User Datagram Protocol (UDP)

Grundlagen

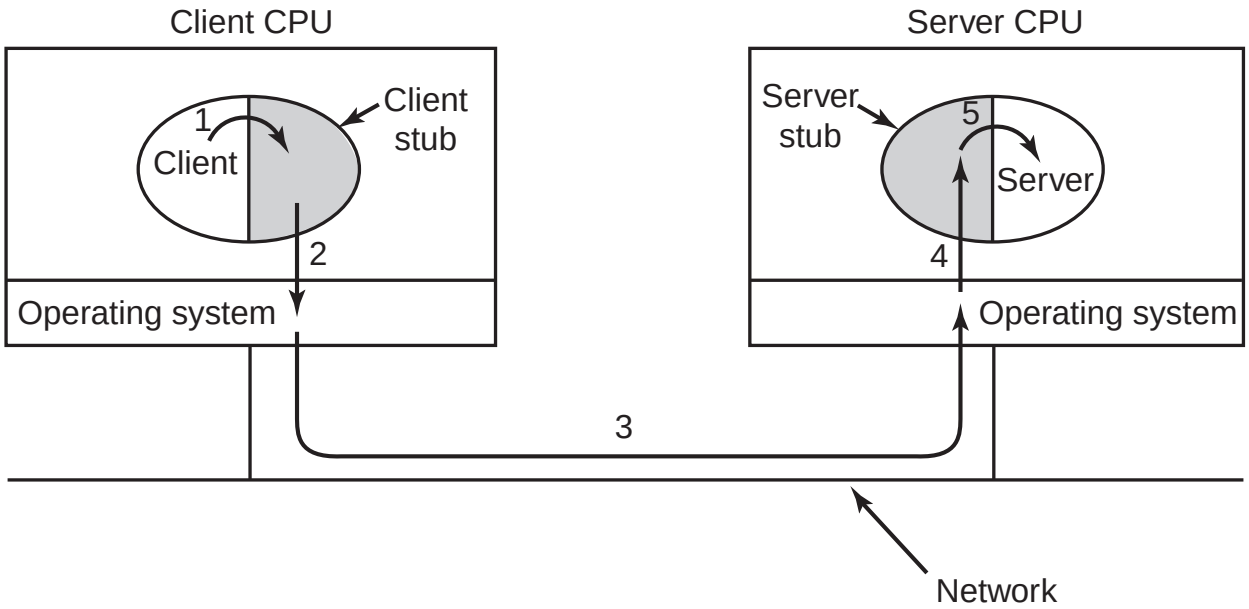
- Wert des Protokoll Feldes im IP Header ist 17 (vgl. RFC1700)
- Jedes UDP Datagramm erzeugt genau ein (möglicherweise fragmentiertes) IP Paket
- Die relevante Zusatzinformation durch UDP sind die Portnummern, die den Zweck des IP Pakets spezifizieren
- ungesicherter, verbindungsloser Transport (vgl. TCP: gesichert, verbindungsorientiert)
- UDP unterstützt Unicast und Multicast
- Daten sind bei UDP immer eine Folge von 8 Bit Zeichen

Anwendungsgebiete

- Schnittstelle zum IP Protokoll
- Demultiplexing von Übertragung auf der IP Schnittstelle auf verschiedene Ports
- UDP bietet keine Flußkontrolle, Fehlerkorrektur, Retransmission
- ermöglicht daher aber höheren Schichten ungefilterten Zugriff auf das IP Protokoll, sodaß höhere Schichten, z.B. die Anwendungsschicht, eigene Verfahren implementieren können
- Nutzung für
  - real-time Multimedia Anwendung (Streaming,...) (z.B. Real-Time Protocol, RTP)
  - kurze Client-Server Requests (z.B. DNS, Domain Name Service) (z.B. Remote Procedure Call, RPC)

Remote Procedure Call (RPC)

- ermöglicht Funktionsaufrufe auf anderen Rechnern
- identische Syntax wie auf lokalem Rechner
- problematisch sind übergebene Variablen (z.B. Pointer)



(c) Tanenbaum, Computer Networks

UDP Rahmen

Source Port	Destination Port
UDP Length	UDP Checksum
UDP Data	

Felder im UDP Header

- **Source Port:** Nummer, mit der die sendende Applikation vom Host identifiziert werden kann.
- **Destination Port:** Nummer, mit der die empfangende Applikation vom Host identifiziert werden kann.
- **UDP Length:** 16 Bit Länge des gesamten Datagramms in Bytes
- **Checksum:** 16 Bit Prüfsumme des gesamten Datagramms zuzüglich eines Pseudoheaders

UDP Prüfsummenberechnung

Dem UDP Datagramm wird ein Pseudoheader vorangestellt, bei ungerader Anzahl Bytes mit einer 0 aufgefüllt und dann das für IP üblicher Verfahren angewendet. Ist das Ergebnis 0, wird 0*xFFFF* eingesetzt, ist das Feld 0, wurde es nicht berechnet.

<table>
<tr><td colspan="3">Source IP Address</td></tr>
<tr><td colspan="3">Destination IP Address</td></tr>
<tr><td>0</td><td>Protocol</td><td>UDP Length</td></tr>
<tr><td colspan="2">Source Port</td><td>Destination Port</td></tr>
<tr><td colspan="2">UDP Length</td><td>UDP Checksum</td></tr>
<tr><td colspan="3">(Padded) UDP Data</td></tr>
</table>

Größe von UDP Datagrammen

- Der UDP Header erlaubt 65507 Bytes Daten (8 Byte UDP Header, 20 Byte IP Header)
- Ein Host muß IP Pakete von mindestens 576 Byte empfangen können (RFC791), d.h. mindestens 548 Byte UDP Daten.
- Liest eine Anwendung erfolgreich empfangene Daten, kann das Datagramm gekürzt werden.
- Der Zwischenspeicher kann von der Anwendungsschicht vergrößert oder verkleinert werden.

Die meisten UDP basierten Protokolle gehen davon aus, daß maximal 512 Byte Daten pro Datagramm übertragen und verarbeitet werden können.

Berkeley Sockets
(BSD Socket API)

Standard

- Entstanden an der Berkeley University of California (Berkeley Software Distribution, BSD) (Application Programming Interface, API)
- Veröffentlicht als Programmierschnittstelle des BSD4.2 (1983)
- Inzwischen standardisiert als IEEE Std. 1003.1-2004, als Single Unix Specification der Open Group und als ISO/IEC 9945:2002
- Die standardisierte C Schnittstelle ist aus verschiedenen Programmiersprachen nutzbar.

Umfang

Durch den Standard werden Schnittstellen beschrieben, die folgende Aspekte des Zugriffs auf die Transportschicht regeln:

- Adressierung, Adressumwandlung
- Verbindungsaufbau/Verbindungsabbau
- Konfiguration der Interna einer Verbindung
- Konfiguration der möglichen Zugriffsmethoden
- Datentransfer

Adressierung

Die BSD Socket API bietet Datenstrukturen und Konvertierungsfunktionen für Adressen verschiedener Formate:

- `inet_ntop, inet_pton`: Umwandlung von IP Adressen in binäre Darstellung (IPv4 und IPv6)
- `getaddrinfo`: Umwandlung eines symbolischen Namens in Adresse(n) der Transportschicht
- `getpeername`: Liefert die Adresse der Gegenseite einer Verbindung

Verbindungsaufbau/Verbindungsabbau

- `socket`: Belegt Resource (sog. Socket) für den Zugriff auf die Verbindung
- `bind`: Setzt Quelladresse der Verbindung
- `connect`: Setzt Zieladresse und baut ggfs. Verbindung auf (vgl. active open)
- `listen, accept`: Läßt Socket auf Verbindungsaufbau warten (vgl. passive open)
- `shutdown`: Schließt eine Richtung einer Verbindung
- `close`: Schließt eine Verbindung

Datentransfer

- `select, poll`: Testet, ob ein Ereignis auf dem Socket wartet.
- `send, sendto, sendmsg`: Sende Daten, abhängig vom gewählten Transport
- `recv, recvfrom, recvmsg`: Empfange Daten, abhängig vom gewählten Transport

Konfiguration

Die Konfiguration der Vermittlungsschicht und Transportschicht erfolgt über `setsockopt` und `fcntl`. Es folgt eine Auswahl der portablen Optionen:

- `SO_BROADCAST`: Erlaube Broadcasts bei UDP
- `SO_ERROR`: Lese Fehler mit `getsockopt`
- `SO_KEEPALIVE`: Schalte TCP Keepalive ein
- `SO_LINGER`: Warte, bis alle Daten bei einem `close` oder `shutdown` erfolgreich übertragen sind
- `SO_RCVBUF`: Größe des Empfanspuffers
- `SO_RCVLOWAT`: Mindestmenge an Daten, die zur Anwendung hochgereicht werden
- `SO_RCVTIMEO`: Timeout beim Lesen von Daten aus dem Empfangspuffer

Konfiguration (2)

- `SO_REUSEADDR`: Verwende Adresse neu, ohne 2 MSL Timeout abzuwarten
- `SO_SNDBUF`: Größe des Sendepuffers
- `SO_SNDLOWAT`: Minimale Datenmenge bei Sendeoperationen
- `SO_SNDTIMEO`: Timeout beim Senden
- `SO_TYPE`: Abfrage des Verbindungstyp/Protokolls

- `TCP_MAXSEG`: Setze MSS
- `TCP_NODELAY`: Schaltet Nagle Algorithmus aus
- `O_NONBLOCK`: Aufrufe auf dem Socket blockieren nicht mehr.

Client

```
from socket import *
from ReadUntilEOF import *

c_so = socket(AF_INET, SOCK_STREAM)
c_so.setsockopt(SOL_SOCKET, SO_REUSEADDR,1)
c_so.bind( ( "0.0.0.0", 7777 ) )

c_so.connect( ( "localhost", 6666 ) )

c_so.sendall("client request line 1\n")
c_so.sendall("client request line 2\n")
c_so.shutdown(SHUT_WR)

print readUntilEOF(c_so)

c_so.close()
```

Server

```
from socket import *
from ReadUntilEOF import *

a_so = socket(AF_INET, SOCK_STREAM)
a_so.bind( ( "localhost", 6666 ) )
a_so.listen(10)

s_so, addr = a_so.accept()
a_so.close()

print "Connection from ", addr
print readUntilEOF(s_so)
s_so.sendall("server response line 1\n")
s_so.sendall("server response line 2\n")

s_so.close()
```

Ablauf

- S: `a_so = socket(AF_INET, SOCK_STREAM)`
  - Erzeuge Socket für TCP Verbindung
- S: `a_so.bind( ( "localhost", 6666 ) )`
  - Registriere Socket mit Port 6666 und Quell-IP 127.0.0.1
- S: `a_so.listen(10)`
  - Socket wechselt in den Zustand LISTEN
  - 10 Warteplätze für eingehende Verbindungen
- S: `s_so, addr = a_so.accept()`
  - Applikation wartet, bis eine vollständig aufgebaute Verbindung besteht

Ablauf

- C: `c_so = socket(AF_INET, SOCK_STREAM)`
  - Datenstruktur für TCP Verbindung wird angelegt
  - IP Header Version ist 4
  - IP Header Protocol ist 6 (TCP)
- C: `c_so.setsockopt(SOL_SOCKET,SO_REUSEADDR,1)`
  - Socket im Zustand TIME_WAIT kann benutzt werden
- C: `c_so.bind( ( "0.0.0.0", 7777 ) )`
  - IP Header Source IP wird anhand der Routing Tabelle ermittelt
  - TCP Header Source Port ist 7777
  - Socket ist im Zustand CLOSED

Ablauf

- C: `c_so.connect( ( "localhost", 6666 ) )`
  - IP Header Source IP ist 127.0.0.1
  - TCP Header Destination Port ist 6666
  - Client sendet
    IP 127.0.0.1.7777 > 127.0.0.1.6666: S
    2476749965:2476749965(0)
  - Socket ist im Zustand SYN_SENT
- S: Empfängt SYN
  - Daten werden in der Listen Queue gespeichert
  - Server antwortet
    IP 127.0.0.1.6666 > 127.0.0.1.7777: S
    2976505067:2976505067(0) ack 2476749966
  - Socket ist im Zustand SYN_RCVD

Ablauf

- C: Empfängt SYN-ACK
  - Prüft (Host,Port) Paar und Sequenznummer
  - Client antwortet
    IP 127.0.0.1.7777 > 127.0.0.1.6666: . ack 2976505068
  - Socket ist im Zustand ESTABLISHED
- S: Empfängt ACK
  - Holt Verbindungsdaten aus der Warteschlange
  - Socket ist im Zustand ESTABLISHED
  - `s_so, addr = a_so.accept()` liefert Filedescriptor

Ablauf

- S: `print "Connection from ", addr`
  - Gibt Source IP und Source Port aus
  - Die Daten könnten auch über `s_so.getpeername()` ausgelesen werden.
- S: `print readUntilEOF(s_so)`
  - Liest Daten vom Filedescriptor, bis EOF signalisiert wird

Ablauf

- C: `c_so.sendall("client request line 1\n")`
  - Schreibt 22 Bytes in den Puffer des Sockets
  - Client sendet
    IP 127.0.0.1.7777 > 127.0.0.1.6666: P
    2476749966:2476749988(22) ack 2976505068
- C: `c_so.sendall("client request line 2\n")`
  - Schreibt 22 Bytes in den Puffer des Sockets
  - Client sendet
    IP 127.0.0.1.7777 > 127.0.0.1.6666: P
    2476749988:2476750010(22) ack 2976505068
- C: `c_so.shutdown(SHUT_WR)`
  - Client sendet
    IP 127.0.0.1.7777 > 127.0.0.1.6666: F
    2476750010:2476750010(0) ack 2976505068
  - Socket ist im Zustand FIN_WAIT_1

Ablauf

- S: Empfängt ”client request line 1\n”
  - Bestätigt mit
    IP 127.0.0.1.6666 > 127.0.0.1.7777: . ack 2476749988
  - Applikation wartet weiter auf Daten
- S: Empfängt ”client request line 2\n”
  - Bestätigt mit
    IP 127.0.0.1.6666 > 127.0.0.1.7777: . ack 2476750010
  - Applikation wartet weiter auf Daten
- S: Empfängt FIN
  - Socket ist jetzt im Zustand CLOSE_WAIT
  - Applikation liest EOF vom Filedeskriptor
  - `s_so.sendall(”server response line 1\n”)`
  - Server sendet
    IP 127.0.0.1.6666 > 127.0.0.1.7777: P
    2976505068:2976505091(23) ack 2476750011

Ablauf

- S: `c_so.sendall(""server response line 2\n")`
  - Schreibt 23 Bytes in den Puffer des Sockets
  - Server sendet
    IP 127.0.0.1.6666 > 127.0.0.1.7777: P
    2976505091:2976505114(23) ack 2476750011
- C: empfängt beide Zeilen
  - Bestätigt mit
    IP 127.0.0.1.7777 > 127.0.0.1.6666: . ack 2976505114
- S: `s_so.close()`
  - Server sendet
    IP 127.0.0.1.6666 > 127.0.0.1.7777: F
    2976505114:2976505114(0) ack 2476750011
- C: Empfängt FIN
  - Client bestätigt:
    IP 127.0.0.1.7777 > 127.0.0.1.6666: . ack 2976505115

Application Layer

Domain Name System (DNS)

Standards

Das Domain Name System bildet ein verteiltes Verzeichnis zur Umwandlung von Namen und Adressen.

Der Internet Standard 13 (DOMAIN) umfaßt

- **RFC1034** Domain Names - Concepts and Facilities
- **RFC1035** Domain Names - Implementation and Specification

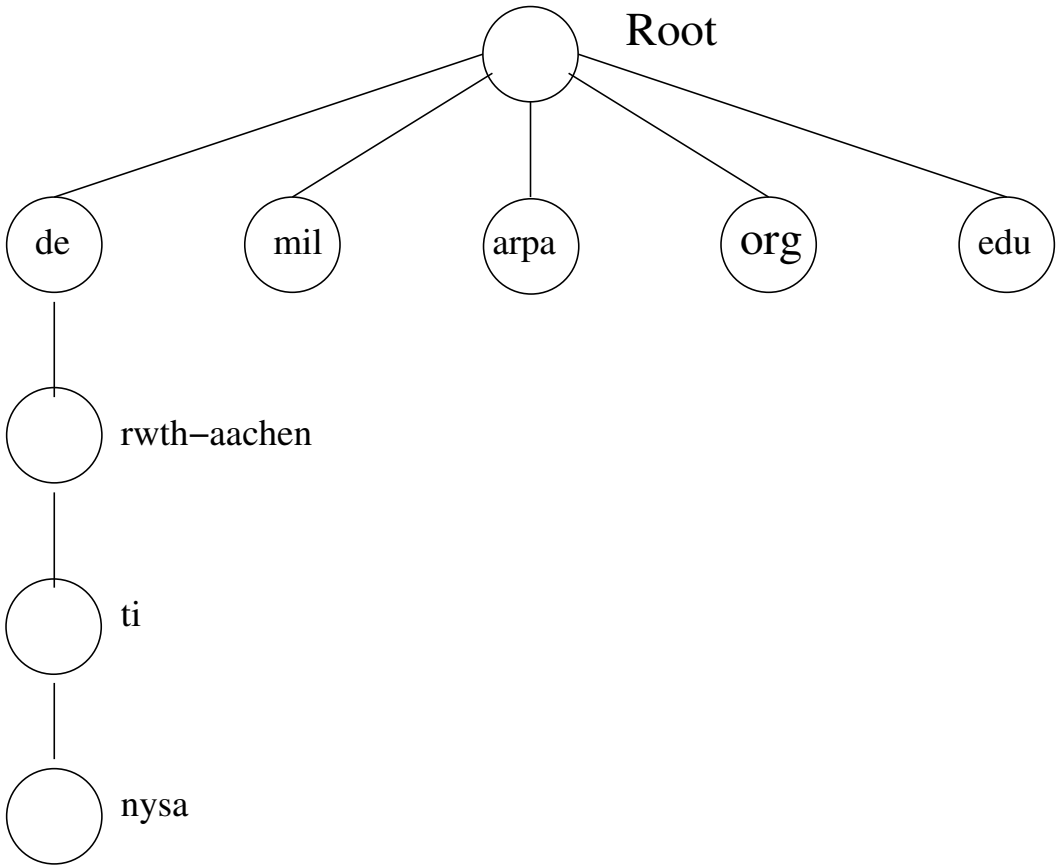
Eine Vielzahl von Erweiterungen finden sich in jüngeren RFCs (vgl. `http://www.dns.net/dnsrd/rfc/`) und sind zum Teil in aktuellen DNS Servern implementiert.

Syntax für Namen

Die dem DNS zugrunde liegende Datenbank hat einen hierarchischen Aufbau, der sich in den Namen widerspiegelt:

- Namen bestehen aus Folgen von Bezeichnern (Label), die maximal 63 Zeichen lang sind.
- Die Bezeichner werden durch einen Punkt (’.’) getrennt.
- Zwischen Groß- oder Kleinschreibung wird nicht unterschieden.
- RFC1035 empfiehlt für Label, mit einem Buchstaben (a-z oder A-Z) zu beginnen, dann Buchstaben, Ziffern oder ’-’ und nicht mit einem ’-’ zu enden.
- Namen, die mit einem Punkt (’.’) enden, werden als vollständig angenommen, andernfalls ist eine Erweiterung nötig, um zum **Fully Qualified Domain Name, FQDN** zu kommen.

Verzeichnisaufbau



Top Level Domain

Second Level Domain

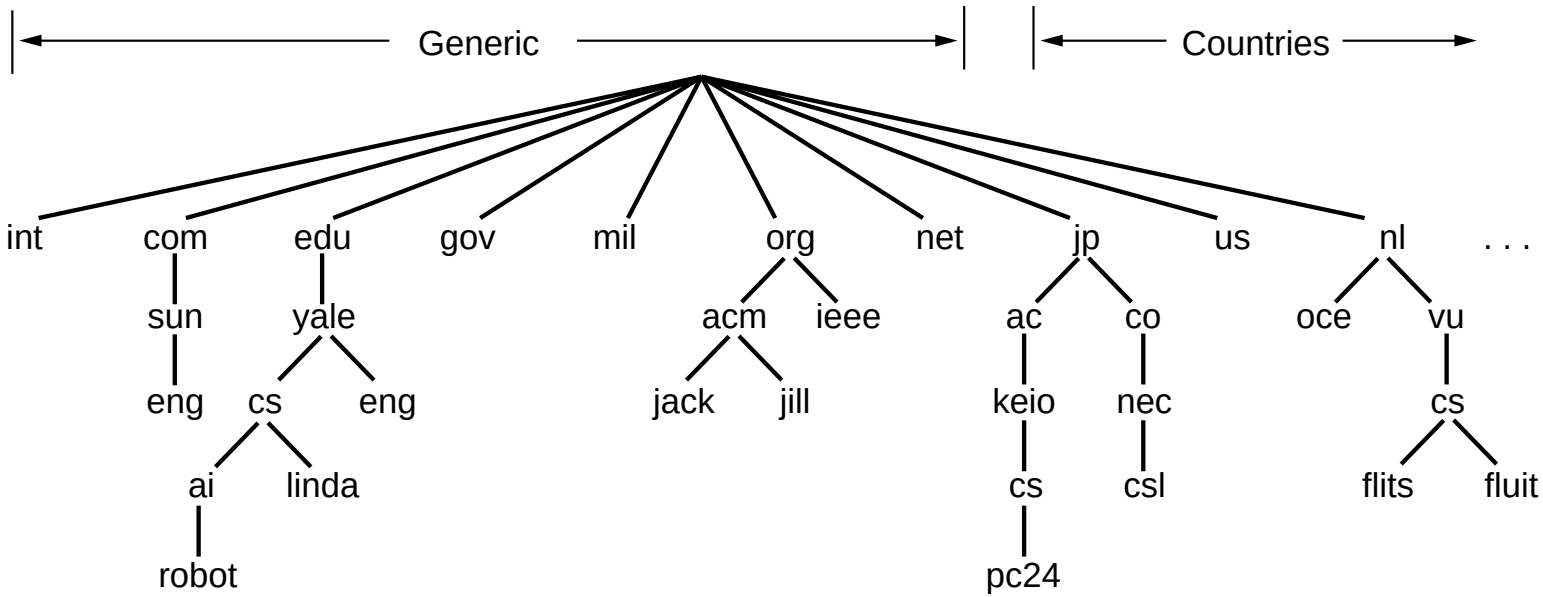
Verzeichnisaufbau

Namen werden in umgekehrter Reihenfolge des Labels geschrieben, d.h. die Top Level Domain (TLD) zuletzt.

Beispiele:

- **nysa** oder **RWTH-Aachen**: Bezeichner gemäß RFC1035
- **nysa.ti.rwth-aachen.de.**: FQDN
- **nysa.ti**: Unvollständiger Name
- **182.35.130.134.in-addr.arpa.**: FQDN

Top Level Domains



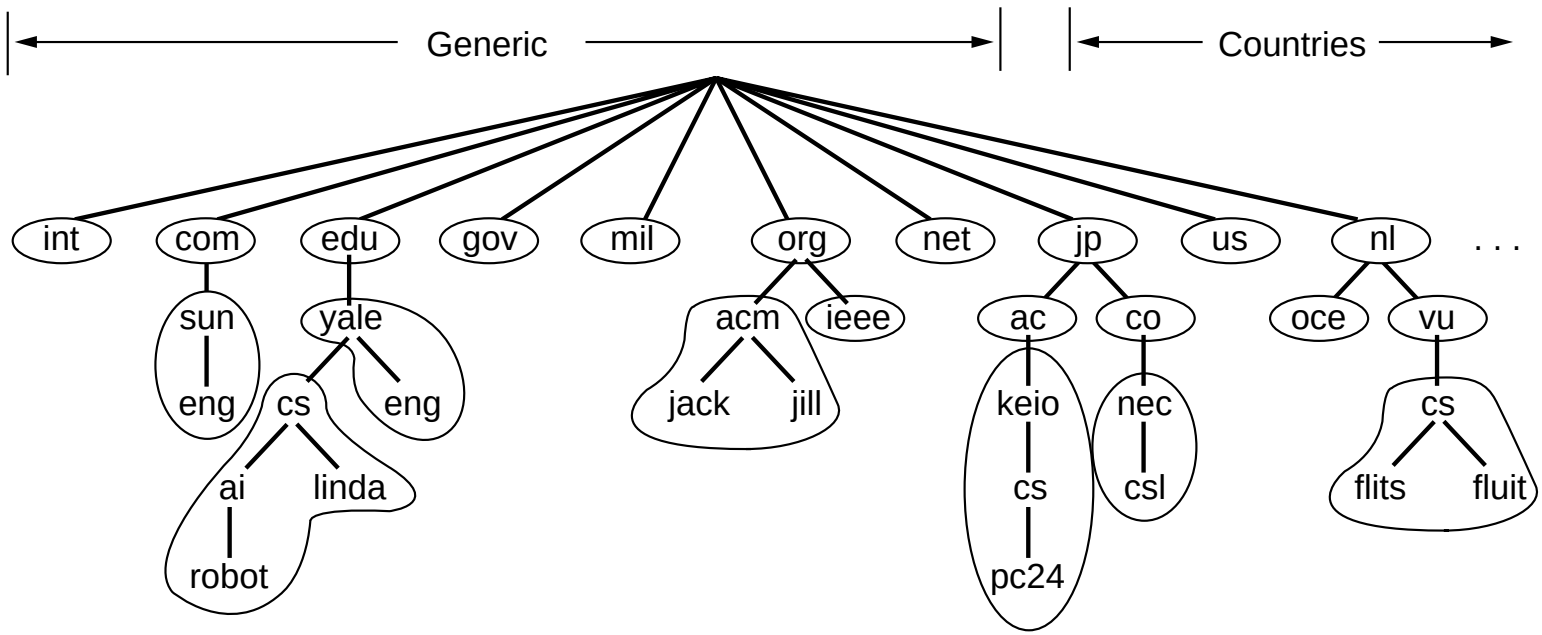
(c) Tanenbaum, Computer Networks

Top Level Domains

- **Country Coded TLD**: ISO 3166 Ländercode, Administrativer Kontakt unter `http://www.iana.org/root-whois/index.html`
- **Generic TLD**: Namen, die einer bestimmten Organisation/Verwendung zugeordnet sind, z.B. `.com`, `.edu`, Administrativer Kontakt unter `http://www.iana.org/gtld/gtld.html`
- **Infrastucture TLD**: TLD für Namen, die aus technischen Gründen benutzt werden, `.arpa` erzeugt eine separaten Baum, `.root` wird nur als Marker der Rootzone verwendet.

```
$host -t TXT \
  vrsn-end-of-zone-marker-dummy-record.root
```

DNS Zonen



(c) Tanenbaum, Computer Networks

DNS Zonen

- Zonen sind separat administrierbare Unterbäume des Verzeichnisses
- Der Administrator einer Zone ist dafür verantwortlich, DNS Server für diese Zone bereitzustellen
- Der Administrator einer Zone kann die Verwaltung von Unterbäumen an andere Administratoren delegieren.
- Primary DNS Server einer Zone ist der (ggfs. redundant aufgebaute) Server, auf dem die Konfigurationsdaten der Zone administriert werden.
- Eine Zone kann weitere Secondary DNS Server haben, die die Konfigurationsdaten vom Primary DNS Server herunterladen (sog. Zone Transfer).
- Ein DNS Server ist Authoritative, wenn er eine aktuelle Kopie der Zone hat.

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	

Felder im DNS Rahmen

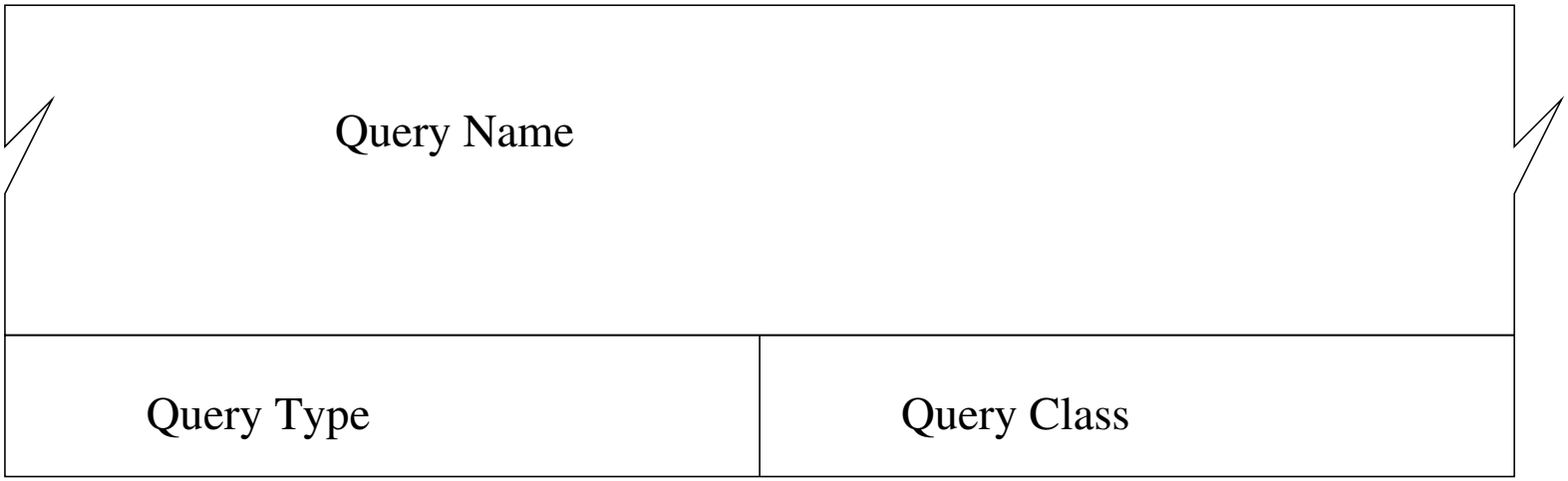
- **Identification**: 16 Bit Code, mit dem die Antwort einem Request zugeordnet werden kann.
- **Flags**: 16 Bit, beschreibt Art der Antwort, Fehlerstatus, ...
- **Number of Questions**: 1 für DNS Anfragen, 0 für Antworten
- **Number of RRs**: Anzahl Resource Records (Antworten auf eine Anfrage)
- **Number of Authority RRs**: Anzahl Authority Records (DNS Server, die die gesuchte Information sicher haben)
- **Number of Additional RRs**: Anzahl Additional Records (Zusatzinformation, z.B. die Adressen der DNS Server aus den Authority Records)

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



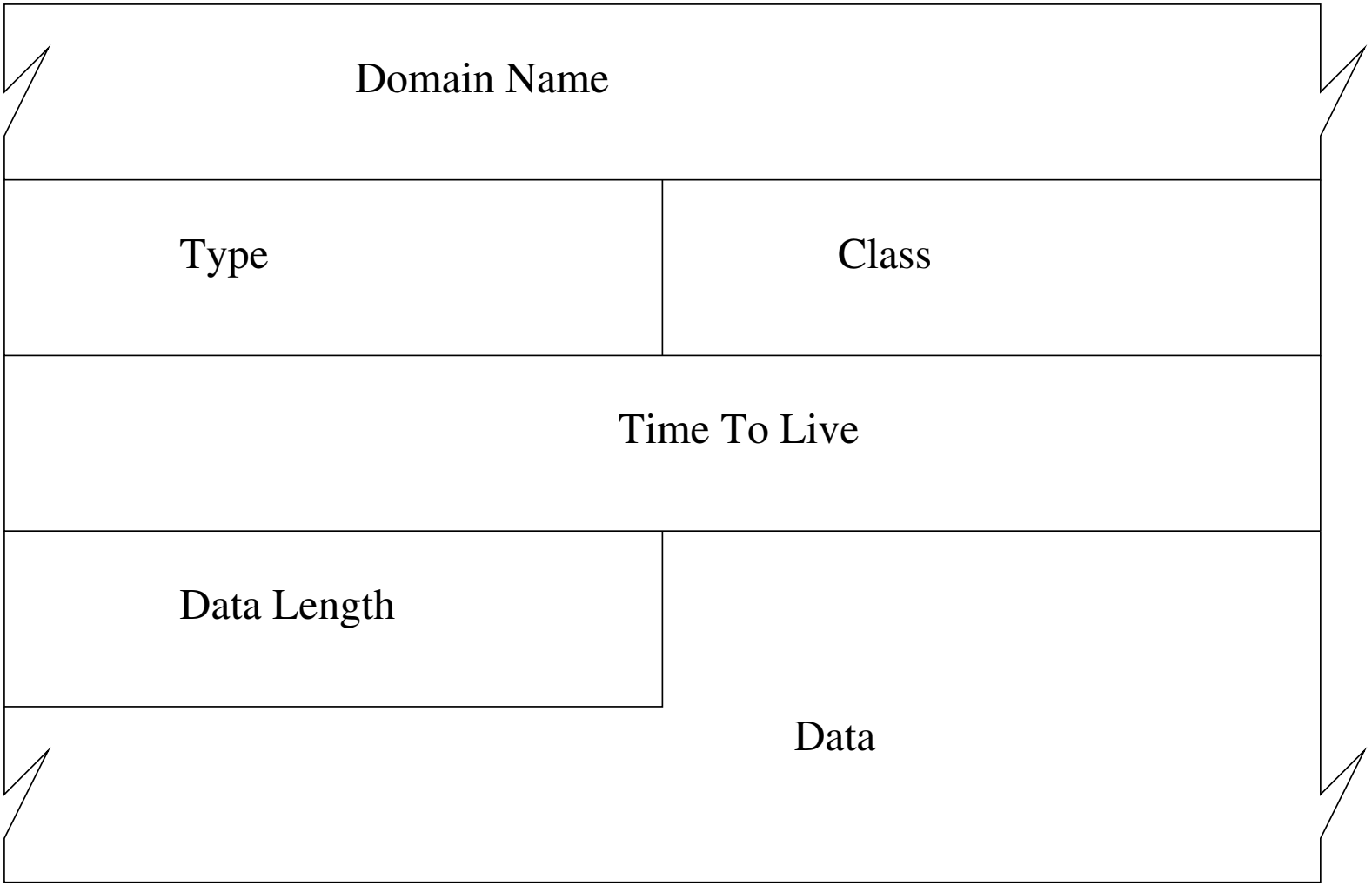
- **Query Name**: Name, der abgefragt wird
- **Query Type**: Typ der Anfrage, entweder Standardtyp oder
  - **ANY**: Jeder Standardtyp
  - **AXFR**: Zonentransfer
- **Query Class**: 1 für Internet (IN)

Standardtypen

Hier ist eine Auswahl der gebräuchlichsten Datensätze

Typ	Wert	Name
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-Mailserver)
AAAA	28	IPv6 Adresse (RFC1933)
SRV	32	Server Record (Host:Port des Servers)
NAPTR	35	Naming Authority Pointer (RFC2915)

Resource Records



Felder eines Resource Records

- **Domain Name**: Name, zu dem die Daten gehören
- **Type**: Standardtyp der Daten
- **Class**: 1 für IN
- **Time To Live**: Zeit in Sekunden, die der Eintrag von einem DNS Server oder Client aus dem Cache verwendet werden darf.
- **Data Length** Länge in Byte der folgenden Daten
- **Data**: Daten und Kodierung abhängig vom Typ.

Transport

- Üblicherweise werden DNS Anfragen per UDP gestellt
- Zeitüberschreitungen und Wiederholungen werden von den Clients gesteuert.
- Zonentransfers finden über TCP statt (aber nur zu den Secondary DNS Servern)
- DNS Server binden sich üblicherweise auf Port 53
- Der Quellport bei Anfragen ist beliebig.
- Sind mehr als 512 Byte zu übertragen, muß der Client via TCP erneut nachfragen (TC Flags ist in der Antwort gesetzt).

Beispiel

Abfrage eines SRV Records (TTL und Query Class):

```
$ dig -t SRV _xmpp-client._tcp.jabber.org
;; QUESTION SECTION:
;_xmpp-client._tcp.jabber.org.   SRV
;; ANSWER SECTION:
_xmpp-client._tcp.jabber.org.    SRV   5222 jabber.org.
;; AUTHORITY SECTION:
jabber.org.                       NS   ns2.jeremie.com.
jabber.org.                       NS   ns1.jeremie.com.
;; ADDITIONAL SECTION:
jabber.org.                        A   208.245.212.98
ns1.jeremie.com.                   A   208.245.212.29
```

Auflösung von Namen

- Ein Host stellt die Anfrage bei den (bis zu 3) konfigurierten Nameservern (RD ist üblicherweise 1)
- Hat ein DNS Server die Antwort im Cache, sendet er die zugehörigen Daten
- Kann er die Anfrage nicht lokal beantworten, fragt er (nicht rekursiv) einen der konfigurierten ROOT Server.
- Antwort ist eine Liste der Authoritative DNS Server der zugehörigen Zone in den Authority RRs.
- Eine Anfrage bei einem dieser DNS Server führt entweder zum Ergebnis, oder zu einer weiteren Liste von DNS Servern einer untergeordneten Zone.
- Dies wird fortgesetzt, bis ein Server die Anfrage beantworten kann.

Beispiel mit Rekursion

```
$ dig www.heise.de
;; QUESTION SECTION:
;www.heise.de.                  IN   A
;; ANSWER SECTION:
www.heise.de.           74272   IN   A   193.99.144.85
;; AUTHORITY SECTION:
heise.de.               74272   IN   NS  ns.pop-hannover.d
heise.de.               74272   IN   NS  ns.heise.de.
heise.de.               74272   IN   NS  ns2.pop-hannover.
;; ADDITIONAL SECTION:
ns.pop-hannover.de.     10562   IN   A   193.98.1.200
ns2.pop-hannover.net.   75522   IN   A   62.48.67.66
ns.heise.de.            85574   IN   A   193.99.145.37
```

Beispiel ohne Rekursion

```
$ dig +norec www.heise.de @198.41.0.4
;; QUESTION SECTION:
;www.heise.de.                  IN    A
;; AUTHORITY SECTION:
de.                     172800  IN    NS    Z.NIC.de.
de.                     172800  IN    NS    A.NIC.de.
de.                     172800  IN    NS    C.DE.NET.
;; ADDITIONAL SECTION:
A.NIC.de.               172800  IN    A     194.0.0.53
C.DE.NET.               172800  IN    A     208.48.81.43
Z.NIC.de.               172800  IN    A     194.246.96.1
Z.NIC.de.               172800  IN    AAAA  2001:628:453:4905::53
```

Beispiel ohne Rekursion (2)

```
$ dig +norec www.heise.de @194.246.96.1
;; QUESTION SECTION:
;www.heise.de.                  IN   A
;; AUTHORITY SECTION:
heise.de.               86400   IN   NS   ns.heise.de.
heise.de.               86400   IN   NS   ns.pop-hannover.de.
heise.de.               86400   IN   NS   ns2.pop-hannover.ne
;; ADDITIONAL SECTION:
ns.heise.de.            86400   IN   A    193.99.145.37
ns.pop-hannover.de.     86400   IN   A    193.98.1.200
```

Beispiel ohne Rekursion (3)

```
$ dig +norec www.heise.de @193.99.145.37
;; QUESTION SECTION:
;www.heise.de.          IN   A
;; QUESTION SECTION:
;www.heise.de.  IN      A
;; ANSWER SECTION:
www.heise.de.   86400  IN   A   193.99.144.85
;; AUTHORITY SECTION:
heise.de.       86400  IN  NS   ns.pop-hannover.de.
heise.de.       86400  IN  NS   ns2.pop-hannover.net.
heise.de.       86400  IN  NS   ns.heise.de.
;; ADDITIONAL SECTION:
ns.heise.de.    86400  IN   A   193.99.145.37
```

Reverse Lookup

- IP Adressen in Dotted Notation bilden Namen im DNS Verzeichnis
- Die TLD ist arpa, die Second Level Domain ist in-addr.
- Eine IP Adresse a.b.c.d wird in der Zone administriert als **d.c.b.a.in-addr.arpa**
- Der Typ eines in-addr.arpa Eintrages ist PTR, der Wert ist der FQDN des Hosts mit der entsprechenden Adresse.
- Die Auflösung des Namens **d.c.b.a.in-addr.arpa** erfolgt wie für alle FQDN

Beispiel

Der Host `www.heise.de` hat die IP Adresse `193.99.144.85`

```
$ dig -t PTR 85.144.99.193.in-addr.arpa
;; QUESTION SECTION:
;85.144.99.193.in-addr.arpa.          PTR
;; ANSWER SECTION:
85.144.99.193.in-addr.arpa.   PTR   www.heise.de.
;; AUTHORITY SECTION:
144.99.193.in-addr.arpa.      NS    ns.s.plusline.de.
144.99.193.in-addr.arpa.      NS    ns.heise.de.
144.99.193.in-addr.arpa.      NS    ns.plusline.de.
;; ADDITIONAL SECTION:
ns.heise.de.                  A     193.99.145.37
ns.s.plusline.de.             A     212.19.40.14
ns.plusline.de.               A     212.19.48.14
```

Reverse Lookups und CIDR

- Die Segmente eines FQDN geben die Ebenen der Administrativen Kontrolle an.
- Dieser Mechanismus funktioniert nicht beim Reverse Lookup, wenn Netze eine Netzmaske haben, die nicht auf einer Bytegrenze endet.
- RFC2317 beschreibt eine mögliche Konfiguration des Parent DNS Servers, der die Kontrolle an untergeordnete Server weiterleitet.
- Einige DNS Server haben eigene Lösungen, die Delegation zu implementieren (z.B. BIND9 $GENERATE).

Beispiel

Unterhalb der Domain `test.net` mit Adressbereich `1.2.3.0/24` wird eine Domain `sub.test.net` mit Adressen `1.2.3.128/25` angelegt.
Auszug aus dem Zonenfile der `test.net` Domain (RFC1035):

```
$ORIGIN test.net
@     NS      ns.test.net.
ns    A       1.2.3.1
sub   NS      ns.sub.test.net.
```

```
$ORIGIN 3.2.1.in-addr.arpa
1     PTR     ns.test.net.
128   CNAME   128.3.2.1.sub.test.net.
129   CNAME   129.3.2.1.sub.test.net.
```

Beispiel Fortsetzung

Das Zonenfile der `sub.test.net` Domain:

```
$ORIGIN sub.test.net
@       NS    ns.sub.test.net.
ns      A     1.2.3.129
host2   A     1.2.3.130
129     PTR   ns
130     PTR   host2
```

DNS Load Balancing

- Server können nur eine begrenzte Anzahl paralleler Verbindungen bedienen.
- Viele Dienste erfordern/erlauben, daß Verbindungen lange geöffnet bleiben (z.B. IMAP IDLE, vgl. RFC2177).

Die Lösung besteht darin, auf den Clients den Namen (nicht die IP) des Servers zu konfigurieren. Bei jedem Verbindungsaufbau wird der Name in eine Adresse umgewandelt, wobei die TTL des Eintrages beachtet wird. Dadurch kann der Administrator später weitere Server hinzuschalten.

Beispiel: Round Robin DNS

```
$ host www.google.de
 www.google.de      CNAME  www.google.com
 !!! www.google.de CNAME record has zero ttl
 www.google.com     CNAME  www.l.google.com
 !!! www.google.com CNAME record has zero ttl
 www.l.google.com     A    209.85.135.104
 www.l.google.com     A    209.85.135.147
 www.l.google.com     A    209.85.135.99
 www.l.google.com     A    209.85.135.103
```

Bei weiteren Anfragen ist die Reihenfolge der Resource Record möglicherweise anders.

Hypertext Transport Protocol
HTTP

HTTP 0.9

Die Urversion des Hypertext Transport Protocols bietet ein einfaches Request-Response Modell aufbauend auf einer TCP Verbindung.

- Client baut eine TCP Verbindung auf, der Default für den Zielport ist 80.
- Er sendet eine Anfragezeile der Form
  GET Path?Search, abgeschlossen durch ein CRLF (ASCII Codes 13, 10, Carriage Return Line Feed)
- Der Server antwortet mit einem Textdokument mit HTML Markup (`http://www.w3c.org/MarkUp/`).
- Das Ende des Dokuments wird durch Schließen der Verbindung angezeigt.

Aktueller Standard

Aktuell ist HTTP in der Version 1.1 (RFC2616). Die Änderungen zur ersten Version machen es zu einem vielseitig verwendbaren Transport Protokoll.

- Neben Text mit HTML Markup werden andere Datentypen und Formate unterstützt.
- Gegenseitige Authentifizierung der Kommunikationspartner ist möglich.
- Virtual Hosting ermöglicht verschiedene, unabhängige Dienste auf einem Server hinter einer einzigen IP Adresse.
- Für den Nutzer transparente Daten- und Transportkodierungen bieten effiziente Nutzung der Serverresourcen.
- HTTP unterstützt Infrastruktur, die den Einsatz aus privaten Netzen ermöglicht.

SMTP Header (RFC822)

Das Format einer HTTP Nachricht entspricht weitgehend einer E-Mail (SMTP, Simple Mail Transfer Protocol), d.h. nach der Anfragezeile folgen Name-Wert Paare (sog. Header), dann - durch eine Leerzeile getrennt - die Nutzdaten. Zeilenumbrüche bestehen immer aus CRLF (ASCII Codes 13, 10).

- Namen und Werte der Header sind durch Doppelpunkt und Leerzeichen voneinander getrennt.
- Bei den Namen wird nicht zwischen Groß- und Kleinschreibung unterschieden.
- Mehrere Header mit gleichem Namen sind möglich, wenn die Syntax für den Wert eine Folge von durch Komma getrennten Werten vorschreibt.
- Die Reihenfolge von Headern ist nur von Bedeutung, wenn es Header mit gleichem Namen sind.

Header Folding

Lange Zeilen (mehr als 72 Zeichen) sollen zur Verbesserung der Lesbarkeit vermieden werden. Header Zeilen müssen dazu umgebrochen werden. Dazu wird an LWSP (Linear White SPace), d.h. Leerzeichen (SPACE) oder Tabulator (HT) getrennt.

- Enthält der Wert eines Headers ein LWSP, kann es durch CRLF + LWSP ersetzt werden (Folding).
- Beginnt umgekehrt eine Headerzeile mit einem LWSP, ist es die Fortsetzung der vorausgehenden Zeile. Dann muß vor der weiteren Verarbeitung das vorausgehende CRLF entfernt werden.

Beispiel

```
GET /Test HTTP/1.1
Host: localhost:8080
Accept: text/html, text/plain, text/css,
  text/sgml, */*;q=0.01
Accept-Encoding: gzip
Accept-Language: en
User-Agent: Lynx/2.8.6rel.4 libwww-FM/2.14
  SSL-MM/1.4.1 GNUTLS/1.6.2
```

Statuscodes

HTTP benutzt wie SMTP Statuscodes bestehend aus 3 Ziffern mit folgender Interpretation:

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

Hex/Base16 Kodierung (RFC3548)

RFC822 (SMTP) erlaubt nur den (7 Bit) ASCII Zeichensatz. Um Binärdaten in Headern zu übertragen, müssen diese daher auf Zeichenketten aus dem ASCII Alphabet abgebildet werden.

Bei der Hex Kodierung werden die 16 möglichen Werte einer 4 Bit Folge abgebildet auf die 16 Zeichen 0123456789abcdef.

Damit wird jedes Byte dargestellt als 2 ASCII Zeichen.
Beispiel: Die Bytes 127, 62 werden dargestellt als `7F 3E`

Beim Dekodieren wird nicht zwischen Groß- und Kleinschreibung unterschieden.

Base64 Kodierung (RFC3548)

Base16 Kodierung ist bei großen Datenmengen ineffizient, da sie die Datenmenge (gemessen in Byte) verdoppelt. Die Base64 Kodierung ist in diesem Fall um 33% besser.

- Je 6 Bit werden als ein Zeichen kodiert.
- Zielalphabet sind die Zeichen A-Z, a-z, 0-9 und “+”, “/” in dieser Reihenfolge (z.B. $27_{10} = 011011_2 = b_{64}$).
- Da immer je 24 Bit kodiert werden müssen, werden Bytefolgen mit 0 Bits aufgefüllt.
- 6 Füllbits werden durch ’=’ dargestellt.

Beispiel: Die Bytes 127, 62 werden dargestellt als `fz4=`, denn: 127,62 entspricht den Bits 011111 110011 1110`00` `000000`. Die je 6 Bit entsprechen den Zahlen 31(f), 51(z), 56(4) und Füller(=).