

Lustiges Implementieren von RFCs

Domain Name System (DNS)

Inhalt

#1 WAS SIND RFCS?

Ein kurzer Überblick von RFCs

#2 WAS IST DNS?

High-Level Übersicht und Historie von DNS

#3 LESEN (UND VERSTEHEN) DER RFCS

Theoretische Grundlagen von DNS mit Hilfe von RFCs verstehen

#4 RAN ANS WERK

Implementierung eines DNS Servers

#5 FRAGERÜNDCHEN

All eure Fragen :)

Was sind RFCs?

RFCs

- RFC is die Kurzschreibweise für 'Request for Comments', also 'Bitte um Kommentare'
- Eine Reihe technischer Dokumente zum Internet
- RFCs werden in 3 Kategorien eingeteilt: STD, BCP und (FYI) Eingestellt

Einige RFCs, aber nicht alle, definieren Internetstandards, wie zum Bsp.:

- IP [RFC 791](#)
- UDP [RFC 768](#)
- und DNS [RFC 1035](#)

RFCs

EDITORS / VERFASSTER

- Internet Engineering Task Force (IETF)
- Internet Architecture Board (IAB)
- Internet Research Task Force (IRTF)
- Unabhängige Beitragende

STATUS

- Proposed Standard, Draft Standard, Internet Standard
- Best Current Practice
- Historic / Obsolete
- Informational
- Experimental
- Unknown
- Draft

Was ist DNS?

"The Domain Name System (DNS) is the hierarchical and decentralized naming system used to identify computers, services, and other resources reachable through the internet"

Wikipedia, [Domain Name System](#)

Geschichtsstunde

DIE LAGE VOR DNS

[RFC 1034 \(SECTION 2.1\)](#)

- Alle Verknüpfungen von Host Namen zu IPs wurden von dem NIC in EINER! `HOSTS.TXT` Datei verwaltet, welche von ALLEN Hosts über FTP abgerufen wurde.
- Durch den großen Wachstum an Hosts geriet dieses System schnell an seine Grenzen

HOST TABLE FORMAT

[RFC 952](#)

```
NET : 10.0.0.0 : ARPANET :  
NET : 128.10.0.0 : PURDUE-CS-NET :  
GATEWAY : 10.0.0.77, 18.10.0.4 : MIT-GW.ARPA,MIT-GATEWAY :  
PDP-11 : MOS : IP/GW,EGP :  
HOST : 26.0.0.73, 10.0.0.51 : SRI-NIC.ARPA :  
DEC-2060 : TOPS20 : TCP/TELNET,TCP/ECHO,ICMP :
```

TOPS-20 ist ein proprietäres Betriebssystem für DEC 36bit Mainframes auf Basis von TENEX und ist in Assembly geschrieben. (Erstveröffentlichung: 1976)

Geschichtsstunde

PROTOKOLL

TCP Verbindung über Port 101 auf Host ``SRI-NIC.ARPA`` (26.0.0.73 oder 10.0.0.51)

REQUESTS [RFC_953](#)

```
<command key> <argument(s)> [<options>]
```

```
HNAME SRI-NIC.ARPA <CRLF>
```

- Text Query Requests (in Plain Text)
- Verschiedene Command Keys, wie ``HNAME``, ``ALL`` oder ``DOMAINS``

RESPONSE [RFC_953](#)

```
<response key> : <rest of response>
```

```
HOST : 26.0.0.73, 10.0.0.51 : SRI-NIC.ARPA,SRI-NIC,NIC :  
      DEC-2060 : TOPS20 : TCP/TELNET,TCP/SMTP,TCP/TIME,  
      TCP/FTP,TCP/ECHO,ICMP :
```

- Antworten in Plain Text
- Verschiedene Response Keys, wie ``NET``, ``HOST`` oder ``ERR``
- Error Codes: ``NAMNFD``, ``ADRNFD``, ``ILLCOM`` und ``TMPSYS``

Vergleich

DNS

[RFC_1035](#)

- Primär über UDP (aber auch TCP)
- Baumartige Namesstruktur
- Jedes Blatt kann 0 - n Dateneinträge beinhalten, sog. Resource Records, kurz RRs
- Client- / Serverarchitektur
- Verteilt

DOD INTERNET HOST TABLE SPECIFICATION

[RFC_952](#)

- <- Nur TCP
- <- Tabellen Struktur
- <- Jeder "Record" ist eine Zeile
- <- Auch Client- / Serverarchitektur
- <- Zentral

Lesen (und verstehen) der RFCs

DNS RFCs

Es gibt zwei grundlegende RFCs, welche das Protokoll DNS definieren:

- Domain Names - Concepts and Facilities [RFC 1034](#)
- Domain Names - Implementation and Specification [RFC 1035](#)

Mit der Zeit (und den wachsenden Ansprüchen an DNS) sind weitere ergänzende RFCs hinzu gekommen:

- A Means for Expressing Location Information in the Domain Name System (LOC RR) [RFC 1876](#)
- Incremental Zone Transfer in DNS [RFC 1995](#)
- Extension Mechanisms for DNS (EDNS(0)) [RFC 6891](#)
- und weitere ...

Ziele von DNS

[RFC 1034](#) beschreibt diverse Ziele:

Verteilte Infrastruktur und lokales Caching

The sheer size of the database and frequency of updates suggest that it must be maintained in a distributed manner, with local caching to improve performance. [...]

Abrufen von diversen Daten, welche nicht nur einen Anwendungsfall abdecken

[...] that it be generally useful, and not restricted to a single application. We should be able to use names to retrieve host addresses, mailbox data, and other as yet undetermined information. [...]

Nutzung von UDP und TCP

We want name server transactions to be independent of the communications system that carries them. [...]

Resource Records (RRs)

DEFINITION [RFC 1034 \(SECTION 3.6\)](#)

A domain name identifies a node. Each node has a set of resource information, which may be empty. The set of resource information associated with a particular name is composed of separate resource records (RRs). [...]

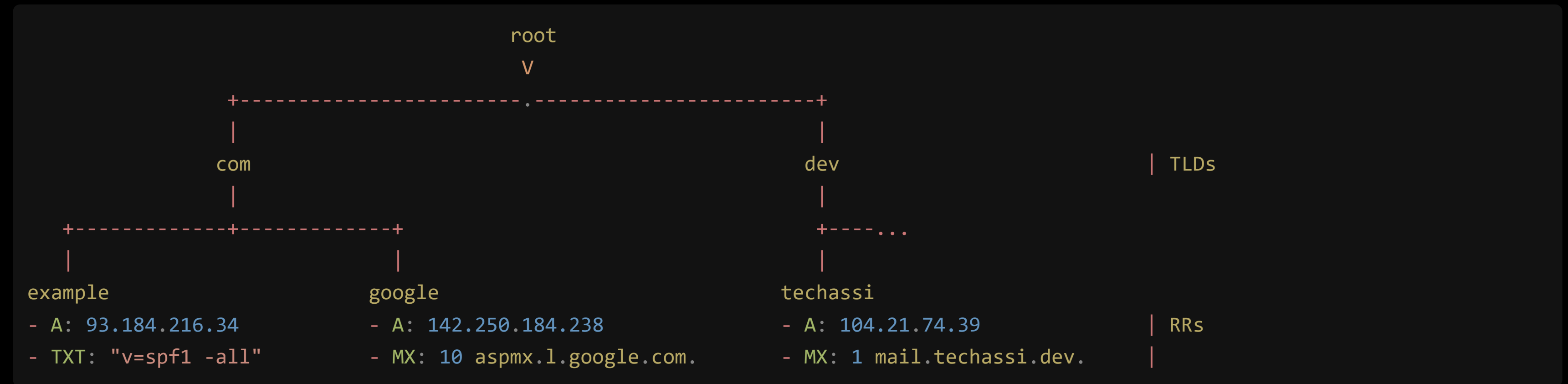
Resource Records beschreiben Daten, welche zu einer Node in einem Baum gehören.

AUFBAU EINES RRS [RFC 1034 \(SECTION 3.6\)](#)

OWNER	which is the domain name where the RR is found.
TYPE	which is an encoded 16 bit value that specifies the type of the resource in this resource record.
CLASS	which is an encoded 16 bit value which identifies a protocol family or instance of a protocol.
TTL	which is the time to live of the RR. [...]
RDATA	which is the type and sometimes class dependent data which describes the resource [...]

Baumstruktur der Domain Names

RFC 1034 empfiehlt die Implementierung von Zonen und Caches als Baumstruktur.



Zusammenfassung

In *RFC 1034* werden theoretische Grundlagen von DNS definiert und erklärt. Im Gegensatz dazu beinhaltet *RFC 1035* Details zur technischen Implementierung. In der nächsten Sektion werden wir fast nur noch mit diesem arbeiten.

- Domain Namen sind in einer Baumstruktur organisiert
- Jedes Blatt (Node) kann 0-n Resource Records beinhalten
- RRs beschreiben Daten zu dieser Node
- RRs haben verschiedene Typen, Klassen und Inhalte

Ran ans Werk

Server / Client Architektur

SERVER

Ein DNS Server übernimmt folgende Aufgaben:

- Bearbeitung von einkommenden DNS Anfragen (Queries / Questions)
- Anfragen können auf Grund von diversen Gründen abgelehnt werden
- Anfragen nach unbekannten Domain Names können auf diverse Arten behandelt werden
- Server-seitiges Caching
- Logging von Anfragen
- noch mehr? Bestimmt...

CLIENT

Ein DNS Client ist für folgende Aufgaben zuständig:

- Aufkommende DNS Anfragen (von User Space Programmen) bearbeiten
- DNS Query erstellen und an Server senden
- Antworten interpretieren
- Lokales Client-seitiges Caching

Fachjargon

BIT

Kleinste binäre Einheit: Kann Wert *0* oder *1* annehmen

OCTET

8 Bit: `00 00 00 00` oder `10 00 00 01`

Warum nicht Byte? Früher war ein Byte nicht unbedingt 8 Bit. Bytes haben früher Werte zwischen 4 und 10 Bits angenommen. Aufgründdessen hat man sich entschieden die Einheit *Octets* zu nehmen (Octo => Acht in Griechisch und Latein).

ENDIANNESS

Endianness oder *Most significant bit (MSB)* gibt an, welches Bit das höchste ist:

- Big Endian: Das höchste Bit ist links. `10 00 00 00` => 128
- Little Endian: Das höchste Bit ist rechts. `10 00 00 00` => 1

DNS arbeitet mit dem *Big Endian System* (ist auch die gängigste Schreibweise)

RFC 1035 (3.2.1)

Das 'Wire' Format gibt an, wie Daten in binärer Form über das 'Wire' (Kabel) transportiert werden.

RR Types

[RFC 1035 \(3.2.2\)](#)

RR Typen definieren die Art von Resource Records. Umgangssprachlich Record. Einige bekannte Typen sind:

A (1)

Eine IPv4 Host Adresse

```
93.184.216.34
```

NS (2)

Ein Authoritative Name Server

```
a.iana-servers.net.
```

CNAME (5)

Canonical Name für ein Alias

```
mail.techassi.dev
```

MX (15)

Mail Exchange

```
1 mail.techassi.dev.
```

TXT (16)

Text

```
"v=spf1 mx -all"
```

AAAA (28)

Eine IPv6 Host Adresse

```
2a00:1450:4001:831::200e
```

RR Classes

RFC 1035 (3.2.4)

Klassen beschreiben verschiedene Netzwerke. Das bekannteste und heute genutzte Netz ist das Internet auf Basis von IP. DNS erlaubt somit die Bereitstellung von RRs für verschiedene Netze.

IN (1)

The Internet

CS (2)

The CSNET Obsolet

CH (3)

The CHAOS net

HS (4)

Hesiod [Dyer 87]

RRs in Go

`pkg/types/rr/rr.go`

```
type RR interface {
    // Get header of RR
    Header() *Header

    // Set header of RR
    SetHeader(Header)

    // Set resource record data
    SetData(...interface{}) error

    // String returns the representation of any RR as text
    String() string

    // Len returns the records RDLENGTH
    Len() uint16

    // Unpack unpacks the RDATA
    Unpack([]byte, int) (int, error)

    // Pack packs the RDATA
    Pack([]byte, int) (int, error)
}
```

```
type Header struct {
    // Name specifies an owner name
    Name string

    // Type specifies a RR type code
    Type uint16

    // Class specifies a RR class code
    Class uint16

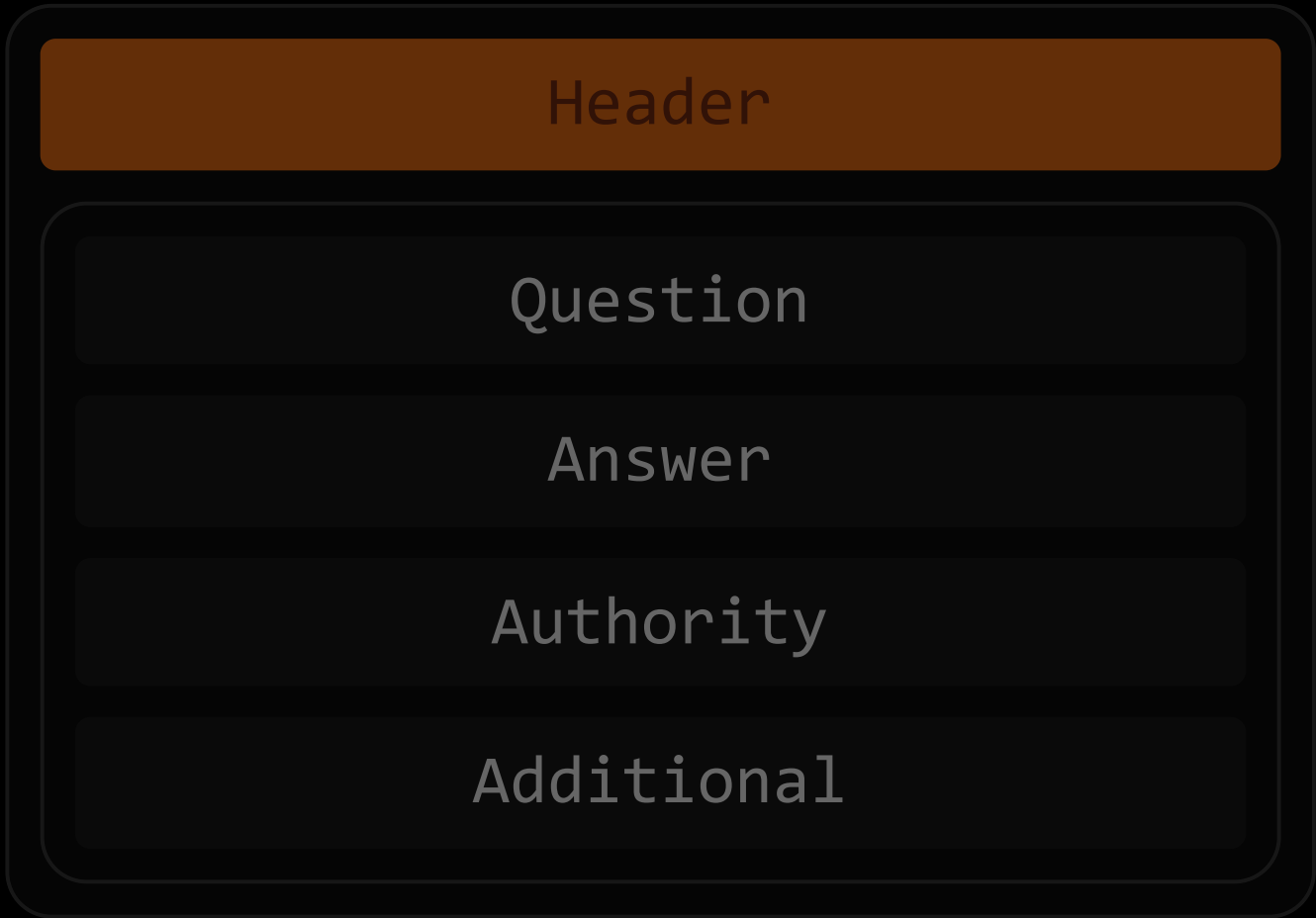
    // TTL specifies the time interval in seconds that the
    // RR may be cached before it should be considered
    // outdated
    TTL uint32

    // RDlength specifies the length of RDATA in octets
    RDlength uint16
}
```

Message Format

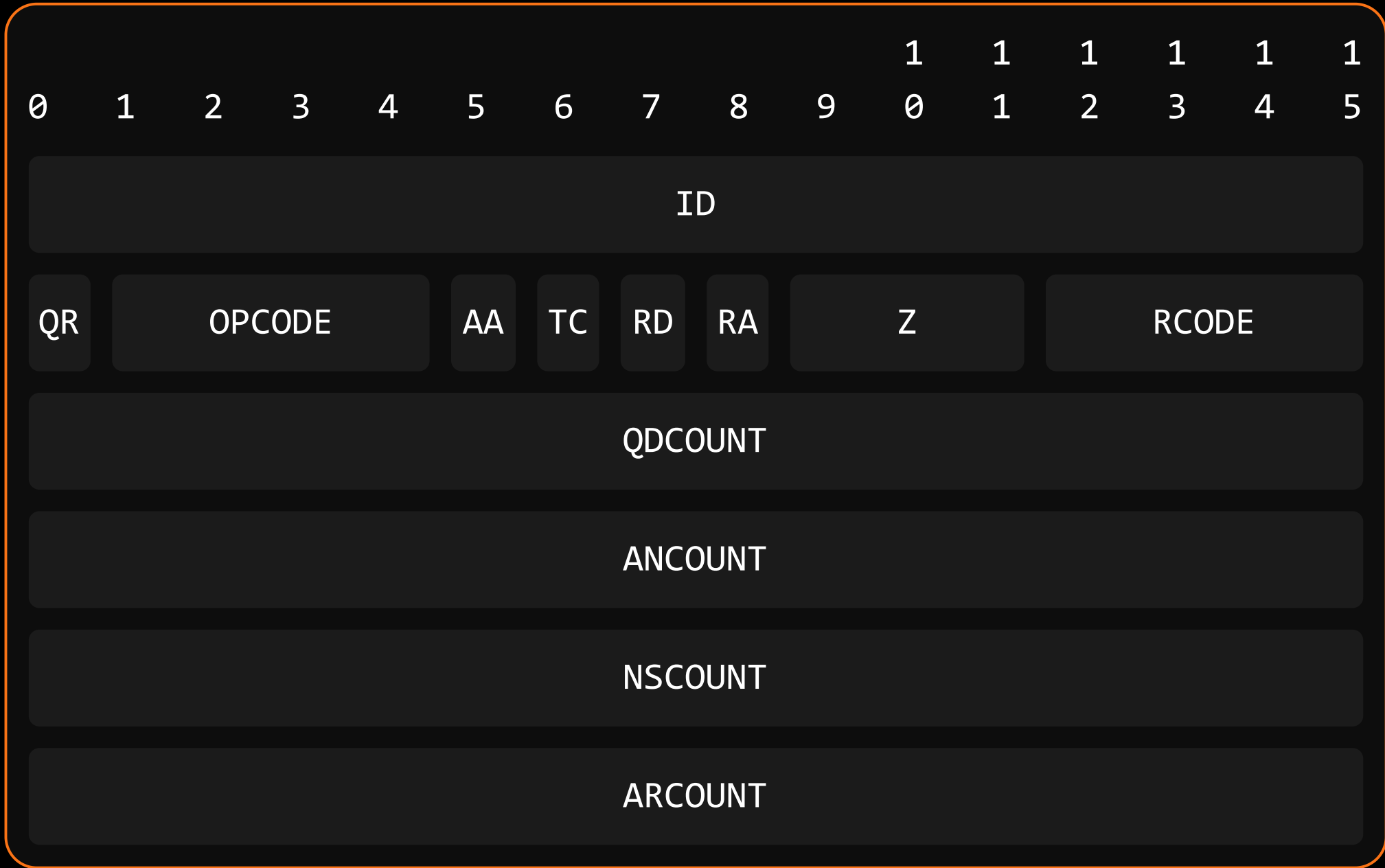
MESSAGE

Eine DNS Nachricht ist in 5 Abschnitte unterteilt.



HEADER

Der Header hat eine feste Größe von $6 \times 16 \text{ Bits} = 96 \text{ Bits}$ (12 Octets).



DNS Message in Go

`pkg/types/dns/message.go`

```
type Message struct {
    Header      Header
    Question    []Question
    Answer       []rr.RR
    Authority    []rr.RR
    Additional   []rr.RR

    Compression CompressionMap
}
```

```
type Code uint16

const (
    NoError Code = iota
    FormatError
    ServerFailure
    NameError
    NotImplemented
    Refused
)
```

```
type Header struct {
    ID                uint16    // ID
    IsQuery           bool       // QR
    OpCode            opcode.Code // OPCODE
    Authoritative     bool       // AA
    Truncated         bool       // TC
    RecursionDesired  bool       // RD
    RecursionAvailable bool       // RA
    Zero              bool       // Z
    RCode             rcode.Code // RCODE
    QDCount           uint16    // Question count
    ANCount           uint16    // Answer count
    NSCount           uint16    // Authority count
    ARCount           uint16    // Additional record count
}
```

```
type Code uint16

const (
    Query Code = iota
    IQuery
    Status
)
```

DNS Question in Go

[pkg/types/dns/question.go](#)

```
type Question struct {  
    Name  string  
    Type  uint16  
    Class uint16  
}
```

(Un)packer

pkg/packers/*

UNPACKER

PKG/PACKERS/UNPACKER.GO

Der Unpacker wandelt den Byte Nuffer von einkommenden DNS Nachrichten in Structs um

```
type Unpacker interface {
    Unpack(dns.Header, []byte, int) (dns.Message, error)

    UnpackHeader([]byte) (dns.Header, int, error)

    UnpackQuestion([]byte, int) (dns.Question, int)

    UnpackRRList(uint16, []byte, int) ([]rr.RR, int, error)

    UnpackRR([]byte, int) (rr.RR, int, error)

    UnpackRRHeader([]byte, int) (rr.Header, int)
}
```

PACKER

PKG/PACKERS/PACKER.GO

Der Packer wandelt Structs in einen Byte Buffer um, welcher zurück an Clients gesendet wird

```
type Packer interface {
    Pack(dns.Message) ([]byte, error)

    PackHeader(dns.Header, []byte, int) (int, error)

    PackQuestion(dns.Question, []byte, int) (int, error)

    PackRRList([]rr.RR, []byte, int) (int, error)

    PackRR(rr.RR, []byte, int) (int, error)

    PackRRHeader(*rr.Header, []byte, int) (int, error)
}
```

Eine Ebene tiefer (1)

pkg/pack/*

```
func UnpackUint8(data []byte, offset int) (uint8, int) {  
    return data[offset], offset + 1  
}
```

```
func UnpackIPv6Address(data []byte, offset int) (net.IP, int) {  
    hi, offset := UnpackUint64(data, offset)  
    lo, offset := UnpackUint64(data, offset)  
  
    ip := make(net.IP, net.IPv6len)  
    binary.BigEndian.PutUint64(ip, hi)  
    binary.BigEndian.PutUint64(ip[8:], lo)  
  
    return ip, offset  
}
```

Eine Ebene tiefer (2)

Kein valider Go Code

```
func UnpackDomainName(data []byte, offset int) (string, int) {
    if (data[offset] == 0x00) return "."; offset + 1

    for {
        b := int(data[offset]); offset++

        switch b & 0xC0 {
        case 0x00:
            if b == 0x00 {
                break
            }

            buf = append(buf, data[offset:offset+b]...)
            buf = append(buf, '.'); offset += b
        case 0xC0:
            if (!followed) offsetBeforePtr = offset + 1
            offset = int(b^0xC0)<<8 | int(data[offset])
            followed = true
        }
    }

    if (followed) offset = offsetBeforePtr
    return string(buf), offset
}
```