	2B	4C	$\infty$	$\infty$
2	0	2B	3B	7B u. 7C	12C
3	0	2B	3B	6B	9B u. 9C
4	0	2B	3B	6B	8B

#### 4.6.7 Dijkstra Algorithmus

Es wird für einen Knoten die Routingtabelle erstellt, indem nacheinander alle Knoten im Netzwerk nach der Entfernung zu ihren Nachbarn gefragt werden. Angefangen wird bei sich selbst. D.h. die Entfernungen zu den eigenen Nachbarn kommen in die Tabelle. Dann werden die benachbarten Knoten, beginnend bei dem mit der geringsten Entfernung besucht und nach ihren Nachbarn gefragt.

Schritt	A	B	C	D	E	bereits besucht
1	$\infty$	$\infty$	8C	2D	0	{E}
2	$\infty$	7D	5D	2D	0	{E,D}
3	9D	6D	5D	2D	0	{E,D,C}
4	8D	6D	5D	2D	0	{E,D,C,B}
5	8D	6D	5D	2D	0	{E,D,C,B,A}

#### 4.6.8 Bewertung von Netzen

	A	B	C	D	E
A	$d_{00}$	$d_{01}$	$d_{02}$	$d_{03}$	$d_{04}$
B	$d_{10}$	$d_{11}$	$d_{12}$	$d_{13}$	$d_{14}$
C	$d_{20}$	$d_{21}$	$d_{22}$	$d_{23}$	$d_{24}$
D	$d_{30}$	$d_{31}$	$d_{32}$	$d_{33}$	$d_{34}$
E	$d_{40}$	$d_{41}$	$d_{42}$	$d_{43}$	$d_{44}$

**durchschnittlicher Abstand**  
(average distance):

$$d = \frac{\sum_{i=0}^N \sum_{j=0}^N d_{ij}}{N \cdot (N - 1)}$$

wobei  $d_{ij}$  die kürzeste Verbindung zwischen 2 Knoten (Routern) ist.

**Durchschnittlicher Grad der Vermaschung (connectivity):**

$$cf = \frac{\sum_{i=0}^N M_i}{N \cdot (N - 1)}$$

wobei  $M_i$  die Anzahl der Nachbarn von einem Knoten (Router) ist.

## 4.7 Internet Group Management Protocol (IGMP)

Bei Multicast Paketen sollen Daten an eine Gruppe von Zielsystemen (Prozesse) gesendet werden. IP-Multicast ermöglicht geringe Latenzen bei großen Gruppen. Ein IGMP Paket wird in einem IP Rahmen transportiert. Das Type Feld im IP Header hat dann den Wert 2. Die Adresse 224.0.0.1 richtet sich an alle multicastfähigen Knoten im LAN. Bei der Verwaltung von Gruppen wird zwischen Hosts und Routern unterschieden. Auf einem Host können mehrere Prozesse zur selben Gruppe gehören. Sobald ein Prozess auf einem Host einer neuen Multicastgruppe angehört sender der Host ein IGMP-Report. Wenn kein Prozess auf dem Host mehr der Gruppen angehört wird diese Gruppe nicht mehr dem Router gemeldet. Die Router fragen die Gruppenzugehörigkeit periodisch ab. Dazu sendet er ein IGMP-Query an den Host, der dann mit einem IGMP-Report für jede Gruppe antwortet. Somit lernt der Router hinter welchen Schnittstellen sich Teilnehmer einer Gruppe befinden und ändert sein Routingverhalten so, dass Multicast Pakete nur auf die Schnittstellen weitergeleitet werden hinter denen sich Teilnehmer der entsprechenden Gruppe befinden. Für Ethernet werden die 23 niederwertigen Bit der IP Adresse in die letzten 3 Byte der Multicast Ethernet Adresse 01 : 00 : 5E : 00 : 00 : 00 kopiert.

0 - 3	4 - 7	6 - 15	16 - 32
Version	Type	unused	Checksum
Multicast Group Address			

<b>Version</b>	immer 1, in späteren Versionen gibt es das KackFeld nicht mehr
<b>Type</b>	1 für Host Membership Query oder 2 für Host Membership Report
<b>Checksum</b>	übliche 16 bit IP-Prüfsumme
<b>Group Address</b>	0 falls Host Membership Query, sonst Multicast IP Adresse

## 4.8 Open Shortest Path First (OSPF)

OSPF Daten werden im IP Rahmen versendet (Protocol 89) und benutzt den Link State Algorithmus Dijkstra um intra-AS Routen zu steuern. Jeder Router sendet seinen Status (Entfernungen zu benachbarten Routern) an alle Router. Jeder Router berechnet seine eigene Routingtabelle dann nach dem Dijkstra Algorithmus. OSPF kann Load Balancing (paralleles nutzen verschiedener Pfade mit gleichen Kosten) und echte Authentifizierung. OSPF erlaubt eine hierarchische Aufteilung des AS in kleinere AS („Areas“), die durch **Area Border Router** mit dem **Backbone AS** verbunden sind. Vier Routertypen existieren in OSPF:

<b>Internal Router</b>	Router in einer Area, die nicht mit dem Backbone verbunden ist
<b>Area Border Router</b>	Verbinden Area und Backbone
<b>Backbone Router</b>	Interne Router im Backbone
<b>Boundary Router</b>	Verbindung zu anderen AS

## 5 Transportschicht (Transport Layer)

Die Transportschicht bietet einen verbindungslosen und einen verbindungsorientierten Dienst, unabhängig von der zu Grunde liegenden Vermittlungsschicht.

Im verbindungsorientierten Dienst bietet die Transportschicht einen gesicherten, bestätigten Datenstrom zwischen den Endpunkten.

Endpunkte werden durch Transportadressen (Ports) beschrieben. Im Internet z.B. sind die Endpunkte Prozesse, die durch Ports identifiziert werden. Der verbindungslose Dienst regelt nur die Adressierung. Die Transportschicht liegt nur beim Nutzer/Endgerät und bietet unabhängig von der eigentlichen Übertragung, es können Verzögerungen durch Speicher, Buffer und Jitter auftreten.

### 5.1 Funktionen

- Adressierung
- Segmentierung
- Verbindungsauf und -abbau
- Fehlererkennung und -behebung
- Flusskontrolle

#### 5.1.1 Verbindungsaufbau durch three-way handshake

Um eine Verbindung aufzubauen, sendet Host 1 ein Connection Request (CR) mit einer initialen Sequenznummer an Host 2, dieser antwortet mit einem SYN-ACK zu der Sequenznummer von Host 1 und sendet gleichzeitig seine initiale Sequenznummer (SYN-ACK). Host 1 bestätigt die Sequenznummer von Host 2 mit einem ACK. Danach können Daten ausgetauscht werden. Probleme können jedoch doppelte, verspätete oder verlorene Pakete darstellen. Timeouts und das Durchnummerieren (Sequenzieren) zusammengehöriger Pakete sind Lösungsansätze. Die ACK-Pakete quittieren dann immer empfangene Pakete unter Angabe der zugehörigen Sequenznummer.

#### 5.1.2 Verbindungsabbau

Man unterscheidet zwei verschiedene Typen:

**Asymmetrischer Abbau:** Ein Host kann gesamte (duplex) Verbindung trennen

**Symmetrischer Abbau:** Jeder Host beendet seine (Hin-)Verbindung separat

Eine Verbindung abzubauen führt zu dem Problem, dass ein Disconnect Request (DR) verloren gehen kann und der Sender damit nicht weiß, ob die Verbindung korrekt beendet wurde. Auch hier ist ein three-way handshake (evtl. mit timeout) eine mögliche Lösung.

### 5.1.3 Pseudoheader zur Prüfsummenberechnung

0 - 7	8 - 15	16 - 31
Source IP Address		
Destination IP Address		
0	Protocol	TCP/UDP Length
TCP/UDP Header		
(Padded) TCP/UDP Data		

Um gewisse IP Felder mit in die Prüfsummenberechnung zu übernehmen und damit z.B. die IP Adressen mit abzusichern, wird die Prüfsumme über den gesamten Pseudoheader gebildet. Die **TCP/UDP Length** bezieht sich ausschließlich auf den Header und die Daten des TCP/UDP Rahmens, nicht auf den Pseudoheader. Ist die Prüfsumme laut Berechnung 0x0000, so wird sie auf 0xFFFF gesetzt. Wird ein Rahmen mit Checksumme 0x0000 empfangen, wurde die Prüfsumme nicht berechnet.

## 5.2 Transmission Control Protocol (TCP)

TCP (Protocol 6 bei IP) stellt einen verbindungsorientierten und gesicherten Transport zur Verfügung. Dabei wird jedes Byte sequenziert, d.h. ein acknowledgement über jede korrekt empfangene Dateneinheit generiert (s.u.).

0 - 3	4 - 9	10 - 15	16 - 19	20 - 25	26 - 31
Source Port Number			Destination Port Number		
Sequence Number					
Acknowledgment Number					
length	reserved	<i>U A P R S F</i> <i>R C S S Y I</i> <i>G K H T N N</i>	Window Size		
TCP Checksum			Urgent Pointer		

**Länge: 20 Byte (ohne Extensions)**

<b>Source Port</b>	Nummer, mit der die sendende Applikation vom Host identifiziert werden kann (ausgehender Port)
<b>Destination Port</b>	Nummer, mit der die empfangende Applikation vom Host identifiziert werden kann (eingehender Port)
<b>Sequence Number</b>	Sequenznummer des ersten Bytes dieses Segments. Bei SYN ist das die ISN
<b>ACK Sequence Number</b>	beinhaltet die Sequenznummer des Bytes, das als nächstes vom Sender erwartet wird
<b>Header Length</b>	Länge des TCP Headers inklusive Optionen in 4 Byte Einheiten
<b>FIN</b>	Sender an Empfänger: es folgen keine weiteren Daten mehr
<b>SYN</b>	Sender an Empfänger: sende initiale Sequenznummer
<b>RST</b>	Verbindung wird zurückgesetzt, jeglicher Kontext wird verworfen
<b>PSH</b>	Empfänger soll Daten sofort an Anwendungsschicht übergeben
<b>ACK</b>	Daten im Feld ACK Sequence Number sind gesetzt
<b>URG</b>	Daten im Urgent Pointer sind gesetzt
<b>ECE</b>	ECN-Echo, Bit 9 im 'reserved' Feld
<b>CWR</b>	Congestion Windows Reduced, Bit 8 im 'reserved' Feld
<b>NS</b>	Nonce Sum, experimentelle Erweiterung, Bit 7 im 'reserved' Feld
<b>Window Size</b>	Menge an Daten, die der Sender des Pakets beginnend mit der ACK Sequence Number noch zwischenspeichern kann (Einheit hängt von Optionen ab).
<b>TCP Checksum</b>	Prüfsumme über erweiterten Header und Daten
<b>Urgent Pointer</b>	Ist das URG Flag gesetzt, zeigt der Pointer auf das erste Byte, das auf die Urgent Daten im Segment folgt
<b>TCP Length</b>	Pseudo Header, Länge von TCP Header und Daten (für Prüfsummenberechnung)

### 5.2.1 Bezeichnungen

<b>Segment</b>	TCP Rahmen mit Daten
<b>MSL</b>	Maximum Segment Lifetime, Zeit, die ein Segment maximal im Netz verbringen darf (120 Sekunden)
<b>SYN, FIN, ACK, RST</b>	TCP-Rahmen, in dem das entsprechende Flag gesetzt ist, auch kombiniert (SYN-ACK, etc.)
<b>MSS</b>	Maximum Segment Size, maximale Datenmenge in einem Segment

### 5.2.2 Socket und Connection

Das **Socket** beschreibt einen eindeutigen Endpunkt einer Verbindung, in dem er die Internet Adresse mit dem Port angibt. Als **Connection** werden alle Informationen bezeichnet, die eine Kommunikationsbeziehung zwischen zwei Prozessen auszeichnen (Sockets der Endpunkte, TCP Optionen, aktuelle Sequenznummern und Fenstergrößen).

### 5.2.3 Verbindungsaufbau, Verbindungsabbau und Datenübertragung

Um eine TCP Verbindung aufzubauen, sendet der Client ein SYN Rahmen mit initialer Sequenznummer  $S_{\text{client}}$ . Der Server sendet ein SYN-ACK, mit dem er  $S_{\text{client}} + 1$  bestätigt und gleichzeitig seine ISN  $S_{\text{server}}$  mitteilt. Danach bestätigt der Client wiederum  $S_{\text{server}+1}$  und eine Verbindung ist aufgebaut. Ein vollständiger Verbindungsabbau muss in festgelegter Reihenfolge und in beiden Richtungen erfolgen. Dabei wird ein FIN mit aktueller Sequenznummer  $S$  gesendet und eventuell die letzte Sequenznummer der Gegenseite mit einem ACK bestätigt. Die Gegenseite muss mit  $S + 1$  bestätigen und selbst ein FIN senden. Dieses FIN muss noch bestätigt werden und dann ist die Verbindung erfolgreich abgebaut. Will eine Seite die Verbindung sofort abbrechen, kann sie ein RST schicken. Dies führt dazu, dass die Gegenseite jeden Kontext der Verbindung verwirft. Ein RST wird nicht bestätigt. TCP benutzt ein Schiebefensterprotokoll mit variablen Segmentgrößen. Wird ein korrekt übertragene Segment empfangen, bestätigt der Empfänger dies mit einem ACK, nicht bestätigte Segmente werden erneut gesendet. Ist die Prüfsumme eines Segments falsch, wird auf eine erneute Übertragung gewartet. Mehrfach übertragene Segmente werden verworfen. Startet das Segment am Fensteranfang (Empfängerpuffer), darf auch ein "Delayed ACK" gesendet werden, was bedeutet, dass der Empfänger mit dem Senden des ACKs warten darf bis er selber Daten senden muss (Piggybacking) oder ein Timer (praktisch 200ms) abläuft. Über die Windows Size steuert der Empfänger die maximale Geschwindigkeit des Senders. Abhängig von den Optionen beim Verbindungsaufbau werden NACKS unterstützt (z.B. bei Selective Repeat ACK)

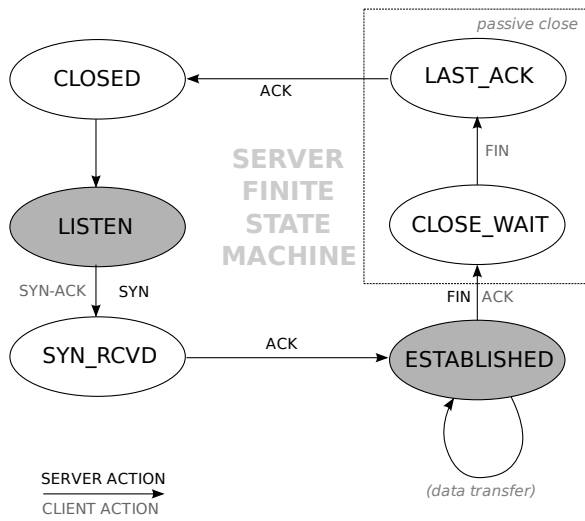


Abbildung 1: Server State Machine

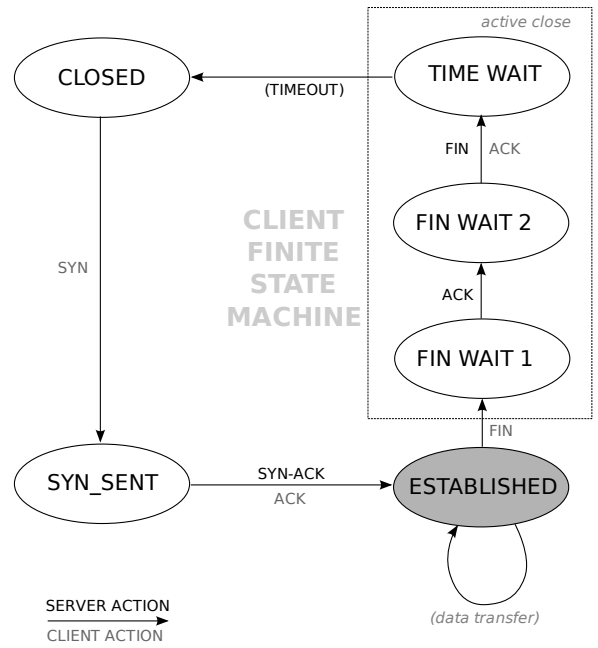
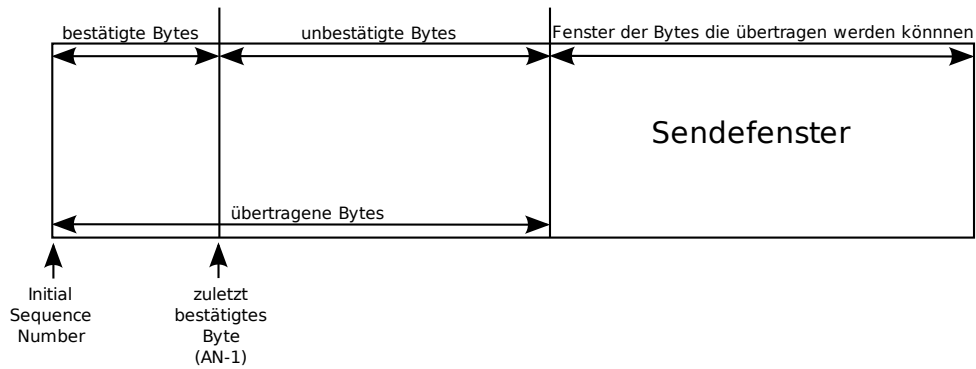


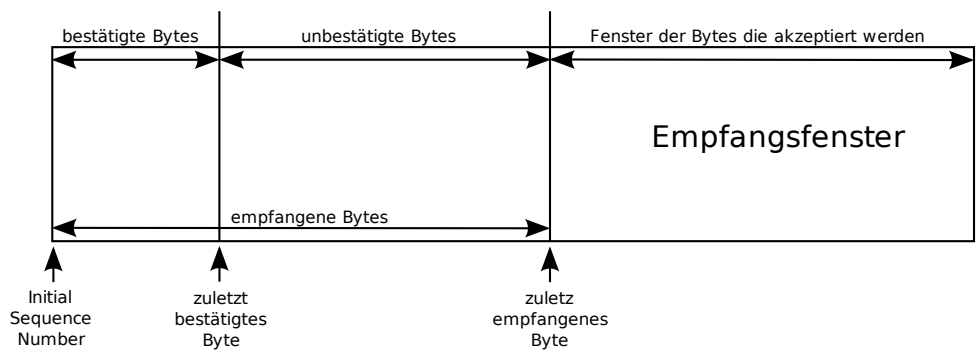
Abbildung 2: Client State Machine

### 5.2.4 Window Management

Flusssteuerung beim Sender:



Flusssteuerung beim Empfänger:



Die Quittungsummer bezieht sich immer auf die Sequenznummer des Senders.

### 5.2.5 Reset (RST)

Ein RST ist ein TCP Segment ohne Daten. Wird ein RST nach Empfang eines ACKs gesendet, ist die Sequenznummer die Quittungsnummer des empfangenen ACKs. Andernfalls ist die Sequenznummer 0 und die Quittungsnummer, wie üblich, die Nummer des ersten freien Bytes im Fenster des RST-Senders (RST-ACK).

#### Generierung eines Reset-Rahmens:

1. beim Empfang eines ungültiges Segments. Ein ungültiges Segment ist z.B. ein SYN für einen Port, der nicht zu einer Verbindung gehört, die im Zustand SYN\_SENT oder LISTEN ist.
2. Ist eine Verbindung in den Zuständen LISTEN, SYN\_SENT, SYN\_RCVD und es wird ein ACK mit einer ungültigen Quittungsnummer empfangen. Kommt ein ACK mit falscher Sequenznummer für eine Verbindung in den anderen Zuständen an, wird kein RST sondern ein ACK mit den aktuellen Quittungsnummern gesendet.

#### Verarbeitung von RST-Rahmen

1. Im jedem Zustand außer SYN\_SENT muss die Sequenznummer des RST im Sendefenster liegen, sonst wird der RST verworfen
2. Im Zustand SYN\_SENT wird der RST-ACK verworfen, wenn die Quittungsnummer die Sequenznummer im SYN nicht bestätigt
3. Im Zustand LISTEN wird RST verworfen
4. Im Zustand SYN\_RCVD kehrt der Servr nach LISTEN zurück, wenn er sich vorher im LISTEN befunden hat
5. In allen anderen Fällen geht er in den Zustand CLOSED zurück

#### Zeitverhalten

1. Wird der initiale SYN nicht beantwortet, so unternimmt der Sender eine (konfigurierbare) Anzahl von Versuchen, bevor er der Anwendungsschicht eine Fehlermeldung sendet.
2. Wird der SYN-ACK nicht beantwortet, so unternimmt der Sender eine (konfigurierbare) Anzahl von Versuchen, bevor er den Kontext wieder freigibt
3. Die Zeit zwischen erneutem Übertragen und nicht bestätigter Segmente ist implementationsabhängig
4. Gibt ein Segment eine Window Size von 0 Byte an, fragt die Gegenstelle periodisch, ob wieder Daten gesendet werden können (Persist Timer)

#### Keepalive

Ist eine Verbindung im Zustand ESTABLISHED, es fließen jedoch keine Daten, kann dies zu Problemen führen. Ist einer der Endpunkte ausgeschaltet oder neu gestartet, bemerkt die Gegenseite nichts davon. Außerdem halten die Endpunkte den Kontext für unbestimmte Zeit aufrecht. Diese Probleme werden mit Keepalive Timern umgangen. Nach 2 Stunden Inaktivität, wird ein ACK mit den aktuellen Sequenznummern gesendet (und entsprechend beantwortet). Erfolgt kein ACK, werden weitere Tests durchgeführt.

#### Nagle Algorithmus

Übergibt die Anwendungsschicht langsam Daten an die Transportschicht, führt das zu vielen Paketen mit wenig Nutzdaten. Dies kann durch den Nagle Algorithmus verhindert werden. Dieser besagt, dass erst Daten gesammelt werden, solange nicht alle Segmente bestätigt worden sind. Ist die MSS erreicht oder sind alle Segmente bestätigt, soll sofort gesendet werden. Delayed ACKs können dabei zu Problemen führen.

## 5.2.6 TCP Optionen

Ein Optionsfeld ist immer von der Länge 4 Byte (zur Not NULL padding) und haben (bis auf 2 Ausnahmen) die Felder Typ, Länge, Wert.

### Ausnahmen

- 0 End of Options, 1 Byte, kein Längenfeld
- 1 No Operation, 1 Byte, kein Längenfeld

### Weitere Optionen

- 2 Maximum Segment Size (MSS), 1 Byte Optionstyp, Länge 4, 2 Byte für MSS
- 3 Window Scale Factor, Länge 3, 1 Byte Shift, 1 Byte Padding
- 4 Selective ACK Permitted, Länge 2 (NUR in SYN) (kein Wert, nur Typ und Länge)
- 5 Selective ACK, Länge variabel
- 8 Timestamp, Länge 10, 8 Byte für 2 Zähler

### MSS Optionen

Ziel ist, die MSS so einzustellen, dass keine Fragmentierung auf Vermittlungsschicht nötig ist. Die MSS wird im SYN und SYN-ACK gesetzt. MSS ist üblicherweise die MTU des Interfaces abzüglich der minimalen Headerlänge (40 Bytes bei IPv4) benutzt wird, sonst Standardwert 536 Bytes. Beide Seiten verwenden das Minimum der eigenen MSS und der MSS der Gegenstelle zur Segmentierung. Bei bestehender Verbindung kann Segmentierung durch ICMP (Destination Unreachable, Fragmentation needed) verkleinert werden. (Path MTU Discovery).

### Timestamp Optionen

Die Entscheidung, ob ein Segment erneut übertragen werden muss, hängt maßgeblich von der Zeit ab, wie lange die Gegenstelle durchschnittlich für eine Antwort benötigt. Außerdem muss die Fenstergröße berücksichtigt werden, wodurch die Antwortzeit nur geschätzt werden kann. Die Timestamp Option wird benutzt, wenn im SYN und SYN-ACK die Option gesetzt wurde (beide Hosts verstehen die Option). Das Timestamp Value (TSval, 4 Byte) und der Timestamp Echo Reply (TSecr, 4 Byte) wird bei TCP mit folgendem Algorithmus berechnet.

### Window Scale Option

Bei Verbindungen mit hoher Datenrate und hoher Latenz (lange Laufzeit von Paketen) ist das 64KByte Fenster nicht ausreichend, um hohen Durchsatz zu erreichen. Der 1 Byte Shift Count der Window Scale Option gibt an, um wie viele Bit der Wert im Window Size Feld des TCP Rahmens nach links verschoben werden muss, um die tatsächliche Größe des Fensters in Byte zu erhalten. Dabei müssen Sender und Empfänger beide im SYN und SYN-ACK ihren Shift Count angeben. Der maximale Shift Count ist 14, d.h.  $64k \cdot 2^{14}$  ist die maximale Fenstergröße.

### Selective ACK (SACK)

Selective ACK darf nur verwendet werden, wenn beide Hosts SACK Permitted im SYN SYN-ACK bestätigen. Es können immer nur zusammenhängende Blöcke von Segmenten quittiert werden. Das erste angegebene Segment ist korrekt empfangen worden (das davor nicht) und das letzte angegebene Segment ist noch nicht empfangen worden, also ein Halboffenes Interval, [...]. Falls keine weiteren Optionen verwendet werden (speziell Timestamp), ist die maximale Anzahl von Blöcken 4. Durch das SACK verfahren werden auch fehlende Segmentblöcke implizit angefordert.

### Duplicate Selective ACK (DSACK)

Diese Option wurde eingeführt, um den Sender über unnötig wiederholte Segmente zu informieren. (Folie 7 S. 12)

## 5.2.7 TCP Flusskontrolle

Um Überlastungen im Netz (Engpässe bei Sender oder Empfänger) durch große Fenster zu verhindern, werden 4 Algorithmen verwendet:

1. TCP Slow Start
2. Congestion Avoidance
3. Fast Retransmits
4. Fast Recovery

### TCP Slow Start und Congestion Avoidance

Eine TCP Verbindung kann in 2 Abschnitte geteilt werden. Der erste Abschnitt beschreibt die Slow Start Phase. Der Sender begrenzt sein Sendefenster selbstständig durch das sogenannte Congestion Window (CW). Die Größe wird oft auf die Maximum Segment Size (MSS), höchstens aber auf das Vierfache der MSS, initialisiert:

$$CW := MSS \quad (\text{oder } 4 \cdot MSS)$$

Sobald soviele Oktette quittiert wurden, wie die MSS angibt, wird das Congestion Window um ein weiteres MSS erhöht

$$CW = CW + MSS$$

Dadurch ergibt sich ein exponentieller Anstieg für CW. Die Slow Start Phase dauert so lange an, bis die Grenze Slow Start Threshold (ssthresh) erreicht wird. Sie wird oft initialisiert mit

$$ssthresh = W$$

wobei  $W$  für die Advertised Window Size steht.

Die zweite Phase heißt Congestion Avoidance. In dieser Phase wird das CW lediglich dann um  $MSS$  erhöht, wenn alle gesendeten Oktette bestätigt werden. D.h., das Congestion Window erhöht sich bei erfolgreicher Übertragung pro Round Trip Time (RTT) um Eins, höchstens aber bis Window Size. Sobald Pakete verloren gehen (Timeout, doppelte ACKs), wird das  $ssthresh$  auf die Hälfte des vorher genutzten CW gesetzt oder auf das Doppelte von  $MSS$ , je nach dem, welches größer ist. Außerdem wird das CW wieder auf  $MSS$  gesetzt. Somit befindet sich der Sender wieder in der Slow-Start-Phase.

$$ssthresh = \max\left\{\frac{CW}{2}, 2 \cdot MSS\right\}$$

Doppelte ACKs treten immer dann auf, wenn unter mehreren Paketen eines verloren geht. Der Empfänger macht somit den Sender auf einen verlorenes (oder verspätetes) Paket aufmerksam. Der Empfänger bekommt zwar neue Pakete, aber wartet eigentlich noch auf das (vermutlich verloren gegangene) Paket, welches mit der (doppelten) ACK Nummer beginnt.

### TCP Fast Retransmit and Recovery

Wenn der TCP Sender im Status Fast Retransmit ist sendet er nach dem 3. duplicated ACK (also nach insgesamt 4 ACKs mit gleicher Quittungsnummer) sofort das fehlende Paket ohne auf das Ablaufen des RTO zu warten. Ist der der Sender im Fast Recovery Status wird nach einem Fast Retransmit das CW nicht auf  $MSS$  gesetzt, sondern auf  $ssthresh + n \cdot MSS$  wobei  $n$  die Anzahl der insgesamt angekommenen Duplicated ACKs ist (also die Anzahl der Segmente, die der Empfänger noch im Puffer hat).

### Retransmission Timeout (RTO) und Jacobsen Algorithmus

Die RTO ist die Zeit, die gewartet wird bis ein Segment erneut gesendet wird, falls kein ACK empfangen wurde. Ein Problem bei einer TCP Verbindung ist, dass die RTT nicht exakt bekannt ist und sich die RTO an der RTT orientiert. Die RTO wird nach Messungen der RTT ( $M$ ) geschätzt. Es muss auf jeden Fall gelten:  $RTO > RTT$ . Die RTO darf nicht zu klein sein, da sonst zu viele Segmente erneut übertragen werden und sie darf nicht zu groß sein, da sonst unnötig viel Zeit verstreicht bevor ein Segment erneut übertragen wird. Zur Bestimmung der RTO wird der Jacobsen Algorithmus verwendet.

$$RTT_{i+1} = \alpha \cdot RTT_i + (1 - \alpha) \cdot M_i \quad \text{mit z.B.} \quad \alpha = \frac{7}{8}$$

Neuer Wert wird stärker gewichtet als alter Wert. Die RTO ergibt sich dann wie folgt:

$$RTO = \beta \cdot RTT$$

Ein festes  $\beta$  verursacht Probleme. Darum wird  $\beta$  oft an die Standardabweichung von RTT gekoppelt:

$$RTO = RTT + 4 \cdot D \quad \text{mit} \quad D_{i+1} = \alpha D_i + (1 - \alpha) |RTT_i - M_i|$$

## 5.3 User Datagram Protocol (UDP)

UDP ist ein verbindungsloser Dienst mit dem Wert 17 im Protocol Feld vom IP Header. UDP arbeitet wie TCP mit Portnummern, ist jedoch ungesichert. UDP unterstützt Unicast und Multicast und ist immer eine Folge von Bytes. UDP bietet keine Flusskontrolle, Fehlerkorrektur und Retransmission und ermöglicht höheren Schichten damit ungefilterten Zugriff auf das IP Protokoll, damit z.B. eigene Verfahren implementiert werden können (u.A. Streaming, RTP, DSN, RPC).

### 5.3.1 UDP Rahmen

0 - 15	16 - 31
Source Port	Destination Port
UDP Length	UDP Checksum
UDP Data	

**Länge: 8 Byte**

<b>Source Port</b>	Nummer, mit der die sendende Applikation vom Host identifiziert werden kann (ausgehender Port)
<b>Destination Port</b>	Nummer, mit der die empfangende Applikation vom Host identifiziert werden kann (eingehender Port)
<b>UDP Length</b>	16 Bit Länge des gesamten Datagrams (mit Header) in Bytes
<b>UDP Checksum</b>	Prüfsumme über gesamtes Datagramm zuzüglich Pseudoheader
<b>Urgent Pointer</b>	Ist das URG Flag gesetzt, zeigt der Pointer auf das erste Byte, das auf die Urgent Daten im Segment folgt
<b>TCP Length</b>	Pseudo Header, Länge von TCP Header und Daten (für Prüfsummenberechnung)

## 6 Anwendungsschicht (Application Layer)

### 6.1 Domain Name System

Das Domain Name System bildet ein verteiltes Verzeichnis zur Umwandlung von Namen in Adressen, wobei die Namen einer festgelegten Syntax entsprechen müssen: max. 63 Zeichen pro Bezeichner (Label), die mit Punkten getrennt werden und bei der Groß- und Kleinschreibung keine Rolle spielen. Namen, die mit einem Punkt enden, werden als vollständig angenommen, andernfalls ist eine Erweiterung nötig, um zum **Fully Qualified Domain Name (FQDN)** zu kommen. Das Verzeichnis baut sich in umgekehrter Reihenfolge auf. Auf das Wurzelverzeichnis (Root) folgen die **Top Level Domains (TLD)**, die sich in die Kategorien Country Coded für Länder (.de, .fr, .us, ...), Generic für Organisationen o.Ä. (.com, .net, .org, ...) und Infrastructure (.root, .arpa) einteilen lassen. Top Level Domains zeichnen außerdem separat administrierbare Unterverzeichnisse aus, sogenannte Zonen, für die ein Administrator einen DNS Server bereitstellen und verwalten muss. Weitere Unterbäume können wiederum an andere Administratoren verteilt werden. Der Primary Server einer Zone ist der Server, auf dem die Konfigurationsdaten administriert werden, die Secondary Server besitzen jeweils Kopien, die sie von dem Primary Server herunterladen (der sogenannte Zone Transfer). Secondary Server, die den Zone Transfer mit dem Primary Server durchführen, nennt man autorativ, da diese eine aktuelle Kopie der Zone besitzen.

#### 6.1.1 DNS Rahmen

Identification	Flags
Number of Questions	Number of RRs
Number of Authority RRs	Number of Additional RRs
Questions	
Answer RRs	
Authority RRs	
Additional RRs	

**Länge: 28 Byte**

<b>Identification</b>	16 Bit Code, mit dem die Antwort einem Request zugeordnet werden kann
<b>Flags</b>	16 Bit Flags (s.u.)
<b>Number of Questions</b>	0: Antwort, 1: Anfrage
<b>Number of RRs</b>	Anzahl Resource Records (Antworten auf eine Anfrage)
<b>Number of Authority RRs</b>	Anzahl Authority Records (DNS Server, die die gesuchte Information sicher haben)
<b>Number of Additional RRs</b>	Anzahl Additional Records (Zusatzinfos, z.B. die Adressen der DNS Server aus den Authority Records)

#### Die Flags

QR	Opcode	AA	TC	RD	RA	Empty (0)	RCode
----	--------	----	----	----	----	-----------	-------

- QR** 0: Anfrage, 1: Antwort
- Opcode** 0: Default, 1: Inverse Abfrage, 2: Server Status
- AA** Antwort ist „authoritative“
- TC** (Truncated) Paket enthält nur 512 Bytes der Antwort
- RD** 1: Server soll Anfrage rekursiv bearbeiten
- RA** 1: Server bietet Rekursion an
- RCode** Fehlerstatus, 0: Kein Fehler, 3: Name nicht gefunden

### Question Datensatz

Query Name	
Query Type	Query Class

- Query Name** Name, der abgefragt wird
- Query Type** Typ der Anfrage, entweder Standardtyp oder Any (jeder Standardtyp) oder AFXR (Zonentransfer)
- Query Class** 1 für Internet (IN)

### Standardtypen

Typ	Name	Wert
A	1	IPv4 Adresse
NS	2	Name Server Name
CNAME	5	Canonical Name (Alias)
SOA	6	Start Of Authority, Administrative Daten einer Domain
PTR	12	Pointer Record (Verweis)
HINFO	13	Host Info (Informationen über den Rechner)
MX	15	Mail Exchange (Zuständiger E-Mail Server)
AAAA	28	IPv6 Adresse
SRV	32	Server Record (Host:Port des Servers)
NAPTR	35	Naming Authority Pointer

### Resource Records

Domain Name	
Type	Class
Time To Live	
Data Length	Data
Data	

<b>Domain Name</b>	Name, zu dem die Daten gehören
<b>Type</b>	Standardtyp für Daten
<b>Class</b>	1 für Internet (IN)
<b>Time To Live</b>	(TTL), Zeit in Sekunden, die der Eintrag von einem DNS Server oder Client aus dem Cache verwendet werden darf
<b>Data Length</b>	Länge der folgenden Daten in Byte
<b>Data</b>	Daten und Kodierung abhängig vom Typ

### 6.1.2 Transport

Üblicherweise werden DNS Anfragen per UDP gestellt und Zeitüberschreitungen und Wiederholungen werden von dem Client gesteuert. Zonentransfers von dem Primary zu den Secondary Servern finden über TCP statt. Der Zielport ist üblicherweise auf Port 53 festgelegt, der Quellport kann aber beliebig gewählt werden. Sind mehr als 512 Byte zu übertragen, muss der Client via TCP erneut nachfragen (TC Flag ist dann in der Antwort gesetzt)

### 6.1.3 Auflösung von Namen

Ein Host stellt die Anfrage bei den (maximal 3) konfigurierten Nameservern (RD Flag ist üblicherweise 1, d.h. Server soll Anfrage rekursiv bearbeiten). Befinden sich die Informationen im Cache eines DNS Servers, sendet er die zugehörigen Daten. Kann die Anfrage nicht lokal beantwortet werden, so wird einer der konfigurierten ROOT Server angefragt. Die Antwort besteht dann aus einer Liste der Authoritative DNS Server der zugehörigen Zone, die in den Authority Resource Records (Authority RRs) Feldern abgelegt werden. Eine Anfrage bei einem dieser Server führt dann entweder direkt zum Ergebnis, oder ergibt eine weitere Liste von DNS Servern einer untergeordneten Zone, welche dann wiederum abgefragt werden, bis letztendlich die Anfrage beantwortet werden kann.

### 6.1.4 Reverse Lookup

IP Adressen bilden in der Dotted Notation den Namen im DNS Verzeichnis, wobei die Top Level Domain (TLD) „arpa“ und die Second Level Domain „in-addr“ benutzt wird. Eine IP Adresse a.b.c.d wird in der Zone als „d.c.b.a.in-addr.arpa., administriert. Der Typ entspricht PTR (Pointer Record, Verweis, Alias für eine IP) und der Wert stellt den Fully Qualified Domain Name (FQDN) des Hosts mit der entsprechenden Adresse dar. Die Auflösung des Namens d.c.b.a.in-addr.arpa erfolgt genau wie für alle FQDNs, erlaubt aber zusätzlich die Umwandlung von IP Adressen in DNS Namen (Reverse Lookup). Auch CIDR (Netzwerk mit Maskierung) funktioniert, jedoch muss die Netzmaske auf einer Bytegrenze enden. Hier wird die Kontrolle durch den Parent DNS Server an untergeordnete Server weitergeleitet und die Segmente (Label) geben die Ebenen der administrativen Kontrolle an.

### 6.1.5 Load Balancing

Da Server nur eine begrenzte Anzahl an paralleler Verbindungen bearbeiten können, jedoch ein Vielzahl an Diensten lange Verbindungen zu den DNS Servern erfordern, können Administratoren auch den Namen und nicht die IP konfigurieren. Das bedeutet, dass eine Anfrage beim Verbindungsaufbau in ein Adresse umgewandelt wird, wobei aber die Time To Live des Eintrages beachtet wird. Dadurch können Server nachträglich hinzugeschaltet und die Anfragen auf diese verteilt werden, da nach Ablauf des

TTL alle Chaches, die eine Kopie dieses Eintrags speichern, automatisch ablaufen und somit erneut abgefragt werden müssen.

## 6.2 Hypertext Transfer Protocol (HTTP)

Der aktuelle Standard ist HTTP 1.1. Im Gegensatz zur Urversion 0.9 werden auch andere Datentypen und Formate unterstützt als Text mit HTML Markup. Es wurde außerdem die **Authentifizierung** der Kommunikationspartner ermöglicht. **Virtual Hosting** ermöglicht mehrere unabhängige Dienste auf einem Server hinter **einer** oder mehreren IPs. Das heißt hinter einer IP können sich verschiedene Domains verbergen (Verwendung für virtual Server). Das Grundprinzip ist das eine HTTP Request gesendet wird und als Antwort ein HTTP Response empfangen wird.

### 6.2.1 Aufbau einer HTTP Nachricht

Anfragezeile(für Request) / Statuszeile (für Response)
Header (Name-Werte Paare)
⋮
“Leerzeile”
optionaler Datenblock

Zeilenumbrüche bestehen aus CR-LF (ASCII: 13,10). Namen und Werte der Header sind durch “: “ (Doppelpunkt und Leerzeichen) getrennt. Bei den Namen wird nicht zwischen Groß- und Kleinschreibung unterschieden. Wenn die Syntax eines Wertes mehrere durch Komma getrennte Werte ist, sind Header mit gleichem Namen möglich. Nur bei Headern mit gleichem Namen ist die Reihenfolge wichtig. Um die **Lesbarkeit** von HTTP Nachrichten zu verbessern werden lange Zeilen an LWSP (Leerzeichen oder Tabulatoren) umgebrochen. Das LSWP wird dann durch ein CRLF + LSWP ersetzt. Somit werden Zeilen die mit einem LWSP beginnen als Fortsetzung einer Zeile interpretiert.

Eine Anfragezeile hat die Form

**Methode ' ' URI ' ' HTTP-Version**

Name der Methode	Zweck
OPTIONS	Lesen von Transportoptionen, z.B <i>Allow</i>
GET	Lesen von Daten vom Server
HEAD	Wie GET, doch Server sendet keinen Datenblock
POST	Senden von Daten an den Server
PUT	Speichern von Daten unter der URI
DELETE	Löschen von Daten unter der URI
TRACE	Server sendet den Request als Response
CONNECT	Nur für SSL/TSL Proxies

Ein Server muss nicht alle diese Methoden implementieren.

Eine Statuszeile hat die Form

**HTTP-Version Status-Code Reason-Phrase**

Die Statuszeile enthält den dreistelligen Statuscode, welcher in 5 Hauptgruppen unterteilt wird.

Code	Bedeutung
100 - 199	Inforamtion
200 - 299	Erfolg
300 - 399	Wiederhole mit anderen Daten (Umleitung)
400 - 499	Client-Fehler
500 - 599	Server-Fehler

## 6.2.2 Kodierung von HTTP Nachrichten

Zur Kodierung der ASCII-Zeichen (binäre Schreibweise ist nicht erlaubt) wird die Hexadezimalschreibweise verwendet, d.h. 1 Byte (also ein ASCII-Zeichen) wird zu 2 Hexadezimalzeichen. Für große Datenmengen wird die Base64 Schreibweise verwendet, da sie um 33% effizienter ist. Bei der Base64 Schreibweise werden pro Zeichen 6 Bit kodiert. Gültige Zeichen sind A-Z, a-z, 0-9, '+' und '/' (in dieser Reihenfolge). Zum Beispiel wird  $27_{10}$  kodiert zu  $b_{64}$ . Es werden immer 3 Byte zu 4 Base64 Zeichen kodiert. Falls die Anzahl der Bytes nicht durch 4 teilbar ist wird der Bytestrom mit Nullen aufgefüllt ('='  $\cong$  000000).

## 6.2.3 Uniform Resource Identifier (URI)

Die HTTP URL hat die Form:

**“http://” host [ “:” port ] [ path [ “?” query ] ]**

Dabei sind Teile in eckigen Klammern “[ ]” optional.

**Host** Hostname des Dienstes  
**Port** Portnummer (80, falls weggelassen)  
**path** Pfad zur Resource, jeder Pfad startet mit “/”  
**query** Serverspezifische Parameter, üblicherweise Name “=” Wert Paare, getrennt durch “&” (sog. Formkeys)

IP Adresse statt Hostname ist in URIs zu vermeiden. Falls eine IPv6 Adresse angegeben werden muß, wird sie in eckige Klammern geschrieben. Da URIs auch über andere Transportwege als Computernetze weitergegeben werden, dürfen sie nur druckbare Zeichen enthalten. Im Pfad einer URI sind mindestens die folgenden Zeichen US-ASCII Zeichen zu ersetzen:

Codes 0-31 und 127 Control Character  
Code 32 Space  
<> # % " ' Begrenzer  
{ } | \ ^ [ ] ‘ Diese Zeichen können von bestimmten Transportmechanismen verändert werden

Einige Zeichen haben an bestimmten Stellen der URI eine spezielle Bedeutung und müssen dann ebenfalls ersetzt werden (z.B. im Query Anteil)

- ; / ? : @ & = + \$ ,

Spezielle Zeichen und nicht druckbare Zeichen werden durch ihre Hexadezimaldarstellung, eingeleitet durch ein %-Zeichen, angegeben.

Beispiel: *Host a.b.c, Port 80, Path /a.b/c* und Formkeys  $M=a+b$ ,  $N="a=b"$  wird geschrieben als:

**`http://a.b.c/a_b/c?M=a%2Bb&N=%22a%3Db%22`**

Die URL wird vom Client in einen HTTP 1.1 Request umgesetzt. Dazu wird die Serverinformation, d.h. Hostname und Port in den Host Header kopiert. Die Anfragezeile enthält im Normalfall als URI nur Path, Query und HTTP-Version. Ausnahme ist der Proxy Request.

**Beispiel:**

Ein GET Request auf *http://localhost:8080/test* führt zu:

*GET /test HTTP/1.1  
Host: localhost:8080*

Der Header mit Namen Host ist bei HTTP 1.1 verbindlich

### 6.2.4 HTTP Header

In Request und Response sind unterschiedliche Header üblich. Prinzipiell können beiden Nachrichten beliebige Header hinzugefügt werden (sog. extension-header).

Für Requests sind die folgenden Headergruppen standardisiert:

- General Header
- Request Header
- Entity Header

Für Responses entsprechend:

- General Header
- Response Header
- Entity Header

### einige standardisierten General Header:

<b>Cache-Control</b>	Legt fest, was beim Caching der Daten zu beachten ist. (z.B. no-cache, max-age)
<b>Connection</b>	Zeigt an, ob die TCP Verbindung für weitere Anfragen verwendet werden kann (z.B. keep, close)
<b>Date</b>	Zeitpunkt, zu dem die Nachricht erzeugt worden ist.
<b>Pragma</b>	Implementationsspezifische Parameter (z.B. no-cache)
<b>Trailer</b>	Bei Chunked-Encoding können die angegebenen Header am Ende der Nachricht im Datenblock auftauchen.
<b>Transfer-Encoding</b>	Legt fest, wie der Datenblock übertragen wird (z.B. chunked).
<b>Via</b>	Wird von Systemen zwischen Quelle und Ziel eingesetzt, um Schleifen zu entdecken.

### einige Request Header

<b>Accept, ...</b>	Liste der Mediatypen, Kodierungen und Sprachen, die der Client akzeptiert z.B.: Accept-Charset...
<b>Authorization</b>	Übergabe von Daten und Anforderung der Authentifizierung Authorization oder Proxy-Authorization
<b>Host</b>	Servername (eventuell mit Port)
<b>If-;Bedingung;</b>	Anforderung von Daten nur unter gegebener Bedingung z.B.: If-Match, If-Modified-Since
<b>Max-Forwards</b>	Maximale Anzahl von Proxies in der Kette
<b>Referer</b>	URI, von der aus die aktuelle URI angewählt worden ist
<b>TE</b>	Liste der möglichen Transfer-Encodings
<b>User-Agent</b>	Identifikation des Clients

### einige Response Header

<b>Age</b>	Alter eines Dokumentes (im Cache)
<b>ETag</b>	Dokumentversion
<b>Location</b>	URI des Dokumentes (benutzt im 201 Created und z.B. im 302 Redirect)
<b>Authenticate</b>	z.B.: Proxy- oder WWW-Authenticate Authentifizierungsdaten des Clients
<b>Retry-After</b>	Im 503 Service Unavailable und bei 3xx Redirect, wann die Daten verfügbar sein werden
<b>Server</b>	Identifikation (Typ, Version) des Servers
<b>Vary</b>	Liste der Request-Header, die die Antwort festlegen, nötig für Proxies, um zu entscheiden, ob die Antwort aus dem Cache benutzt wird

### Entity Header

<b>Allow</b>	Methoden, die der Server erlaubt
<b>Content-Encoding</b>	Kodierung, die für die Daten verwendet worden ist (z.B. gzip)
<b>Content-Language</b>	ausgewählte Sprache
<b>Content-Length</b>	Anzahl Bytes im Datenblock, falls nicht <i>Transfer-Encoding: chunked</i>
<b>Content-Location</b>	URI des Dokuments
<b>Content-MD5</b>	Hash der Daten, Prüfsumme des (dekodierten) Datenblocks
<b>Content-Range</b>	gelieferter Bereich in der Antwort <i>206 Partial Content</i>
<b>Content-Type</b>	Medientyp
<b>Expires</b>	Wann Daten im Cache veraltet sind
<b>Last-Modified</b>	Zeitpunkt der letzten Änderung

### Beispiele für Transferencoding

<b>identity</b>	Die Nachricht wird nicht speziell kodiert
<b>gzip</b>	Transparente Komprimierung mit Lempel-Ziv Algorithmus
<b>compress</b>	Unix compress LZW Format
<b>deflate</b>	zlib Format (RFC1950 + RFC1951)
<b>chunked</b>	Der Datenblock besteht aus Länge-Wert kodierten Segmenten. Jedes Segment beginnt mit einer Zeile mit Länge in Hexadezimaldarstellung und optionalem Kommentar. Es folgt die angegebene Anzahl Bytes. Das letzte Segment hat die Länge 0. Auf das letzte Segment können HTTP Header folgen.

Der **Content-Type** gibt den Medientyp im Datenblock an. Format ist stets *Type "/" Subtype\*(;" Parameter)*. Parameter dienen zur weiteren Festlegung der Interpretation, z.B. *text/plain; charset=utf8*. Der Typ *multipart/form-data* dient der Übertragung von Formkeys mit Zusatzinformationen wie Zeichensatz oder Medientyp bei Dateiübertragung. Viele Clients (z.B. MS Windows, Mobiltelefone) benutzen zur Feststellung des Medientyps nicht den Content-Type Header, sondern eventuell vorhandene Dateiheader.

### 6.2.5 HTTP Proxies

Proxies sind integraler Bestandteil einer HTTP Infrastruktur.

<b>Haupteinsatzzwecke</b>	
<b>Effizienzsteigerung</b>	Zwischenspeichern von statischen Daten reduziert den Netzwerkverkehr
<b>Zugriffskontrolle</b>	Netzwerkbereiche können nur über Proxies erreicht werden, die Authentifizierung vorschreiben
<b>Protokollierung/Abrechnung</b>	Proxies können Datenvolumen und Zugriffszeiten protokollieren
<b>Routing</b>	Zugriff aus privaten Netzen kann über die öffentliche Adresse eines Proxies ermöglicht werden
<b>Sicherheit</b>	Komplexe Webservers werden durch einfache Proxies vom Internet isoliert
<b>Proxytypen</b>	
<b>Vorwärtsproxies</b>	Der Client muß konfiguriert werden, um den Proxy zu benutzen. Die Anfragezeile bei Vorwärtsproxies enthält nicht nur den Pfad, sondern die komplette URL
<b>Rückwärtsproxies</b>	Dem Client gegenüber verhalten sie sich wie Server. Sie blenden Pfade fremder Server in den eigenen Bereich ein.
<b>Transparente Proxies</b>	Die TCP Verbindung von Clients wird abgefangen und auf den Proxy umgeleitet. Dieser baut bei Bedarf eine eigene Verbindung zum Server auf.

## 6.2.6 HTTP Authentication

Soll mittels HTTP auf geschützte Bereiche sowohl auf einem Server als auch hinter einem Proxy zugegriffen werden, erlaubt der Standard, dass Authentifizierungsdaten abgefragt werden. Wird der Zugriff verweigert, sendet ein Server eine Response mit *Statuscode 401*, ein Proxy eine Response mit *Statuscode 407*. In der Response findet sich der *WWW-Authenticate* oder *Proxy-Authenticate* Header, der festlegt, wie der Client auf den geschützten Bereich zugreifen kann. Der Client muß dann die Authentifizierungsdaten im *Authorization* bzw. *Proxy-Authorization* Header liefern.

### basic authorization

Basic Authentication funktioniert dadurch, dass beim Zugriff auf den geschützten Bereich Benutzername und Kennwort im Klartext übertragen werden. Findet ein Zugriff ohne gültige Authentifizierung statt, folgt vom Server eine Antwort mit Status 401 und *WWW-Authenticate* Header mit Parametern *basic* und dem Namen des Bereiches. Der Client wiederholt den Request und sendet den Header *Authorization: Credentials*, wobei *Credentials* Benutzername ":" Kennwort in Base64 Kodierung ist.

### digest access authorization

Der entscheidende Nachteil der Basic-Authentication ist, dass jeder, der die Datenübertragung abhört, Benutzernamen und Kennwort erfährt. Dies wird bei der Digest-Authentication vermieden. Der *WWW-Authenticate* oder *Proxy-Authenticate* Header hat die Form: *Digest Challenge*, wobei Challenge eine Folge von Name "=" Wert Paaren ist, die durch "," getrennt werden. Die Antwort hat die Form *Digest Response*, wobei Response dasselbe Format wie Challenge hat. In der Antwort sind einige Parameter die einfach nur zurückgesendet werden und in der eigentlichen Response (Response=Value) ist ein HASH-Wert in den verschiedenste Parameter mit einfließen wie z.B. ein zufälliger Text des Clients (cnonce=Value).

<b>Challenge Parameter</b>	
<b>realm</b>	Name des Sicherheitsbereiches
<b>domain</b>	Liste von URI Präfixen, für die die Credentials gelten
<b>nonce</b>	Eindeutige Challenge
<b>opaque</b>	Wert, der vom Client zum Server zurückgeschickt wird
<b>stale</b>	Zeigt an, daß der vorherige Nonce abgelaufen ist
<b>algorithm</b>	Zu verwendender Hash Algorithmus, z.B. MD5, SHA1
<b>qop</b>	angebotene Sicherheitsstufen, z.B. Authentizität (auth), Integrität (auth-int)
<b>auth-param</b>	z.Zt nicht benutzt
<b>Response Parameter</b>	
<b>username</b>	Name, unter dem der Client sich anmeldet
<b>Challenge Param.</b>	realm, nonce, algorithm, opaque (werden einfach von der Challenge Nachricht übernommen)
<b>uri</b>	URI, die zur Challenge geführt hat (die Anfragezeile könnte von einem Proxy verändert worden sein)
<b>response</b>	Hash, der aus Nonce, Benutzername, Kennwort und möglicherweise weiteren Bestandteilen der Nachricht berechnet wurde
<b>cnonce</b>	Zufälliger Text des Clients, der in den Hash einbezogen wird und Chosen Plaintext Angriffe verhindert
<b>qop</b>	gewählte Sicherheitsstufe
<b>nonce-count</b>	ähler, wie oft der nonce in Requests verwendet worden ist
<b>auth-param</b>	z.Zt nicht benutzt

### 6.2.7 Cookies

Cookies werden in der Response vom Server im "Set-Cookie" oder "Set-Cookie2" (RFC2965) Header gesetzt. Sie bestehen aus Segmenten, die durch Semikolon voneinander getrennt sind. Erstes Segment ist ein Name "=" Wert Paar, das den Namen des Cookies festlegt. In weiteren Requests sendet der Client das Cookie im "Cookie" Header an den Server zurück, sofern im Cookie enthaltene Bedingungen erfüllt sind. Der Server kann über das Cookie verschiedene Requests demselben Client zuordnen.

Cookies werden in den Kopfzeilen (Header) von Anfragen und -Antworten via HTTP übertragen. Cookies entstehen, wenn bei einem Zugriff auf einen Webserver neben anderen HTTP-Kopfzeilen in der Antwort zusätzlich eine Cookie-Zeile übertragen wird (siehe Aufbau). Diese Cookie-Informationen werden dann lokal auf dem Endgerät gespeichert, üblicherweise in einer Cookie-Textdatei. Bei nachfolgenden weiteren Zugriffen auf den Webserver wird der eigene Browser alle Cookies in dieser Datei herausuchen, die zum Webserver und Verzeichnispfad des aktuellen Aufrufs passen, und schickt diese Cookie-Daten im Header des HTTP-Zugriffs mit zurück, womit die Cookies jeweils an jenen Webserver zurückgehen, von dem sie einst stammten.

### Felder im Cookie

<b>Comment</b>	Für Menschen lesbarer Kommentar zum Cookie
<b>Domain</b>	TDomain (oder ein Suffix beginnend mit .), für den das Cookie gilt
<b>Max-Age</b>	Nach dieser Zeit in Sekunden soll der Client das Cookie nicht mehr benutzen
<b>Path</b>	Pfad (oder ein Präfix), für den das Cookie gilt. Nur für URIs mit einem Pfad unterhalb des Attributes wird das Cookie benutzt
<b>Secure</b>	Versende das Cookie nur bei ausreichend hoher Sicherheit (z.B. bei verschlüsselter Verbindung)
<b>Version</b>	Version des Cookies (verbindlich = 1 für RFC2109, nicht benutzt für Netscape)

RFC2965 benutzt einige weitere Felder. Version ist ebenfalls 1.

#### Verarbeitung auf Clientseite

- Ist Domain nicht angegeben, wird der FQDN der URI benutzt.
- Ist Max-Age nicht angegeben, wird das Cookie bei Beenden des Clients verworfen
- Der Standardwert für Path ist der Path der aktuellen URI bis zum letzten “/”
- Ein fehlendes Secure wird als nicht gesetzt angenommen
- Cookies werden vom Client verworfen wenn:
  - Path kein Präfix des aktuellen Path ist
  - der Host nicht zur Domain gehört
  - Domain keinen eingeschlossenen Punkt enthält
  - Falls das Präfix, das zusammen mit Domain den FQDN des Hosts ergibt, einen Punkt enthält

## 6.3 TLS

### 6.3.1 RSA Verfahren

Das RSA Verfahren benutzt drei Zahlen  $N$ ,  $E$ ,  $D$  wobei das Paar  $(E, N)$  den öffentlichen und das Paar  $(D, N)$  den privaten Schlüssel bildet. Kenn man den öffentlichen Schlüssel, kann man eine Zahl  $m$ , die zwischen 0 und  $N - 1$  liegt verschlüsseln, indem man folgendes berechnet:

$$C = m^E \pmod N$$

Wer den privaten Schlüssel kennt, kann  $C$  wieder durch

$$m = C^D \pmod N$$

entschlüsseln.

Für die “modBerechnung von großen Zahlen gibt es zwei Formeln:

$$(a \cdot b) \pmod n = [(a \pmod n)(b \pmod n)] \pmod n \quad a, b \text{ müssen bestimmt gewählt werden}$$

$$(a + b) \pmod n = [(a \pmod n) + (b \pmod n)] \pmod n$$

Algorithmus	Beispiel
Bestimme zwei Primzahlen $P$ und $Q$	$P = 7, Q = 23$
Setze $N := P \cdot Q, \phi(N) := (P - 1)(Q - 1)$	$N = 7 \cdot 23 = 161,$ $\phi(N) = (7 - 1)(23 - 1) = 6 \cdot 22 = 132$
Bestimme ein $E$ , so dass der größte gemeinsame Teiler von $E$ und $\phi(N)$ 1 ist und $E < \phi(N)$ gilt (am besten Fermat Zahlen, $2^k + 1$ )	$E \in 3, 17, \dots, 65537, \dots$
Bestimme $D$ so, dass $D \cdot E \bmod \phi(N) = 1$ , d.h. prüfe ob $k \cdot \phi(N) + 1$ durch $E$ teilbar ist. Wenn ja ist das oft $D$	

Gelingt es einem Angreifer,  $N$  im Public Key zu faktorisieren, kann er den Private Key berechnen, was bei ungeschickter Wahl von  $P$  und  $Q$  einfach ist. Es ist weder bekannt, ob es keinen effizienten Algorithmus zur Faktorisierung von  $N$  gibt, noch ob RSA nicht auch durch andere Wege angreifbar ist.

### 6.3.2 Digitale Signatur

Um Integrität und Authentizität eines Dokumentes nachweisen zu können, kann es digital signiert werden. Derjenige, der die Echtheit beurkunden soll, ergänzt das Dokument um seine eigene Identität (beglaubigt durch). Aus den Daten berechnet er einen Hashwert nach einem vorher festgelegten und veröffentlichten Verfahren (z.B. mit MD5, SHA1, SHA256,...). Diesen Wert verschlüsselt er mit seinem Private Key und hängt das Ergebnis als Signatur an das Dokument an. Jeder kann nun die Echtheit des Dokumentes überprüfen, indem er mit dem Public Key die Signatur verschlüsselt und das Ergebnis mit dem Hash vergleicht.

### 6.3.3 Zertifikate

Problem bei öffentlichen Schlüsseln ist, dass man sich nicht sicher sein kann, ob der erhaltene öffentliche Schlüssel tatsächlich von der Person ist, von der man den Schlüssel haben möchte. Da sich in einem Rechnernetz die Teilnehmer nicht gegenseitig ausweisen können, basiert die Authentifizierung bei TLS auf vertrauenswürdigen Parteien. Eine vertrauenswürdige Partei erhält eine Datei mit dem Public Key des Servers und dessen Identifikationsmerkmalen (z.B. FQDN). Diese Daten werden geprüft und die Datei mit dem Schlüssel und den Informationen mit dem öffentlichen Schlüssel digital signiert. Dadurch entsteht ein Zertifikat.

Die Zertifikate bilden eine hierarchische Struktur von Vertrauensbeziehungen. An der Spitze der Hierarchie stehen die Root Certificate Authorities. Deren Zertifikate sind von ihnen selbst signiert. Eine Liste der Zertifikate ist in vielen TLS Clients und Servern als vertrauenswürdige vorkonfiguriert. Erhält ein Knoten ein Zertifikat, prüft er zuerst die Identifikationsinformationen der Gegenseite, dann die Signatur des Zertifikates. Um die Signatur prüfen zu können, benötigt er das Zertifikat der Instanz, die signiert hat. Dies wird fortgeführt, bis ein vertrauenswürdige Zertifikat erreicht ist.

## 6.4 Real-time Transport Protocol (RTP)

RTP wurde designed, um eine verbindungslosen Dienst für mehreren Teilnehmern durch eine Übertragung mit Zeitanforderung und möglichst geringer Latenz anzubieten. Dazu werden kurze Header für kleine Pakete benutzt, jedes Paket enthält einen Zeitstempel zur Berechnung der im Netzwerk entstandenen Laufzeitschwankungen und es können mehrere Datenströme (verschiedenster Medientypen) aufgeteilt und zusammengemischt werden. Zudem wird Isonchronität garantiert, d.h. es sollen gleiche Zeitabstände zwischen den Paketen bei Quelle und Ziel entstehen (Jitterausgleichsspeicher, siehe RTCP Interarrival Jitter).

RTP wird üblicherweise über UDP transportiert um (in echtzeit häufig unnötige) Wiederholungen von Paketen zu vermeiden und um QoS Garantien zu umgehen sowie Multicastunterstützung zu erhalten. Portnummern sind beliebig wählbar, doch folgt i.d.R. dem (geraden) RTP Port der (ungeraden) RTCP Port zum kontrollieren des Dienstes.

### 6.4.1 RTP Rahmen

0 - 1	2	3	4 - 7	8	9 -15
Version	P	X	CC	M	Payload Type
Sequence Number					
Timestamp					
SSRC Identifier					
CSRC Identifier (optional)					
Extension Header (optional)					
Data					

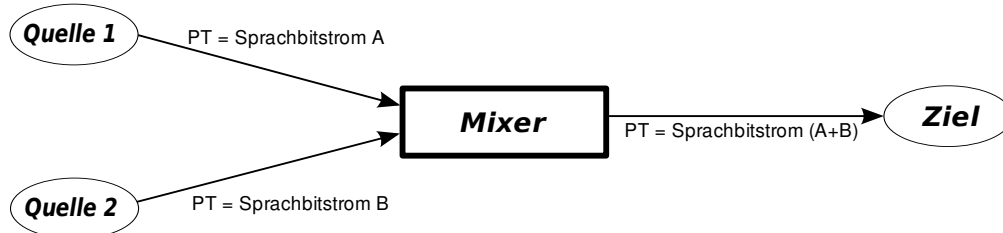
**Länge: 12 Byte**

<b>Version</b>	Version, aktuell 2
<b>P</b>	Padding, 1 wenn das Paket Füllbits enthält, das letzte Byte enthält dann deren Anzahl
<b>X</b>	Protokollerweiterungen werden benutzt (genau ein Erweiterungsheader)
<b>CC</b>	Anzahl an CSRC's (s.u.)
<b>M</b>	Marker, dessen Bedeutung vom Profil abhängt
<b>PT</b>	Payload Type, Art der Daten im Rahmen
<b>Sequence Number</b>	Zahl, die pro Paket um 1 erhöht wird, zufälliger Startwert
<b>Timestamp</b>	Zeitstempel der Erzeugung des ersten Bytes der Daten, zufälliger Startwert, Auflösung muss zur Synchronisation ausreichen
<b>SSRC</b>	(Synchronization Source) Eindeutige Kennung einer RTP Datenquelle
<b>CSRC</b>	(Contributed Source) Eindeutige Kennung einer RTP Datenquelle, wenn mehrere Quellen zu einem Strom gemischt wurden

Der **Erweiterungsheader** besteht aus 16 Bit privaten Daten und 4 Byte Längenangabe der darauf folgenden Daten und wird immer dann benutzt, wenn das Flag X gesetzt ist. Dadurch können profilabhängige Zusatzfunktionen implementiert werden. Um die Daten sinnvoll zu nutzen, wird die Aufteilung und Interpretation eines RTP Rahmens auf Anwendungsebene bearbeitet. RTP legt die unterstützten Datenformate sowie deren Interpretation fest (**Profile**). Die Aufteilung der Daten in Pakete/Rahmen geschieht bei RTP nicht in der Transportschicht oder Sicherungsschicht, sondern durch die Anwendung, um Interaktion mit dem Zeitverhalten der unteren Schichten zu vermeiden. Sie würden Latenz durch das Zusammenfassen von Paketen oder durch automatische Fehlerkorrektur hervorrufen. Damit wird die Kompetenz der Fehlerbehebung, Mehrfachübertragung, Paketverlust, Reihenfolge etc. an die Anwendungsschicht abgegeben, um diesen Problemen im Kontext des angebotenen Services zu begegnen (Application Level Framing). Es ergeben sich hieraus mehrere Anforderungen an den RTP Payload: die verwendete Quellkodierung muss sicherstellen, dass die Aufteilung auf die RTP Pakete so geschieht, dass der Datenstrom bei moderatem Paketverlust noch verwendbar ist, idealerweise sogar unabhängig von Vorgängern sein. D.h., ein RTP Payload sollte einen vollständigen Rahmen des Codecs enthalten und, sofern die MTU nicht überschritten wird, sollte die minimale Datengröße des Codecs nicht unterschritten werden.

Identifiziert wird der Payload durch das **Payload Type** Feld, welches sich zur Festlegung des Codecs einer Liste von standardisierten Quellkodierungen bedient. Diese Zahl wird von allen beteiligten Anwendungen (Sender, Empfänger, Mixer, Translator) interpretiert.

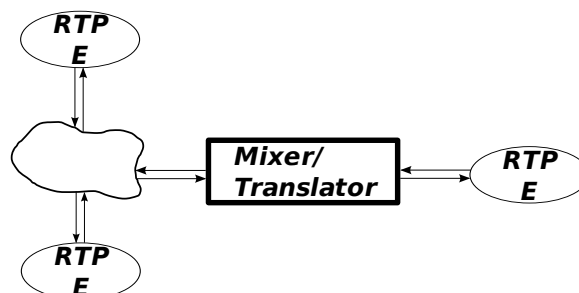
Der **Mixer** wandelt Datenströme um, kann sie z.B. für verschiedene Teilnehmer, die mit einem Netzwerk mit geringer Bandbreite angeschlossen sind, in einen Datenstrom bündeln, mit einem passenden Codec kodieren und dann weiterleiten. Inzwischen werden auch hierarchische Verfahren verwendet, die die Daten in aufeinander aufbauenden Kodierungsschichten (Layern) kodiert. Es können dann aufgrund von Verbindungsstörungen oder geringer Bandbreite höhere Schichten (mit Detailinformationen) bei Bedarf weggelassen werden, was natürlich zu Qualitätsverlust führt.



**Translatoren** ermöglichen es, Datenströme (sogar mit unterschiedlichen Kodierungen) für eine bestimmte Verbindung in ein geeignetes Format umzuwandeln und an ein Gegenstück auf der anderen Seite der Verbindung weiterleiten.



**translated, multicast to unicast** ist ganz doll wichtig und darum tu ich hier mal ein Bild davon rein.



## 6.4.2 Real-time Transport Control Protocol (RTCP)

RTCP ist das Protokoll, mit dem die Teilnehmer einer RTP Session Statusinformationen austauschen. Hierbei geht es vor Allem um

- Rückmeldung zur Übertragungsgüte mittels Sender Report (SR) und Receiver Report (RR)
- Identifikation der Teilnehmer über den "Canonical Name,, (CNAME), da sich die SSRC während einer Session ändern können. Außerdem wird er zur Korrelation der verschiedenen RTP Ströme genutzt.
- Anpassung der eigenen Senderate
- Unterstützung für Mehrpunktkommunikation

### RTCP Paket Typen

<b>SR</b>	(Sender Report) Sende- und Empfangsstatistiken
<b>RR</b>	(Receiver Report) Empfangsstatistiken von Teilnehmern, die aktuell nicht senden
<b>SDES</b>	(Source Description) Information über alle Quellen (besonders über CNAME)
<b>BYE</b>	Teilnehmer verlässt die Konferenz
<b>APP</b>	Anwendungsspezifische Daten

RTCP benutzt wie RTP den verbindungslosen Dienst der Transportschicht, üblicherweise UDP, wobei die Portnummer auf die Portnummer des RTP Stroms folgt. Es können mehrere RTCP Pakete in einem Transportpaket übermittelt werden, jedoch ist die Reihenfolge dann festgelegt und es muss wenigstens eine Empfangsstatistik enthalten (SR oder RR) sowie möglichst den CNAME. Pro Sendeintervall soll nur ein zusammengesetztes Paket pro Teilnehmer gesendet werden.

### RTCP Aufbau

<b>Encryption Prefix</b>	4 Byte Zufallszahl, falls Verschlüsselung benutzt wird
<b>SR/RR</b>	ggf. leere Sende- oder Empfangsstatistik
<b>RR</b>	Folge weiterer Empfangsstatistiken bei großen Konferenzen
<b>SDES</b>	mindestens der CNAME des Teilnehmers, möglicherweise weitere anwendungsspezifische Informationen
<b>BYE/APP</b>	weitere Informationen in beliebiger Reihenfolge

Das **RTCP Sendeintervall** beschreibt die Frequenz, mit der RTPC Pakete gesendet werden. Sie muss der Teilnehmerzahl an einer Konferenz angepasst werden, damit z.B. Bandbreite reserviert wird, um bei steigender Teilnehmerzahl die Datenrate zu erhöhen. Die genutzt Bandbreite wird für eine Session im voraus festgelegt und schließt die Transport- und Netzwerkschicht mit ein (z.B. UDP über IP). Für das Control Protocol wird ein fester, bekannter Prozentsatz der Gesamtbreite zur Verfügung gestellt (z.B. 5%) und wird im RTCP auf Sender und Empfänger mittels Profil festgelegt.

Berechnet wird das Sendeintervall unter Berücksichtigung folgender Punkte:

- proportional zur Anzahl der Teilnehmer
- Die Zeit der Erzeugung zweier RTCP Pakete liegt zufallsgesteuert zwischen dem 0.5- und 1.5-fachen des berechneten Intervalls
- Paketgrößen werden permanent mitgeschätzt
- kommen nahezu gleichzeitig viele neue Teilnehmer hinzu, wird RTCP verzögert
- ein BYE, welches auch vorzeitig gesendet werden darf, löst eine Neuberechnung des Intervalls aus, da es verkleinert werden kann

## Sender Report (SR)

Der SR besteht aus drei großen Blöcken: Header, Sender Info und Report Block

V	P	RC	PT	Length
SSRC of Sender				
NTP Timestamp				
RTP Timestamp				
Sender Packet Count				
Sender Byte Count				
SSRC of Source 1				
Fraction Lost		Number of Lost Packets		
Highest Sequence Number				
Interarrival Jitter				
Last SR				
Delay since Last SR				
Next Report Block				
Profile Specific Extension				

<b>V</b>	Version, aktuell 2
<b>P</b>	Padding, wie RTP
<b>RC</b>	Anzahl Report Blocks im Rahmen
<b>PT</b>	200 für RTCP SR
<b>Length</b>	Länge des Rahmens in 4 Byte Einheiten gezählt ab Sender Info (ab NTP Timestamp)
<b>SSRC of Sender</b>	Identifikation des Senders des RTCP Paketes
<b>NTP Timestamp</b>	Uhrzeit der Erzeugung des SR im Format des Network Time Protocols
<b>RTP Timestamp</b>	Gleiche Zeit wie im NTP Timestamp, aber in Skalierung und Offset des zugehörigen RTP Stroms
<b>Sender Packet Count</b>	Anzahl gesendeter Pakete seit Start der Session unter der aktuellen SSRC
<b>Sender Byte Count</b>	Anzahl gesendeter Bytes der Payload, d.h. keine RTP Header oder Padding
<b>SSRC</b>	Identifikation des Senders, zu dem die Statistik gehört
<b>Fraction Lost</b>	Verhältnis von verlorenen Paketen zu erwarteten Paketen multipliziert mit 256
<b>Number of Lost Packets</b>	Insgesamt verlorene Pakete von dieser SSRC im Verlauf der Session
<b>Highest Sequence Number</b>	Höchste empfangene Sequenznummer von der SSRC, erweitert durch 2 Byte für Überlauf
<b>Interarrival Jitter</b>	Varianz der Zwischenankunftszeiten
<b>Last SR</b>	0, falls noch kein SR empfangen wurde, sonst die mittleren 4 Byte des NTP Zeitstempels des letzten SR
<b>Delay since Last SR</b>	Anzahl der 165536-tel Sekunden seit Empfang des letzten SR; 0, falls undefiniert

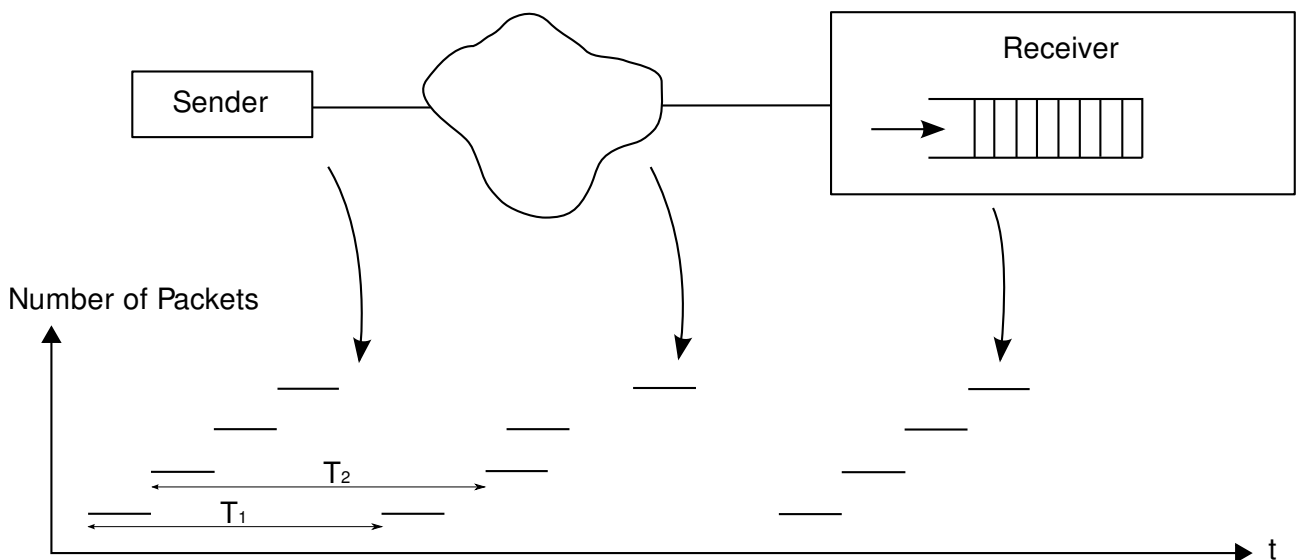
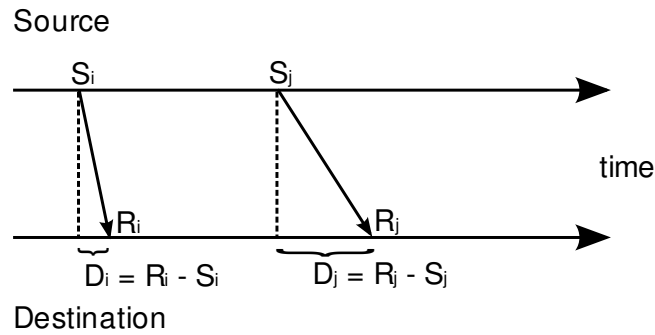
Der **SR/RR Interarrival Jitter** wird kontinuierlich nach einem Algorithmus geschätzt, damit er unabhängig vom Profil des RTP Stroms interpretiert werden kann.

Algorithmus: Sei  $S_i$  der RTP Zeitstempel des  $i$ -ten RTP Paketes,  $R_i$  die Ankunftszeit in derselben Einheit.

$$D(i, j) := (R_j - S_j) - (R_i - S_i)$$

$$J(i) := J(i - 1) + \frac{|D(i - 1, i)| - j(i - 1)}{16}$$

Gesendet wird immer jeweils  $J(i)$  für das aktuelle Paket  $i$ .



### Receiver Report (RR)

Der RR hat zwei Funktionen

1. als Erweiterung des SR, falls die 5 Bit des RC für mehr als 31 Empfänger nicht ausreicht
2. als Benachrichtigung eines reinen Empfängers

Das Datenformat ist ähnlich dem SR: Wert des PT ist 201, kein Sender Info Block

## Source Description (SDES)

V	P	SC	PT	Length
			SSRC/CSRC_1	
			SDES Items	
			SSRC/CSRC_2	
			SDES Items	

**SC** Source Count, d.h. Anzahl Chunks

**PT** SDES = 202

## SDES Chunks

Chunks haben immer die TLV Kodierung (Type, Length, Value)

**Type** 1 Byte, Typ der folgenden Daten

**Length** 1 Byte Länge der Daten

**Data** Nutzdaten in UTF-8, auf 4 Byte Grenze aufgefüllt

## SDES Typen

Number	Type
1	CNAME
2	NAME
3	EMAIL
4	PHONE
5	LOC
5	TOOL
6	PRIV
7	NOTE
8	PRIV

Der **SDES CNAME** Typ unterliegt besonderen Anforderungen, da er unabhängig vom Profil für die Funktion des Protokolls notwendig ist. Die SSRC Kennung kann zur Konfliktlösung und bei Programmneustart geändert werden, deshalb wird CNAME zur Identifikation des Teilnehmers verwendet und ist für den Verlauf der Session eindeutig und konstant. Über ihn können unterschiedliche Medienströme in verschiedenen RTP Sessions synchronisiert werden.

## RTCP Goodbye (BYE)

Das BYE Paket hat denselben Header wie SDES, es folgt lediglich eine Liste der SSRC/CSRC, die die Gruppe verlassen. PT ist 203, SC gibt Anzahl von SSRC/CSRC Feldern im Paket an. Optional kann ein Grund für das Verlassen der Gruppe am Ende angegeben werden.

## Application Defined (APP)

V	P	Subtype	PT	Length
			SSRC/CSRC_1	
			Name	
			Data	

**Subtype** Feld zur Unterscheidung verschiedener APP Pakete

**PT** APP = 204

**Name** 4 Byte ASCII Text

**Data** Daten abhängig vom Profil

## 6.5 Session Description Protocol (SDP)

SDP bietet eine Beschreibung einer Session mit allen Informationen, die von Clients benötigt werden, um an der Session teilzunehmen:

- Name und Zweck
- Zeitraum
- Medientyp
- Verbindungsinformationen (Host, Port, Multicastadressen)
- Bandbreitenanforderung

SDP Daten können über verschiedene Transportprotokolle (z.B. HTTP oder SIP) verbreitet werden. Eine SDP Nachricht besteht ausschließlich aus UTF-8 Text und hat immer einen Block zur Session eine möglicherweise leere Liste von Medienbeschreibungen. Die Grammatik ist als Augmented Backus Naur Form angegeben und muss strikt eingehalten werden, da sonst die Nachricht verworfen wird. Eine SDP Nachricht besteht aus "Parameter = Wert,, Paaren.

### Session Parameter von SDP

**v=** Protokollversion  
**o=** Identifikation des Initiators: Name SessionID SessionVersion Netztyp Adresstyp Adresse  
**s=** Name der Session  
**i=** (optionale) Sessioninformation  
**u=** (optionale) URI der Sessioninformation  
**e=** (optionale) E-Mail Adresse des Senders  
**p=** (optionale) Telefonnummer des Senders  
**c=** (optionale) Verbindungsrate (üblicherweise im Medienblock)  
**b=** (optionale) Bandbreitenanforderung (üblicherweise im Medienblock)  
 Beschreibung der Übertragungszeit bestehend aus:  
     **t=** Startzeitpunkt Stopzeitpunkt (als Sekunden seit 1900)  
     **r=** (optionale) Wiederholungszeitpunkte  
  
**z=** (optionale) Zeitzoneneinformation (für Offsets zum Tagesanfang bei Wiederholungen)  
**k=** (optionale) Schlüssel  
**a=** (optionale) Attribute

### Medien Parameter von SDP

**m=** Datentyp (z.B. audio/video), Quellport, Protokoll (z.B. RTP/AVP oder UDP) zusätzliche Parameter (z.B. RTP Payload Type)  
**i=** (optionale) Titel  
**c=** Verbindungsinformationen (optional, falls im Sessionblock)
 

- Netztyp
- Adresstyp (IPv4/IPv6)
- Adresse (IP Adresse / Multicastadresse)

  
**b=** (optionale) Bandbreiteninformation(en)  
**k=** (optionale) Schlüssel  
**a=** (optionale) Medienattribute

Name	Scope	Name	Scope
cat	Session	keywds	Session
tool	Session	pstime	Media
maxptime	Media	rtpmap	Media
recvonly	Either	sendrecv	Either
sendonly	Either	inactive	Either
orient	Media	type	Session
charset	Session	sdplang	Either
lang	Either	framerate	Media
quality	Media	fntp	Media

## 6.6 Session Initiation Protocol (SIP)

SIP ist ein Signalisierungsprotokoll mit den Zielen, eine Session auf- und abzubauen (e.g. für Telefonanrufe, Multimedia Streaming, Konferenzen). Hierbei werden SDP Nachrichten ausgetauscht. Es werden diverse Dienste auch durch verschiedene Infrastrukturen (Proxies, Router) für die Endnutzer unterstützt. Alle SIP Implementationen müssen TCP und UDP als Transportprotokoll unterstützen, der Standardport ist 5060, TLS ist optional auf Standardport 5061. Das Nachrichtenformat ist weitgehend mit HTTP identisch und grundsätzlich UTF-8, Chunked-Encoding ist aber nicht erlaubt.

### Transaktionen

Um Request und Response zuordnen zu können, werden auf Client- und Serverseite unterschiedliche Regeln verwendet.

**Client** Der *branch* im obersten *Via* sowie der *CSeq* Parameter muss zwischen Request und Response übereinstimmen

**Server** Der *branch* im obersten *Via* stimmt mit dem *branch* im ersten Request der Transaktion überein. Der *sent-by* Anteil im obersten *Via* stimmt überein. Die Methode im Request stimmt überein, außer bei der *INVITE* Methode.

### 6.6.1 SIP Dialog

Ein Dialog ist eine Beziehung zwischen zwei SIP Endpunkten, die für eine gewisse Zeit aufrecht erhalten wird. Dialoge werden durch eine Response mit Statuscode zwischen 100 und 299 erzeugt, deren *To*: Header ein *Tag* enthält. Die Zuordnung geschieht unter Verwendung der folgenden Attribute:

- Call-ID
- From Tag
- To Tag

### 6.6.2 Routing und Forwarding

#### Begriffe

<b>Client</b>	Ein SIP Knoten, der Requests verschickt und Responses empfängt
<b>Server</b>	Ein SIP Knoten, der Requests empfängt und Responses sendet
<b>User Agent (UA)</b>	Ein SIP Knoten, der am Ende der Verarbeitungskette steht. Man unterscheidet Client und Server.
<b>UA Client</b>	User Agent Client, erzeugt Request und verarbeitet Responses
<b>UA Server</b>	User Agent Server, verarbeitet Requests und erzeugt Responses
<b>Proxy</b>	Ein SIP Knoten, der sowohl als Client als auch als Server arbeitet und hauptsächlich Nachrichten routet

Jeder SIP Request muss folgende Felder enthalten:

<b>Request URI</b>	üblicherweise die URI im <i>To</i> Header
<b>To</b>	URI, z.B. SIP URI oder Tel URI des logischen Empfängers
<b>From</b>	URI des logischen Initiators eines Requests
<b>Call-ID</b>	Global eindeutige Kennung für eine Gruppe von Requests
<b>CSeq</b>	Sequenznummer und Methode des Requests
<b>Max-Forwards</b>	Maximale Anzahl Hops, Standard ist 70
<b>Via</b>	Transportadresse, zu der der Response gesendet wird. Der Branch Parameter dient der Zuordnung von Requests/Response zur Transaktion

### Client Routing

Zu welchen Servern ein SIP Request gesendet wird, hängt von folgenden Optionen ab:

1. lokale Konfiguration (Proxy ...)
2. gespeicherte Routen im Kontext des Dialoges
3. URI im obersten *Router* Header, falls vorhanden
4. Request URI

### Server Routing

- Response wird zur Quelladresse des Requestes zurückgesendet, wobei Port mit Via Header verwendet wird
- Bei verbindungsorientiertem Transport (TCP) ist die Verbindung zu nutzen, auf der der Request empfangen wurde
- Schlägt diese Verbindung fehl, wird Adresse und Transportprotokoll des ersten Via Headers verwendet
- Enthält der Via Header einen Namen anstelle einer IP, wird zuerst der SRV Record aufgelöst, dann der A bzw. AAAA Record

### Proxies

Man unterscheidet zwischen zwei Proxyausprägungen: *stateless* und *stateful*. Stateless Proxies dienen zur einfachen Weiterleitung von SIP Nachrichten, Stateful Proxies implementiert mindestens den SIP Transaktionsmechanismus (Client / Server). Er verhält sich gegenüber einem Server wie ein Client und gegenüber einem Client wie ein Server. Er prüft Requests und modifiziert Requests auf Eingangsseite und berechnet den Ziel(e). Weiterleitungen des Request stellen eine eigene Transaktion dar. Responses werden auch verarbeitet.

Die Minimalanforderungen zur Verarbeitung eines Requests sind

1. korrekte Syntax
2. angegebenes URI Schema muss unterstützt werden (Tel, SIP, SIPS)
3. Max-Forwards größer Null
4. optionale Prüfung auf Routing Schleife anhand Adresse in einem Via Header
5. Authentifizierung und Authorisierung

## Server Routing

Enthält der oberste Route Header eine Adresse des Proxies, wird dieser Eintrag entfernt. Damit wird *Loose Routing* implementiert. Entspricht die Request URI einer Adresse, die der Proxy in einem Record Route Header eingetragen hat (Umleitung), wird die Request URI durch den letzten Route Header überschrieben und der Eintrag gelöscht. Für die Request URIs, für die der Proxy zuständig ist, ermittelt er eine Menge von URIs, die als Ziel benutzt werden sollen (Target Set). Während der Weiterleitung kann das Target Set auch erweitert werden, z.B. durch Antworten auf weitergeleitete Requests (3xx Redirect). Ist der Proxy nicht für eine Domain zuständig für die Request URI, enthält das Target Set ausschließlich das in der Request URI genannte Ziel (Weiterleitung). Es ist dem Proxy überlassen, auf welche Weise und in welcher Reihenfolge die Einträge im Target Set zur Weiterleitung genutzt werden, es sind aber folgende Schritte nötig:

1. Kopieren des Requests: Nachrichtenbody bleibt unverändert, Reihenfolge der Header möglichst beibehalten, bei gleichem Namen verändern verboten
2. Anpassen der URI: die URI des Elementes des Target Sets ersetzt die aktuelle Request URI
3. Dekrementieren des Max-Forwards: fehlt Max-Forwards Header wird er mit Wert 70 hinzugefügt
4. ggfs. einen Route Record Header einfügen: Dass der Proxy Zwischenziel der Kommunikation ist, kann an (vorkonfiguriertem) Route Header liegen. Der Proxy kann sich mit einem Record Route Header in den Route Set eines Dialoges eintragen, wodurch sieht er die weiteren Requests und Responses in diesem Dialog.
5. weitere optionale Header einfügen
6. Auswerten der Routing Information, um den nächsten Hop zu bestimmen: bestimmt der Proxy, dass weitere Proxies durchlaufen werden sollen, werden deren URIs in Route Headern vor die bisherigen eingefügt. Um den nächsten Hop zu bestimmen muss (je nach Methode des UACs) ggfs. der Request für den Transport aufbereitet werden.
7. Via Header einfügen und weiterleiten: Abhängig vom gewählten Transport wird ein entsprechender Via Header eingefügt, um den nächsten Hop zu erreichen. Der Via Header muss einen Branch Parameter enthalten, da es sich um eine Clienttransaktion handelt. Bei verbindungsorientierten Transports ist ggfs. ein Content-Length Header nötig.

### 6.6.3 Verarbeitung der Response

Bei Empfang einer Response wird dieser weitergeleitet, falls der Response keiner Transaktion zugeordnet werden konnte. Konnte der Response eine Transaktion zugeordnet werden, werden folgende Schritte abgearbeitet:

1. Bestimmung des Kontextes
2. Entfernung des obersten *Via* Headers
3. Aktualisierung des Kontextes
4. Test, ob die Response sofort weitergeleitet werden muss
5. Im Fall, dass noch keine abschließende Antwort weitergeleitet worden ist, bestimme die geeignete Antwort.

Der Kontext kann über den *Branch* Parameter und den *CSeq* identifiziert werden. Der oberste *Via* Header wird entfernt. Falls kein weiterer *Via* in der Nachricht enthalten ist, ist die Response für den Proxy bestimmt. Damit darf er nicht weitergeleitet werden. Die Response wird zum Kontext hinzugefügt, bis eine abschließende Response der zugehörigen Servertransaktion gefunden ist. Responses mit Statuscode 1xx außer 100 (Trying) und mit Statuscode 2xx müssen sofort über die Servertransaktion weitergeleitet werden. Um eine Servertransaktion zu beenden, falls keine abschließende Antwort einer Clienttransaktion gesendet worden ist, wird bei Bedarf ein 408 (Request Timeout) erzeugt.

## 6.7 Robust Header Compression (ROHC)

Bei der Übertragung vieler kleiner Datenpakete z.B. in einem Stream kann ein großer Teil der tatsächlich übertragenen Daten aus identischen Header Informationen bestehen. Idee der Header Kompression ist es diese identischen Header Informationen so weit wie möglich einzusparen, d.h., z.B. nach Übertragung der initialen Werte nur noch die Änderungen zu übertragen. ROHC wird Beispielsweise benutzt für:

- Voice over IP
- Audio/Video Streaming
- interaktive Spiele

Es wird auch gerne bei Bandbegrenzte Funkverbindungen benutzt. Eine Einschränkung besteht darin, dass auch nach der Kompression die Verbindung eindeutig definiert bleiben muss. Das wird durch Header unterer Schichten realisiert, z.B. in der Sicherungsschicht. "Robust" heißt, dass die Qualität mindestens genau so gut sein muss wie ohne Header Compression. In typischen Anwendungen können mit ROHC mehr als 90% der Header Information eingespart werden.

Protocol Headers	Total Header Size (Byte)	Min. compressed Header Size	Compression Gain
IPv4/TCP	40	4	90%
IPv4/UDP	28	1	96,4%
IPv4/UDP/RTP	40	1	97,5%
IPv6/TCP	60	4	93,3%
IPv6/UDP	48	3	93,75%
IPv6/UDP/RTP	60	3	95%

### Funktionsweise

Der Sender und der Empfänger speichern und aktualisieren den kompletten Header (auch Kontext genannt). Dies ist die Basis für die Kompression. Wenn sich beim Sender etwas relevantes ändert, teilt er es dem Empfänger mit, der dann seinen Kontext aktualisiert. Ein Inkrement der Sequenznummer wird nicht mitgeteilt, da es keine relevante Änderung ist (kann vorhergesehen werden).

Es gibt 3 verschiedene Modi in denen sich eine Verbindung, die ROHC nutzt, befinden kann.

#### 1. Unidirectional Mode (U-Mode)

Pakete werden nur in eine Richtung verschickt, kein Feedback

#### 2. Bidirectional Optimistic Mode (O-Mode)

Feedback Kanal vorhanden, der aber nur für Fehlermeldungen und Acknowledgements für wichtige Änderungen des Kontexts benutzt wird

#### 3. Bidirectional Reliable Mode (R-Mode)

Intensive Nutzung des Feedback Kanals für höchste Qualitätsanforderungen (Ziel: Kein Synchronisationsverlust, niedrige Restfehlerrate)

Die Modi werden gewechselt, wenn ein entsprechendes Request vom Empfänger (genauer: vom Decompressor des Empfängers) über den Feedback-Kanal gesendet wird. Ein Compressor kann sich in 3 verschiedenen Zuständen befinden:

#### 1. Initialization and Refresh (IR):

Startzustand oder Zustand nach Reset  $\Rightarrow$  gesamter Header wird gesendet

## 2. **First-Order (FO):**

Compressor und Decompressor haben beide die statischen Headerelemente gespeichert  $\Rightarrow$  dynamische Headerelemente werden verschickt

## 3. **Second-Order (SO):**

Auch dynamische Headerelemente werden nicht mehr gesendet, sondern nur noch eine Sequenznummer und eine Checksum (1-2 Byte). Der Decompressor muss die dynamischen Headerelemente berechnen.

Auch der Decompressor kann sich in 3 Zuständen befinden:

### 1. **No Context:**

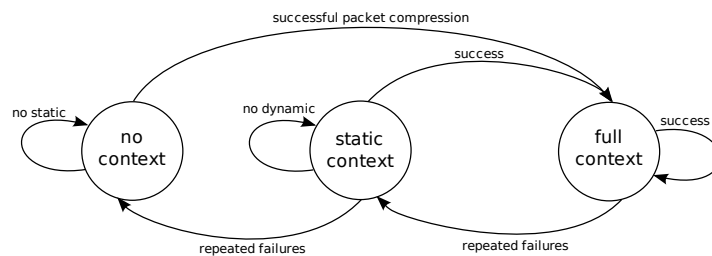
keine Header Information vorhanden, gesamter Header wird benötigt

### 2. **Static Context:**

statische Header Information vorhanden

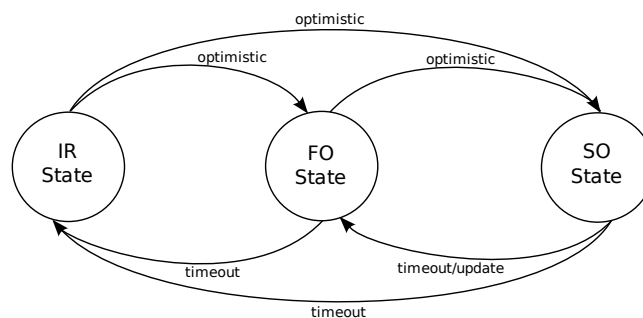
### 3. **Full Context:**

gesamte Header Information vorhanden



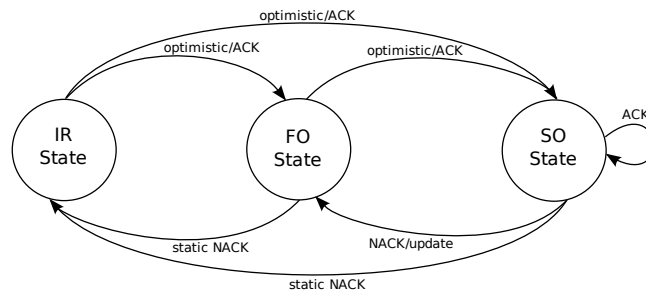
## 6.7.1 State machine des Unidirectional Mode

- Rückkehr vom Zustand SO nach IR und FO durch timeouts
- zum Schutz vor Fehlern in wichtigen Headerelementen werden in den Zuständen IR und FO mehrere Pakete gesendet bevor in den nächsten Zustand gewechselt wird



## 6.7.2 State machine des Bidirectional Optimistic Mode

- Decompressor kann ACKs für erfolgreich empfangene Headerelemente versenden
- Decompressor kann mit NACKs Fehler melden  
einfaches NACK  $\rightarrow$  Zustand FO  
static NACK  $\rightarrow$  Zustand IR



### 6.7.3 State machine des Bidirectional Reliable Mode

- Decompressor sendet ACK für jeden erfolgreich empfangene Änderung
- Compressor wechselt nur nach Empfang eines ACKs in FO oder SO Zustand
- durch die ACKs wird sichergestellt, dass Compressor und Decompressor immer im gleichen Zustand sind

