

2 TCP/IP-Grundlagen für das Tunneling

Um die in den nächsten Kapiteln fokussierten Tunneling-Techniken verstehen zu können, bedarf es zunächst einer Einführung in TCP/IP, da TCP/IP die Grundlage aller folgenden Kapitel darstellt. Dieses Kapitel bietet zudem erste Einblicke in das Tunneling auf den verschiedenen Schichten der Netzwerkkommunikation. Für den Fall, dass Sie sich bereits mit TCP/IP auskennen, können Sie dieses Kapitel überspringen oder einzelne Abschnitte im Bedarfsfall nachlesen.

2.1 Einleitung

Computer kommunizieren über Netzwerksysteme miteinander. Diese Systeme benutzen verschiedenste Medien mit verschiedenen Eigenschaften (etwa Datenrate, Störanfälligkeit) zur Übertragung von Daten (etwa eine Funkverbindung oder eine Glasfaser-Verbindung). Daten werden dabei in Form von *Datenpaketen*, das sind Einheiten einer maximalen Größe und vordefinierten Form, übertragen. In diesen Paketen werden sowohl Steuerinformationen, als auch die eigentlichen Nutzdaten übertragen. Doch damit verschiedene Computer sich gegenseitig “verstehen”, müssen sie dieselbe “Sprache” sprechen, denn sonst wüsste Computer A nicht, wie die Daten, die Computer B über das Netz an ihn sendet, zu interpretieren sind. Geregelt werden die Kommunikationsabläufe und die Aufbauten der Datenpakete daher durch so genannte *Protokolle*.

TCP/IP stellt eine Suite solcher Protokolle dar und beinhaltet zahlreiche, zum Teil standardisierte Protokolle. Die Buchstaben TCP stehen dabei für das Transportprotokoll »Transmission Control Protocol«, die Buchstaben IP für »Internet Protocol«. Doch dazu später mehr.

TCP/IP	OSI
Application Layer	Application Layer Presentation Layer Session Layer
Transport Layer	Transport Layer
Internet Layer	Network Layer
Network (Access) Layer	Data Link Layer Physical Layer

Abbildung 2.1: Das TCP/IP- und das OSI-Modell

Diese Kommunikationsarchitektur ist dabei so aufgebaut, dass sie – ähnlich wie das später noch erläuterte OSI-Modell¹ der ISO² – aus verschiedenen Schichten besteht (Abbildung 2.1).

Die Systeme, die das OSI-Modell implementieren, werden als *Open Systems* bezeichnet, da sie anderen Systemen zur Kommunikation “offen” stehen – schließlich sind die verwendeten Protokolle bekannt [116].

Das gezeigte Modell verwendet, wie auch das gesamte Buch, die englischen Begriffe der einzelnen Protokoll-*Schichten* (Layer). Das hat ganz einfach den Grund, dass so auch weitere Bücher zum Thema TCP/IP einfacher verstanden werden, weil diese oft in englischer Sprache verfasst sind.

Jeder Layer (dt. Schicht) kommuniziert dabei mit seinem logischen Gegenüber. Der Internet-Layer eines Rechners A kommuniziert folglich nur mit dem Internet-Layer des Kommunikationspartners B. Die Layer nutzen dabei jeweils die Dienste der unter ihnen liegenden Schichten, sodass das Abstraktions-, aber auch das Fähigkeitsspektrum mit jeder Schicht ansteigt.

Der Vorteil dieses Ansatzes ist, dass verschiedene Layer auch unterschiedliche Aufgaben übernehmen und eine einzelne Schicht nicht die Verantwortung über die gesamte Kommunikation übernehmen muss.³

Für den Anwender sind diese Layer völlig transparent, er benötigt lediglich eine Endapplikation, um auf die ihm angebotenen Dienste eines Netzwerks zuzugreifen.

Wichtig ist hierbei das Konzept der Einkapselung, wie sie in Abbildung 2.2 dargestellt ist: Der Network Access Layer kapselt die Daten des Internet Layers in sich. Der Internet Layer kapselt wiederum die Daten des Transport Layers in sich, der die Daten vom Application Layer einkapselt, die wiederum die Nutzdaten (*Payload*) der jeweiligen Anwendung, also etwa Teile einer Bilddatei, enthalten.

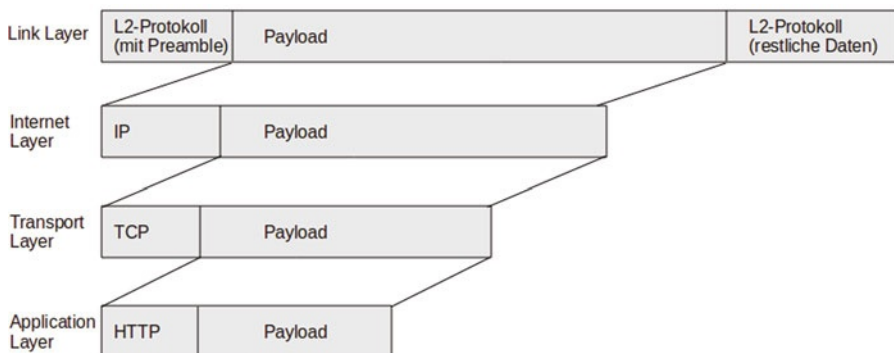


Abbildung 2.2: Einkapselung von Netzwerk-Layern in TCP/IP.

Betrachten wir nun die einzelnen Layer des TCP/IP-Modells beginnend mit der untersten.

¹Open Systems Interconnection Reference Model

²International Organization for Standardization

³Die erste TCP-Implementierung hatte genau diese problematische Eigenschaft. Mittlerweile ist die Arbeitsfunktion von TCP jedoch auf den Transport-Layer beschränkt worden.

2.1.1 Network-Access-Layer

Dieser Layer ist eine Zusammenfassung des *Physical*- und des *Data-Link*-Layers aus dem OSI-Modell. Er hat die Aufgabe, die einzelnen Bits über ein physikalisches Medium zum nächsten Netzwerkrechner (eines Netzwerkpfades) zu übertragen. Dies könnte zum Beispiel ein Crossover-Kabel sein, das an einer handelsüblichen Ethernet-Netzwerkkarte angeschlossen ist. Auf diesem Layer kommunizieren die Systeme mittels sogenannter *Frames*. Typische Medien sind neben Ethernet auch Wireless-LAN, Glasfaser-Verbindungen (FDDI) oder ISDN-Verbindungen. Entsprechend wird von Ethernet-Frames, FDDI-Frames usw. gesprochen.

2.1.2 Internet-Layer

Der Internet-Layer hat die Aufgabe, Daten mit Hilfe des Internet-Protokolls (IP), das wir später noch genauer betrachten werden, zu versenden und zu empfangen. Dazu besitzt jeder Rechner eine eindeutige Adresse – die sogenannte IP-Adresse – die ihn eindeutig in einem Netzwerk identifiziert. Anders als beim Network-Access-Layer erfolgt auf diesem Layer sogenanntes Routing.

Routing stellt sicher, dass Daten über verschiedene Netzwerke versendet werden können. Dazu werden Informationen benötigt, die angeben, über welche Router oder anderen Rechner man diese Netzwerke erreichen kann.

Die Informationen über die Wegfindung, also die Routinginformationen, werden dabei in den *Routingtabellen* der einzelnen Rechner abgelegt. Diese können entweder statisch vom Administrator konfiguriert oder dynamisch über Routingprotokolle verwaltet werden. Letztere Variante bietet je nach Routingprotokoll und verwendetem Routingalgorithmus diverse Angriffspunkte für Hacker, erleichtert aber das Management großer Netzwerke.

Die Routingtabelle eines Systems können Sie über das *route*-Programm aufrufen und administrieren. Unter Unix-Systemen kann zudem alternativ auch das *netstat*-Programm verwendet werden, um sich diese Informationen zu beschaffen.⁴

Kernel IP routing table						
Destination	Gateway	Genmask	Flags	Metric	Ref	Use Iface
191.255.255.1	*	255.255.255.255	UH	0	0	0 venet0
loopback	*	255.0.0.0	U	0	0	0 lo
default	191.255.255.1	0.0.0.0	UG	0	0	0 venet0

Abbildung 2.3: Die Ausgabe des Programms *route* unter Linux.

Wie Sie sehen, ist für jedes Ziel (Destination) ein Router (Gateway) angegeben (oder “*” für den Fall, dass kein Router gesetzt ist), über den die entsprechenden Netzwerkdaten versendet werden sollen. Dieses Gateway hat in seiner Routingtabelle dann wiederum entsprechende Daten, um das Paket an das nächste Gateway in Richtung seines Zieles zu senden. Auf diese Weise werden die *Pakete*, so werden Übertragungseinheiten auf diesem Layer bezeichnet, von einem zum nächsten Router weitergereicht, bis sie schließlich am Ziel ankommen. Neben dem

⁴Bei Verwendung des *netstat*-Programms benötigen Sie für die Routingtabelle die Parameter *-n* (löst keine Hostnames auf, sondern gibt IP-Adressen aus), *-r* (Ausgabe der Routingtabelle).

Gateway werden noch weitere Informationen zum Pfad (etwa die Netzmaske oder die zu verwendende Netzwerk-Schnittstelle) angegeben, was in diesem einführenden Kapitel allerdings nicht von Bedeutung ist.

Vielleicht ist Ihnen bereits der Eintrag *default* aufgefallen. Dieser legt ein Gateway fest, dem die Datagramme gesendet werden sollen, für die in der lokalen Routingtabelle kein konkreter Routingeintrag festgelegt ist. Ohne dieses Gateway bräuchte jeder mit dem Internet verbundene Rechner eine Routingtabelle, die die Routingdaten des gesamten Internets beinhaltet!

In diesem Buch werden die Begriffe *Gateway* und *Router* zwar synonym verwendet, jedoch sind beide nicht exakt gleich definiert. Ein Router leitet Pakete auf dem *Internet-Layer* weiter, ein Gateway verwendet dazu den *Application-Layer*.

Auch ist offensichtlich, dass der Internet-Layer den darunter liegenden Network-Access-Layer nutzt. Nur über diesen können die Pakete über das Kabel versendet werden. Dass die Pakete dazu in ein Frame gepackt werden, ist auch eine logische Konsequenz. Beim Empfänger wiederum entpackt die Network-Access-Schicht den Frame und reicht das erhaltene Pakete unverändert an den Internet Layer weiter – wie ein Ei, das erst geschält werden muss. Der Unterschied zum Ei besteht nun im Besonderen darin, dass jede Schicht erst ihre »Schale« entfernen muss, bevor dem Benutzer die relevanten Daten zur Verfügung stehen. So betrachtet können Sie sich das zuvor erläuterte Prinzip der Einkapselung auch als Zwiebelschalenmodell vorstellen, deren äußerste Schale das Link Layer-Protokoll, und deren innerste Schale das Application Layer-Protokoll darstellt.

2.1.3 Transport-Layer

Die übergeordnete Schicht, der Transport-Layer, hat die Aufgabe, die durch den Internet Layer zum Ziel beförderten Daten an die richtigen Anwendungen auszuliefern. Anwendungen werden dabei sogenannten *Ports* zugeordnet. Somit hat der Transport-Layer die Hauptaufgabe des Multiplexing inne, da er Vermittler zwischen Internet Protokoll und einer Vielzahl an möglichen Anwendungen spielt.

Dieser Layer stellt neben einzelnen Spezialprotokollen (wie etwa dem Routingprotokoll OSPF⁵) im Prinzip nur zwei herausragende Protokolle, nämlich TCP (*Transmission Control Protocol*) und UDP (*User Datagram Protocol*) bereit.

Während TCP eine etwas größere Datenlast pro Paket als UDP verursacht (da es einen größeren Header verwendet), verfügt es über ausgefeiltere Fehlerkorrekturmechanismen, die die Auslieferung der *Segmente* gewährleisten soll. UDP hingegen kümmert sich nicht darum, ob die über dieses Protokoll transferierten *Datagramme* überhaupt ankommen. UDP wird daher für Systeme eingesetzt, die hohes Datenaufkommen bei ständig wechselnden, aber *prinzipiell gleichen* Daten verursachen.⁶ Diese beiden Protokolle werden selbstverständlich noch detaillierter besprochen.

Ein weiterer Unterschied zwischen TCP und UDP ist der, dass TCP *verbindungsorientiert* arbeitet. Das bedeutet, dass eine Verbindung vor der Kommunikation zunächst aufgebaut und am Ende wieder geschlossen werden muss.

⁵Open Shorted Path First

⁶Dies kann beispielsweise bei Statusdaten, die einmal pro Sekunde komplett übertragen werden, der Fall sein.

2.1.4 Application-Layer

Der sogenannte Application-Layer, zu Deutsch die *Anwendungsschicht*, wird von den einzelnen Netzwerkprogrammen und -diensten verwendet. Die Anwendungen (*Applications*) stellen einen Dienst auf einem *Port* zur Verfügung (beziehungsweise greifen auf diesen clientseitig zu) und können über ihn senden und empfangen. Solche Applikationen benötigen jeweils ihre eigenen, in der Regel glücklicherweise standardkonformen Protokolle. Die wichtigsten davon sind das *Hypertext Transfer Protocol* (HTTP) für die Kommunikation zwischen Webserver und Browser, das *File Transfer Protocol* (FTP) für simple Dateiübertragungen, sowie das *Simple Mail Transfer Protocol* (SMTP) zum Versenden/Ausliefern von E-Mail bzw. das *Post Office Protocol 3* und das *Internet Message Access Protocol* (IMAP) zum Abrufen von E-Mails.

Auf dem Application-Layer spricht man meistens nicht mehr von Paketen bzw. Segmenten, sondern von *Messages* (Nachrichten) (UDP-basierte Kommunikation) und *Streams* (TCP-basierte Kommunikation).

2.1.5 Zusammenfassung

Fassen wir also noch einmal zusammen: Ein Programm, das mit einem anderen Programm auf einem entfernten Rechner kommuniziert, benötigt ein Protokoll. Das Protokoll wird benötigt, damit die Kommunikation fehlerfrei funktionieren kann – und eigentlich ist ohne ein wie auch immer geartetes Protokoll auch keine Art von Kommunikation möglich. Schließlich können sich auch nicht zwei Menschen unterhalten, die keine gemeinsame (ggf. non-verbale) Sprache sprechen.

Das Programm baut dazu eine Verbindung mit dem anderen Rechner auf, indem es auf einen Port zugreift, der mit dem entsprechenden Programm verknüpft ist. Diese Verbindung ist nun nichts anderes als ein TCP-Stream (oder beispielsweise eine UDP-Verbindung), der die Integrität der Verbindung sicherstellt. Dieser TCP-Stream nutzt wiederum das IP-Protokoll und damit den Internet-Layer, um sicherzustellen, dass die Pakete auch ihr Ziel finden. Erst der Network-Access-Layer bringt die zwiebelförmig verpackten Daten auf das Kabel.

2.2 Einige Worte zum OSI-Modell

Das OSI-Referenzmodell hat zwar direkt nichts mit der TCP/IP-Suite zu tun, wird aber hin und wieder erwähnt und stellt praktisch eine andere Möglichkeit der Schichtenunterteilung dar, als es beim TCP/IP-Modell der Fall ist.

Die Schichten des OSI-Modells sind also nur eine andere Betrachtungsweise auf denselben Gegenstand. Genau wie bei den TCP/IP-Layern wird also nur eine logische Sicht auf real arbeitende Protokolle abgebildet. Die Protokolle sind dabei das eigentlich Interessante, und die wichtigste Netzwerkprotokollsuite TCP/IP wird in diesem Kapitel behandelt. Die Erläuterung der Schichten soll also nur helfen, die Arbeit der Protokolle zu verstehen.

Der *Physical-Layer* (oft auch »Bitübertragungsschicht« genannt) ist für die Zustellung einzelner Bits zuständig. Dabei übernimmt dieser Layer auch die Aufrechterhaltung einer Bitstrom-Verbindung. Letzteres ist ebenfalls die Aufgabe des *Datalink-Layers* (auch als »Sicherungsschicht« bekannt). Zudem wird auf dem Datalink-Layer auch Flusskontrolle durchgeführt.

Der *Network-Layer* – in diesem ist unter anderem das Internet Protocol (IP) angesiedelt – ist für die Zustellung von Daten zwischen beiden Verbindungsendpunkten zuständig. Der nächsthöhere Layer, der *Transport-Layer*, ist für die Zustellung der Daten auf eine Art und Weise zuständig, die es für die höheren Layer transparent macht, Daten zu senden und zu empfangen. Auch laufen auf dem Transport-Layer Funktionen zur Fehlerkorrektur ab.

Im *Session-Layer* werden der Aufbau und Abbau von Sitzungen (etwa bei SSL) sowie deren Aufrechterhaltung abgewickelt. Außerdem kümmert sich diese Schicht etwa um das Festlegen von Synchronisationspunkten und die Administration des Senderechtes.

Die Unterteilung von *Application-* und *Presentation-Layer* hat zur Folge, dass der Application-Layer zwar ähnliche Aufgaben, wie im TCP/IP-Modell übernimmt, jedoch Aufgaben, die die Datendarstellung betreffen, in den Presentation-Layer ausgegliedert werden (er kümmert sich etwa um die Festlegung eines Zeichensatzes und um die Darstellung von Floating-Point-Werten).

2.3 Die wichtigen Protokolle

Im Folgenden werden die wichtigsten Protokolle, beginnend mit denen des niedrigsten Layers, besprochen. Wir beginnen jedoch nicht mit der Erläuterung von Ethernet- oder PPP-Frames (diese würden für eine kurze Einleitung in TCP/IP, wie diese hier, etwas zu weit gehen, oder werden als Tunneling-Protokolle im nächsten Kapitel besprochen), sondern mit ARP und wenden uns anschließend IPv4, ICMPv4, IPv6 und ICMPv6 zu. Danach betrachten wir TCP und UDP als Transport-Layer-Protokolle. Zum Abschluss des Kapitels werden wir die wichtigsten Application-Layer-Protokolle betrachten, die im Kontext des Tunnelings und der verdeckten Kanäle von Bedeutung sind.

2.4 Link Layer: ARP

In Ethernet-Netzwerken kommunizieren einzelne Hosts über die oben erwähnten Frames miteinander. Diese Frames beinhalten eine Ziel- und eine Absenderadresse. Diese Adresse nennt sich MAC-(Media Access Control-)Adresse (oder auch »Hardwareadresse«) und ist eine 48 Bit lange, weltweit eindeutige Adresse. Diese Adressen sind in die Netzwerkkarten eingebrannt, aber einige Betriebssysteme ermöglichen es auch, diese Adressen manuell zu setzen.

Doch woher weiß ein Rechner, welche MAC-Adresse ein anderer Host hat? An dieser Stelle kommt ARP, das *Address Resolution Protocol*, zum Einsatz. ARP hat die Aufgabe, die für eine IP-Adresse zugehörige MAC-Adresse herauszubekommen. Jeder Rechner, der eine ARP-Anfrage (den sogenannten *Request*) startet, behält die darauf enthaltene Antwort – also die Zuordnung von IP- zu MAC-Adresse – in seinem lokalen Speicher, dem *ARP-Cache*. Der ARP-Cache kann sowohl unter Windows-Systemen als auch unter Unix mit dem *arp*-Kommando abgefragt werden.

Nehmen wir einmal an, dass der Rechner *eygo.sun* mit der IP-Adresse *192.168.0.1* einen *ping* an den Host *yorick.sun* mit der IP-Adresse *192.168.0.5* starten möchte.

Nehmen wir zusätzlich an, dass diese Hosts bisher keinen Kontakt miteinander pflegten oder der letzte Kontakt schon so lange zurückliegt, dass die gegenseitigen ARP-Einträge bereits ge-

löscht wurden. Tatsächlich werden die dynamischen ARP-Einträge nach einer gewissen Zeit aus dem ARP-Cache gelöscht.

eygo.sun muss zunächst herausbekommen, welche MAC-Adresse der Host *192.168.0.5* besitzt. Er sendet eine Ethernet-Broadcast-Nachricht mit ARP, das heißt eine Nachricht an alle Hosts im Netz. Diese Nachricht beantwortet aber jeweils nur derjenige, der eine Antwort auf diesen Request geben kann.

Er bekommt schließlich die Antwort, dass der Host *192.168.0.5* die MAC-Adresse *0:0:cb:59:fd:be* hat. Diese Antwort nennt sich ein »ARP Reply« und wird nicht mehr an die Broadcast-Adresse, sondern direkt an den Host gesendet, von dem der ARP-Request ausging.

Betrachten wir dieses Vorgehen zunächst einmal mit dem Sniffer *tcpdump* unter BSD:

```
root@eygo.sun# tcpdump -i ne3 -vvv
...
11:59:27.681214 arp who-has 192.168.0.5 tell eygo.sun
11:59:27.681442 arp reply 192.168.0.5 is-at 0:0:cb:59:fd:be
...
```

Betrachten wir nun den ARP-Cache beider Hosts mit dem *arp*-Programm:

```
root@eygo.sun# arp -a
? (192.168.0.2) at 00:60:97:30:0c:bb on ne3
? (192.168.0.5) at 0:0:cb:59:fd:be on ne3
```

```
C:\>arp -a
```

```
Schnittstelle: 192.168.0.5 on Interface 0x1000003
  Internetadresse   Physikal. Adresse   Typ
192.168.0.1        00-50-bf-11-35-a5   dynamisch
192.168.0.2        00-60-97-30-0c-bb   dynamisch
```

2.4.1 Reverse-ARP

Die Gegenfunktion zum ARP bietet das *Reverse Address Resolution Protocol* (RARP). Im Gegensatz zum ARP löst es nicht IP- in MAC-Adressen, sondern MAC- in IP-Adressen auf. RARP benötigt einen zugehörigen Server, der die Anfragen der – meist plattenlosen – Clients auflöst. Wie Sie wohl schon errahnen, ist die Hauptaufgabe von RARP die Zuweisung von IP-Adressen an Thin-Clients.

Allerdings ist zu sagen, dass RARP von BOOTP und DHCP praktisch komplett verdrängt wurde, da diese Protokolle einen besseren Funktionsumfang bieten.

2.4.2 Proxy-ARP

Mittels Proxy-ARP kann ein Subnetz gespalten bzw. zusammengefügt werden. Zwischen beiden Netzen fungiert eine Proxy-ARP-Einheit sozusagen als Router für ARP-Anfragen. Zudem leitet

Bit 0		Bit 8		Bit 16		Bit 24	
Version	IHL	Type of Service (ToS)		Total Length			
Identification				Flags	Fragment Offset		
TTL		Protocol		Checksum			
Source Address							
Destination Address							
Options (Padding)							

Abbildung 2.4: Aufbau des IPv4-Headers (jede Zeile entspricht 32 Bit).

es die Pakete zwischen beiden Netzteilen hin und her. Proxy-ARP sollte jedoch nicht verwendet werden, da es sehr leicht ist, Spoofing-Angriffe gegen dieses Protokoll durchzuführen.

2.5 Internet Layer: IPv4

Auf dem Internet Layer wird – wie bereits erwähnt – paketorientiert verarbeitet: Die *Frames* des Link Layers mit ihren Signalisierungsmechanismen werden abstrahiert und die im Link Layer eingekapselten Dateneinheiten als *Pakete* bezeichnet. Die Aufgabe des Internet Layers besteht darin, Pakete von ihrer Quelle zum Ziel zu übertragen und dabei einen Weg über zwischengeschaltete Systeme (*Hops*) zu finden. Die Wegfindung ist die Aufgabe von *Routern*. Eine weitere Aufgabe des Internet Layers ist die Adressierung und die Fehlerbehandlung. Wir werden im Folgenden die wichtigsten Protokolle dieser Schicht betrachten (das sind IPv4 und IPv6, sowie ihre zugehörigen Protokolle) und dabei mit IPv4 beginnen.

Das Internet Protocol Version 4 (IPv4) ist das derzeit noch immer wichtigste Netzwerkprotokoll des Internet Layers. Jedes IPv4-Paket beinhaltet einen mindestens 20 Byte großen Header, der in Abbildung 2.4 dargestellt ist.

Betrachten wir die Bestandteile von IPv4 der Reihenfolge nach:

- Version: Die IPv4-Protokollversion (valide ist nur Version 4 (bzw. Version 6 für IPv6); Version 5 war ein experimentelles Protokoll).
- Internet Headerlänge (*Internet Header Length, IHL*): Dieses Feld gibt die Anzahl der 32-Bit-Wörter des Headers an, entsprechend wird die Größe der eingekapselten Daten höherer Layer nicht mit eingerechnet. Im Normalfall beinhaltet der IPv4-Header 5*32 Bit an Daten. Da der IPv4-Header um Optionen erweitert werden kann, ist es möglich, in diesem Feld größere Werte als 5 zu erhalten.

- ToS, Teil 1 (DSCP): Das Feld *Differentiated Services Code Point* (DSCP) implementiert Quality of Service (QoS), dabei handelt es sich um Qualitätsanforderungen für Pakete. In [37] lässt sich lesen: *Diffserv uses six bits of the IPV4 or IPV6 header to convey the Diffserv Codepoint (DSCP), which selects a PHB*. PHB steht für *Per Hop Behavior* und signalisiert, wie ein Paket behandelt wird, wenn es einen Router passiert. Der Standardwert für die PHB ist 000000 und bedeutet, dass (nach Möglichkeit) ein schnelles Weiterleiten des Pakets erfolgen soll. Die restlichen Werte (»Codepoint-IDs«) werden von der *Internet Assigned Numbers Authority* (IANA) zugewiesen [38]. Typische Anforderungen für DSCP sind ein schneller Verbindungsaufbau, zuverlässige Datenübertragung mit wenig Störungen und hohe Verbindungsstabilität. Tanenbaum nennt verschiedenste Beispiele für die anwendungsabhängige Wahl dieser Anforderungen [131], etwa benötigt eine Videokonferenz eine geringe Zuverlässigkeit, weil es nicht schlimm ist, wenn einzelne Bild-Frames verloren gehen, stattdessen gibt es hierbei hohe Anforderungen hinsichtlich der Verzögerung einer Verbindung und ihrer Bandbreite. Bei einer Dateiübertragungen gibt es auf der anderen Seite eine hohe Anforderung an die Zuverlässigkeit einer Verbindung, doch ist die Anforderung an die Verzögerung einer Verbindung relativ gering.
- ToS, Teil 2 (ECN, *Explicit Congestion Notification*): Die ECN-Bits werden nur benutzt, falls Empfänger und Sender sich auf die Nutzung derselben einigen. Anstatt Routerüberlastung nur durch Paketverlust zu detektieren, kann mit ECN eine drohende Überlastung signalisiert werden [38].
- Total Length: Im Gegensatz zur zuvor besprochenen »Internet Header Length« (IHL) repräsentiert dieses Feld die gesamte Größe des Datenpakets (IPv4-Header samt Payload) in Bytes. Valide Werte liegen im Bereich von 20 bis 0x65535 Bytes.
- Identification (ID): Dieses Feld dient als Identifikationswert für ein Paket, es wird zur Wiederausammensetzung nach einer Fragmentierung verwendet (siehe »Flags«).
- Flags: (3 Bits): Bit 0 hat immer den Wert 0 (es ist für die zukünftige Verwendung reserviert). Bit 1 verbietet das Fragmentieren (also Aufteilen des Pakets in mehrere kleine Pakete) und nennt sich *Don't Fragment-Flag* (DF). Bit 2, das *More Fragments-Flag* (MF), signalisiert hingegen, dass dieses Paket ein Teilpaket eines ehemals größeren, aber nun fragmentierten Pakets ist und weitere Fragmente folgen. Die weiteren Fragmente werden benötigt, um die finale Wiederausammensetzung eines fragmentierten Pakets zu ermöglichen.
- Fragment-Offset: Dieses Feld zeigt an, an welcher Stelle im Empfangspuffer die im Paket enthaltenen Daten eines Fragments platziert werden müssen, also welcher Teil eines fragmentierten Pakets vorliegt (Angabe in Bytes). Es gibt 2^{13} mögliche Fragment-Offset-Werte (das heißt: maximal 8192 Fragmente pro Datenpaket). Jedes Fragment muss ein Vielfaches von 8 Bytes ausmachen. Damit gilt: $2^{13} * 8 = 65535$ Bytes. Abzüglich der Header-Länge sind somit mehr Payload-Daten adressierbar, als ein unfragmentiertes Paket beinhalten kann.

- **TTL (*Time to Live*, 8 Bit):** Dieser Wert wird von jedem Router, den ein Paket passiert, um 1 verringert (dekrementiert). Erreicht die TTL den Wert 0, wird das Paket verworfen. Bei den meisten Implementierungen wird für ein neues Paket der Wert 255 vergeben. Auf diese Weise können Routing-Loops verhindert werden, bei denen Pakete zwischen Routern unendlich lang im Kreis verschickt werden.
- **Protocol:** Dieses Feld ist für uns – wie wir später sehen werden – von besonderer Bedeutung, da hiermit das IPv4-Tunneling ermöglicht wird. Das Protocol-Feld spezifiziert das eingekapselte Netzwerkprotokoll und die Werte des Feldes sind standardisiert. Die häufigsten Werte sind: 1 (ICMP), 2 (IGMP), 6 (TCP) und 17 (UDP).
- **Checksum:** Prüfsumme über den Header zur Fehlererkennung. Dieses Feld wird von jedem passierten Hop neu berechnet, da sich jeweils die TTL verändert.
- **Source IP Address und Destination IP Address (Quell- und Zieladresse des Datenpakets):** Hierbei handelt es sich um einen jeweils 32 Bit großen Adressierungswert.
- **Options:** Dieses Feld wird nur in Spezialfällen verwendet (die zuvor besprochene IHL muss entsprechend erhöht werden, da sich durch Hinzufügen von Options-Bereichen die Headergröße erhöht). Die Options waren ursprünglich zur Weiterentwicklung von IP gedacht und in einem einzelnen IPv4-Paket können mehrere Options hintereinander untergebracht werden. Das Ende der Options muss mit einem EOL-Wert (*End of Options List*, Wert 0) signalisiert werden.

Hinter dem IPv4-Header folgen die eingekapselten Daten (aus Sicht vom Internet Protocol handelt es sich dabei um Payload, tatsächlich jedoch um Transport Layer-Daten). Entsprechend folgt in einem Netzwerkpaket direkt hinter dem letzten Byte des IP-Headers beispielsweise ein UDP- oder ein TCP-Header.

2.5.1 IP-Adressen

Die bereits erwähnten 32-Bit-IP-Adressen dienen zur eindeutigen Adressierung von Netzwerkrechnern in einem IP-Netzwerk (etwa einem lokalen Netzwerk oder dem Internet).⁷ Zwar lässt sich einiges über IP-Adressen und ihre Vergabe⁸ schreiben, doch sind diese Themen für dieses Buch nur von begrenzter Relevanz.

2.5.2 Fragmentierung

Für Netzwerkverbindungen existieren Limits hinsichtlich der maximale Größe an Daten, die durch eine Link Layer-Frame übertragen werden können, die so genannte *Maximum Transmission Unit* (MTU). Für Ethernet-Verbindungen beträgt die MTU meist 1500 Bytes. Verschickt

⁷Es gibt Techniken, wie NAT, die eine eindeutige Identifizierung von Netzwerkrechnern für Systeme außerhalb eines internen Netzes verhindern.

⁸Die Vergabe der IP-Adressen wurde ursprünglich durch die IANA (Internet Assigned Numbers Authority) geleitet. Später vergab die IANA nur noch Adressen an Regional Internet Registry-Organisationen (RIRs), etwa APNIC (Asian-Pacific Network Information Center) oder RIPE (Réseaux IP Européens Network Coordination Centre).

nun ein Rechner über eine Route, die mehrere Netzwerkverbindungen passieren muss, ein recht großes Paket (sagen wir, es sei 1600 Byte groß und hat die ID 1234), und muss dieses Paket dabei über eine Netzwerkverbindung mit einer kleineren MTU übertragen werden, muss das Paket aufgeteilt (fragmentiert) werden.

Hierzu kommen die oben bereits erwähnten IP-Felder *Identification*, *Flags* und *Fragment-Offset* zum Einsatz. Unser 1600 Byte großes Paket soll nun zur Verdeutlichung des Verfahrens über eine Verbindung mit einer MTU von 1492 Bytes übertragen werden. Zunächst wird der erste Teil des Pakets (die 20 Byte des IP-Headers samt den ersten 1472 Bytes des Payloads) über die Verbindung übertragen und das *More Fragments*-Flag gesetzt. Das Abschneiden nach den ersten 1472 Bytes ist notwendig, denn 1492 Bytes MTU - 20 Byte für den IP-Header ergeben den Maximalwert für den anhängbaren Payload (also 1472). 1472 Byte sind gleichzeitig ein Vielfaches von 8 Byte und somit ein valider Wert für das Fragment-Offset des folgenden Paketes (das Fragment-Offset des ersten Paketes ist 0).

Die restlichen Payload-Daten (1600 Byte - 20 Byte für den ursprünglichen Header - 1472 Byte an bereits übertragenem Payload = 108 Bytes) werden in ein zweites Paket gepackt, welches denselben Header, wie das erste Fragment erhält (auch der Identifier-Wert 1234 ist derselbe). Im zweiten Fragment wird das Fragment-Offset angepasst und erhält den Wert 184 (= 1472 Byte aus Fragment 1 : 8 Byte). Außerdem wird im zweiten Fragment das *More Fragments*-Flag nicht mehr gesetzt, da keine weiteren Fragmente mehr folgen. Nachdem das zweite Paket übertragen wurde, kann der Empfänger es verarbeiten. Die Wiederausammensetzung eines Pakets erfolgt ausschließlich beim Empfänger und wird von keinem zwischengeschaltetem Router vorgenommen, da einzelne Fragmente in IP-Netzen unterschiedliche Pfade passieren können; es ist also nicht sichergestellt, dass ein Router jedes Fragment eines Paketes erhält. Basierend auf dem Wert des Fragment-Offsets kann der Empfänger selbst dann die Payload-Bestandteile der einzelnen Fragmente in der richtigen Reihenfolge zusammensetzen, wenn diese in einer Reihenfolge ankamen, in der sie gar nicht verschickt wurden.⁹

2.6 Internet Layer: ICMPv4

ICMP (Internet Control Message Protocol) dient, wie sein Name schon verrät, der Kontrolle des IP-basierten Datenverkehrs. Es dient genauer gesagt der Fehleranalyse, der Informationserstattung und der Konfiguration. Der Standardaufbau des ICMPv4-Headers ist simpel und auf der Abbildung 2.5 zu sehen.

Der ICMP-Type spezifiziert die gewünschte Hauptfunktion eines ICMP-Paketes, die durch den ICMP-Code genauer spezifiziert werden kann. Die Prüfsumme hat dieselbe Funktion wie im IP-Header. Nach den ersten 32 Byte des ICMP-Headers folgen typspezifische Inhalte.

Die IANA (*Internet Assigned Numbers Authority*) hat eine offizielle Liste der ICMP-Types und zugehöriger ICMP-Codes veröffentlicht. Die wichtigsten dieser ICMP-Typen sollen im Folgenden besprochen werden.

⁹Wie bereits erwähnt, können einzelne Pakete (somit also auch Fragmente, die an sich wieder Pakete sind) unterschiedliche Pfade zum Ziel nehmen. Ein Weg über Pfad A kann länger benötigen, als ein Weg über Pfad B oder C, womit Paketreihenfolgen vertauscht werden können.

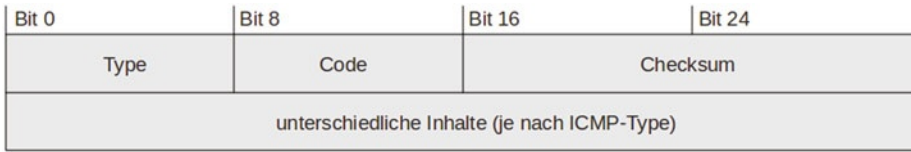


Abbildung 2.5: Aufbau des ICMPv4-Headers.

2.6.1 ICMP-Type 0 und 8

Die wohl bekanntesten ICMP-Typen sind Type 0 und 8. Ersterer ist der sogenannte »Echo Reply«, Letzterer die »Echo Request«. Diese beiden ICMP-Typen werden in der Regel zur Feststellung der Erreichbarkeit eines Hosts verwendet. Ferner wurden in vergangenen Jahren Denial-of-Service-Angriffe mit diesen Typen durchgeführt, die heutzutage jedoch nicht mehr relevant sind.¹⁰

Eine ICMP-Echo-Nachricht besteht aus einem 16-Bit-Identifer sowie einer 16-Bit-Sequenznummer zur Identifikation der Datagramme. Optional können noch Daten an das Paket angehängt werden, um die Netzlast zu erhöhen. Dies macht besonders bei Performance-Tests Sinn, denn die Datagramme müssen inklusive der Payload bestätigt werden.

Ein Ping funktioniert folgendermaßen: Host A sendet an Host B (oder broadcastet an ein Netzwerk) einen Echo-Request, Host B antwortet darauf mit einem Echo-Reply.

2.6.2 ICMP-Type 3

Der ICMP-Type 3 (Destination unreachable) wird immer dann versendet, wenn ein Datagramm nicht zugestellt werden konnte. Genauere Information zum Grund der Fehlzustellung gibt der ICMP-Code an. Es gibt folgende Möglichkeiten:

- Code 0 (*net unreachable*): Das Zielnetzwerk ist nicht erreichbar. Dies kann zum Beispiel auf fehlende Einträge in der Routingtabelle zurückzuführen sein.
- Code 1 (*host unreachable*): In diesem Fall ist zwar das Netzwerk, jedoch nicht der Host erreichbar. Entweder wurde die falsche Adresse angegeben oder der Host hat technische Probleme bzw. ist ausgeschaltet.
- Code 2 (*protocol unreachable*): Bekommt man diese Fehlermeldung, ist das Protokoll, an das das Datagramm weitergereicht werden soll, nicht erreichbar.
- Code 3 (*port unreachable*): Diese Meldung wird ausgesandt, wenn das Datagramm – im Transport-Layer also ein TCP-Streampaket oder eine UDP-Message – nicht an den entsprechenden Port weitergereicht werden konnte. Portscanner benötigen, je nach Scan-Technik, *port unreachable*-Nachrichten, um festzustellen, ob ein bestimmter Port erreichbar ist.

¹⁰Im Speziellen ist hier der sogenannte Ping-of-Death gemeint, der einige Hosts zum Absturz bringen konnte.

- Code 4 (*fragmentation needed and DF set*): Diese Meldung wird ausgesandt, wenn ein Datagramm mit dem »Don't-Fragment-Bit« versandt wurde, aber zu groß für die MTU eines Netzes ist. Dieser Code wird in Verbindung mit der Path-MTU-Discovery und der Fragmentierung verwendet. Wir gingen bereits auf die Fragmentierung von IP-Datagrammen ein.
- Code 5 (*source route failed*): Hat man ein IP-Datagramm mit einer Loose- oder Strict-Source-Route-Option versandt, wird diese Meldung im Falle einer nicht gangbaren Route gesandt.
- Code 6 (*destination network unknown*): Ist ein Netzwerk gar nicht vorhanden, wird diese Nachricht versandt.
- Code 7 (*destination host unknown*): Ist ein einzelner Host gar nicht vorhanden, wird diese Nachricht versandt.
- Code 8 (*source host isolated*): Dieser Typ findet normalerweise gar keine Verwendung. Allerdings soll das Militär hiermit einige Spielereien betreiben.
- Code 9/10 (*communication with destination network/host administratively prohibited*): Die Kommunikation zum Netzwerk bzw. Host wurde vom Administrator untersagt.
- Code 11/12 (*network/host unreachable for type of service*): Diese Meldung wird versandt, wenn das Zielnetzwerk bzw. der Zielhost nicht für den angegebenen Type-of-Service (TOS), der oben besprochen wurde, erreichbar ist.
- Code 13 (*communication administratively prohibited by filtering*): Die Kommunikation zum Zielsystem ist nicht möglich, da ein Firewall-System den Traffic blockt.
- Code 14 (*host precedence violation*): Wurde »precedence« als Type-of-Service-Wert angegeben, wird diese Meldung ausgegeben, wenn diese nicht vereinbar ist.
- Code 15 (*precedence cut-off in effect*): Die »precedence« wurde herabgesetzt, da dies vom Administrator so konfiguriert wurde.

Der Header dieser Nachricht ist nach dem statischen Header mit einem 32 Bit (1 Wort) langen und ungenutzten Bereich versehen. Anschließend folgt der IP-Header des ursprünglichen Datagramms plus 64 Bit der originalen Datagramm-Payload. Der Grund für diesen Anhang liegt in den besseren Diagnosemöglichkeiten, die sich aus den zusätzlichen Informationen ergeben.

2.6.3 ICMP-Type 4

ICMP-Type 4 (»Source Quench«) wird von einem Empfänger versandt, wenn zu viele Pakete bei ihm eintreffen. Der Sender dieses Types teilt dem Empfänger mit, dass jener seine Aussenderate drosseln soll.

Man nennt dieses Verfahren auch »Flusskontrolle«. Die Hauptaufgabe besteht darin, überlastete Prozessoren und Netzwerkschnittstellen zu verhindern.

2.6.4 ICMP-Type 5

ICMP-Type 5 (»Redirect«) wird zur Umlenkung von Routen verwendet. Wird ein Router verwendet, um ein Paket zuzustellen, und kennt dieser Router zusätzlich einen besseren Pfad zum Ziel als den, der über ihn selbst führt, so kann der Router zum Sende-Host eine ICMP-Redirect-Nachricht senden. Der sendende Host wird daraufhin die eigene Routingtabelle so abändern, dass er seine Pakete über den vom Router empfohlenen »besseren« Router sendet.

Via ICMP-Redirect können Datagramme zu einem Host (Code 1), zu einem Netzwerk (Code 0) oder Type-of-Service-basierend für einen Zielhost (Code 3) oder ein Ziernetzwerk (Code 2) umgeleitet werden.

2.6.5 ICMP-Type 9 und 10

ICMP-Type 9 (»Router advertisement«) wird zur Bekanntgabe von Routern verwendet, während Type 10 (»Router solicitation«) diese Meldung anfordert. So kann ein Host sich über die im Netzwerk befindlichen Router informieren und seine Routingtabelle selbst konfigurieren.

2.6.6 ICMP-Type 11

Eine ICMP-Type 11-(»Time exceeded for datagram«-)Nachricht wird versandt, wenn die TTL eines IP-Datagramms abgelaufen ist. Der Host, bei dem die TTL den Wert 0 erreicht, versendet diese Nachricht an den Sender des Datagramms.

Im Normalfall ist der ICMP-Code dann null. Kommt jedoch eines (oder mehrere) von mehreren Fragmenten nicht an, so wird der ICMP-Code auf den Wert eins gesetzt.

Wird zum Beispiel versucht, dem Host *www.google.de* ein Echo-Request-Datagramm zu senden, das mit einer TTL von 1 versehen ist, so wird dieses Datagramm (wenn man sich nicht gerade im entsprechenden Subnetz des google-Hosts befindet) sein Ziel nicht erreichen. Stattdessen wird der nächste Router, der die TTL auf den Wert »0« dekrementiert, ein ICMP-Type-11-Paket zu dem Host zurücksenden, der das ICMP-Echo-Paket sandte. Sehen wir uns die entsprechenden zwei Pakete einmal an:

```
19:02:52.030118 192.168.0.1 > 216.239.59.104: icmp:
echo request (id:5d56 seq:10) [ttl 1] (id 8922)
```

```
19:02:52.031033 192.168.0.2 > 192.168.0.1: icmp:
time exceeded in-transit for 192.168.0.1 >
216.239.59.104: icmp: echo request (id:5d56 seq:10)
[ttl 0] (id 8922) (DF) (ttl 255, id 11123)
```

Zu sehen ist hier, dass der Host 192.168.0.1 ein ICMP-Echo-Request an den Host 216.239.59.104 sendet. Dies jedoch kommt nur bis zum Router des lokalen Subnetzes (in diesem Fall ist dies der Router mit der IP-Adresse 192.168.0.2, ein Default-Gateway). Der Router 192.168.0.2 sendet nun die Meldung an den Host 192.168.0.1 zurück, dass die TTL des ICMP-Datagramms abgelaufen ist.

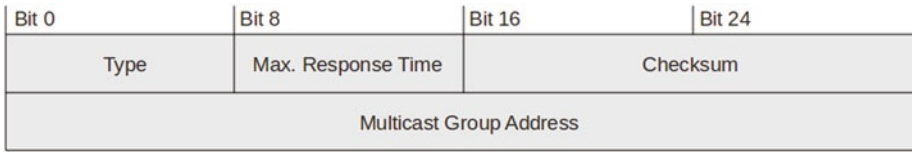


Abbildung 2.6: Aufbau des IGMP-Headers.

2.6.7 ICMP-Type 12

Ein »Parameter problem« (ICMP-Type 12) liegt vor, wenn im Optionsbereich des IP-Headers ein dem Host unbekannter Parameter gefunden wurde. Der Host verwirft dieses Datagramm und sendet ICMP-Type 12 zum Sender zurück.

2.6.8 Weitere ICMP-Typen

Es gibt noch einige weitere ICMP-Typen, doch sollten an dieser Stelle nur die wichtigsten von ihnen besprochen werden. Eine vollständige Liste aller ICMP-Parameter erhalten Sie bei der IANA unter <http://www.iana.org/assignments/icmp-parameters>.

2.7 Internet Layer: IGMP

Das Internet Group Management Protocol (Protokollnummer 2) wird zur Verwaltung von IPv4-Multicast-Gruppen verwendet. Die aktuelle Version IGMPv3 wurde 2002 durch RFC 3376 standardisiert. IGMP verwendet die Multicast-Adressen im Bereich von 224.0.0.1 bis 239.255.255.255 (Class D), um die Adressierung einer Multicast-Gruppe zu ermöglichen. Die Möglichkeit des Multicast-Sendens ist besonders bei Radio- und Videoübertragungen von Belang.

2.7.1 Der IGMPv2-Header

Der Einfachheit halber setzen wir uns zunächst mit IGMPv2 auseinander und besprechen dann anschließend die wichtigsten Neuerungen der Version 3. Der IGMPv2-Header ist recht simpel aufgebaut. Im Folgenden werden wir die einzelnen Felder des Headers besprechen.

Das Type-Feld kann einen der folgenden Werte enthalten:

- Membership Query (0x11)
- Version 1 Membership Report (0x12)
- Version 2 Membership Report (0x16)
- Version 2 Leave Group (0x17)

Ein Multicast-Router sendet Membership Querys zu den Hosts seines Netzwerks. Dies hat den Zweck, herauszufinden, welchen Multicast-Gruppen, welche lokalen Teilnehmer zugeordnet sind. Dabei unterscheidet man zwischen einer generellen Query, die an alle Gruppen gerichtet ist, und einer gruppenspezifischen Query, welches sich an eine einzelne Gruppe richtet. Dabei wird

die Zieladresse 224.0.0.1 verwendet. Ein Host sendet darauf als Bestätigung ein IGMP-Reply-Paket und verwendet dabei als Zieladresse die Adresse der Multicast-Gruppe, die er zudem auch im »Group Address«-Feld des IGMP-Headers angibt.

Ein Host kann an eine Gruppe senden, in der er selbst kein Mitglied ist, und kann gleichzeitig Mitglied in verschiedenen Gruppen sein. Die »Maximum Response Time« ist nur in Querys gesetzt und gibt die Zeitspanne (in Zehntelsekunden) an, die der Router auf einen Membership-Report wartet, bevor er den Host aus der Verteilerliste der Gruppe entfernt.

Das »Checksum«-Feld sollte Ihnen bereits von IP und ICMP bekannt sein. Das »Group Address«-Feld enthält die Adresse einer Multicast-Gruppe, an die ein IGMP-Paket versendet wird. In den oben erwähnten »generellen« Querys wird die Gruppenadresse mit Nullen überschrieben.

In IGMPv3 werden im Header zusätzlich die Senderquellen samt eines Adressvektors und Robustness-Werten übertragen, wodurch der Headeraufbau komplexer wird. Im Kontext dieses Buches sind diese Features allerdings nicht von Bedeutung.

2.8 Internet Layer: IPv6

Es gab verschiedene Gründe dafür, eine neue Version des Internet-Protokolls, nämlich IP-Version 6 zu entwickeln. Der wohl bekannteste Grund ist der, dass der Adressraum aus den über 4 Mrd. IPv4-Adressen knapp wurde. Zu Anfang ging man nämlich recht großzügig mit der Vergabe der Adressen um, und viele große Institutionen und Firmen in den Vereinigten Staaten bekamen gleich ganze Class-A-Netze zugeteilt. Europa kam bei der Verteilung der Adressen noch relativ gut weg, doch Asien und Afrika leiden unter Adressknappheit. Erste IPv6-Netze existieren seit 1996 (»6Bone«).

Eine Maßnahme gegen die stetig schwindende Anzahl von IPv4-Adressen ist die Network Address Translation. Doch auf Dauer wird dies nicht mehr ausreichend sein – besonders nicht, wenn immer mehr mobile Geräte wie Handys und diverse Hausgeräte (Kühlschränke etc.) miteinander vernetzt werden.

Ein weiterer wichtiger Grund für die Einführung von IPv6 ist die erweiterte Sicherheit. So ist IPSec Bestandteil von IPv6, und der Aufbau des Headers ist logischer und flexibler. Zudem unterstützt IPv6 eine Autokonfiguration der Netzwerkverbindungen ähnlich der des Dynamic Host Configuration Protocols (DHCP), Quality of Service (QoS), Mobile IP und Multicast.

In Abbildung 2.7 sehen Sie den IPv6-Header. Wie bei IPv4 werden die ersten 4 Bits zur Angabe der Versionsnummer verwendet.

Die nächsten 8 Bits stellen die »Traffic Class« dar. Die »Traffic Class« dient zur Prioritätsangabe, etwa wie beim DSCP-Feld des IPv4-Headers. Die folgenden 20 Bits (»Flow Label« genannt) dienen zur Identifikation eines Datenstroms – hiermit sollen auch Performancezuwächse in Routern möglich sein, da diese über einen Hash auf diese Zahl eventuelle Fragmente schneller zusammensetzen können.

Die Länge des Payloads wird in den nächsten 16 Bits angegeben. Hierbei ist zu beachten, dass die zusätzlichen IPv6-Header als Payload gelten. Der Payload berechnet sich also aus den IPv6-Erweiterungshheadern (dazu später mehr) plus den Headern der höheren Protokolle plus des Payloads des höchsten Protokolls.

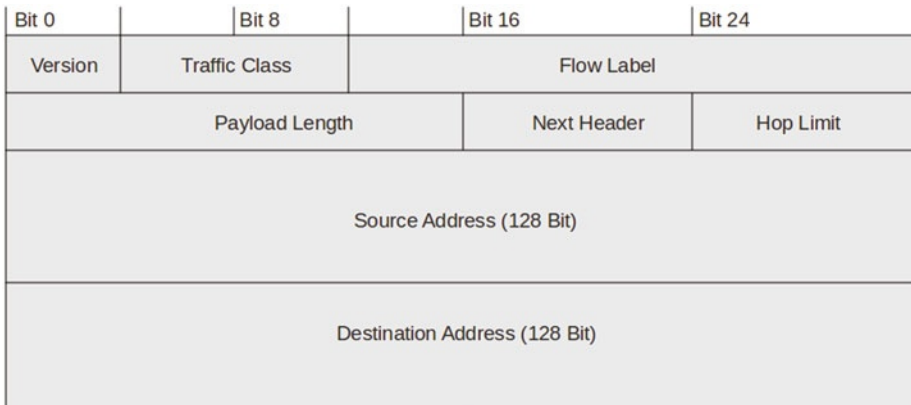


Abbildung 2.7: Aufbau des IPv6-Headers.

Das Feld »Next Header« gibt an, wie der nächste Header auszuwerten, besser gesagt, an welche »Instanz« das Paket weiterzugeben ist. Typische Werte sind 6 (TCP) oder 17 (UDP). Unter IPv4 heißt dieses Feld »Protocol«. Dieses Feld wird aber auch zur Angabe der IPv6-Extension Header verwendet.

Das sogenannte »Hop Limit« hat die gleiche Funktion wie das Feld »Time to Live« bei IP-Version 4. Es wird ebenfalls durch einen 8 Bit langen Wert (unsigned integer) repräsentiert, dessen Wert bei jedem Hop um 1 verringert wird. Erreicht das Feld den Wert 0, wird das Paket verworfen. Typische TTLs (wie auch bei IPv4) werden bei heutigen Betriebssystemen nicht mehr auf 0xff, sondern auf einen Wert ≥ 64 gesetzt. Routen über das Internet sind in den meisten Fällen nicht länger als 32 Hops [163].

Hinzu kommen natürlich noch die Adressierungsfelder: IPv6 hat eine jeweils 128 Bit lange (also im Vergleich zu IPv4 genau 4-mal so große) Adresse. IPv6-Adressen werden dabei in Hex-Form zu jeweils 16 Bit mit getrennten Doppelpunkten dargestellt.¹¹

Ein Beispiel für eine IPv6-Adresse wäre also das Folgende:

1234:1234:2342:2342:1234:1234:2342:2342

Folgen ein oder mehrere 16-Bit-Gruppen, die den Wert 0 haben, aufeinander, kann dies durch zwei Doppelpunkte dargestellt werden. Die Adresse

1234:1234:2342:0000:0000:0000:2342:2342

könnte also auch folgendermaßen dargestellt werden:

1234:1234:2342::2342:2342

Es ist immer nur eine einzige Doppelpunkt-Gruppe bei der Darstellung durch zwei Doppelpunkte pro Adressangabe erlaubt.

Nur damit Sie einen Eindruck davon bekommen, wie viele IPv6-Adressen es gibt: Es sind ganze 340 Sextillionen 282 Quintilliarden 366 Quintillionen 920 Quadrilliarden 938 Quadrillionen

¹¹Selbstverständlich sind auch andere Representationen von IPv4-Adressen möglich, so findet sich hin und wieder etwa die Hex-Darstellung. Die Adresse 127.0.0.1 kann entsprechend als 0x7f000001 dargestellt werden.

436 Trilliarden 463 Trillionen 374 Billiarden 607 Billionen 431 Milliarden 768 Millionen 211 Tausend 456 Stück.

2.8.1 IPv6-Extension-Header

Ein IPv6-Header besteht im Minimalfall nur aus dem Basis-Header. Diesen lernten Sie bereits oben kennen. Doch neben diesem Header gibt es noch die folgenden weiteren Header (wobei die Reihenfolge der tatsächlichen Reihenfolge im Header entspricht, diese kann nämlich nicht willkürlich gewählt werden):

- **Hop-by-Hop Options und Destination Options:** Hier können Optionen für jeden Host, den ein Paket passiert (»Hop-by-Hop«) oder für das Ziel des Pakets (»Destination Options«) festgelegt werden. RFC 2460 definiert zunächst nur 2 Bits (also 4 Kombinationen) der 8 Bits mit Funktionen, die insgesamt zur Verfügung stehen. Dabei kann eine Option entweder übergangen werden (0) oder das Paket (eventuell mit einer Bedingung) verworfen werden (1-3).
- **Routing:** Routing-Optionen ähnlich denen von IPv4; es kann eine Liste von Hops festgelegt werden, die das Paket passieren soll.
- **Fragment:** Dieser Header bietet eine ähnliche Funktionalität wie die Fragment-Felder des IPv4-Headers – auch dieser besteht aus dem Fragment-Offset, dem More-Fragments-Bit und der Identifikation des Pakets.
- **Authentication:** Der Authentication-Header (kurz AH) gewährleistet die Authentizität eines Datenpakets. In Kapitel 3.9.1 wird dieses Thema genauer besprochen. Der Authentication-Header kann auch in Verbindung mit dem ESP-Header verwendet werden.
- **Encapsulation Security Payload:** Der Encapsulation-Security-Payload-Header (kurz ESP) wird in Kapitel 3.9.3 detailliert erläutert. Er bietet sowohl Authentizität als auch Verschlüsselung der Daten. ESP kann mit dem Authentication-Header kombiniert werden.
- **Destination Options:** Optionen für das Ziel eines eingekapselten Pakets.

Ihnen dürfte vielleicht aufgefallen sein, dass der »Destination Options«-Header zweimal gelistet ist. Dies liegt daran, dass nur der Erste für das eigentliche IPv6-Paket gilt. Der Letztere bezieht sich auf die Einkapselung des Pakets via IPSec. Jeder Extension Header sollte nur ein einziges Mal in einem IPv6-Paket vorkommen. Die einzige Ausnahme ist der »Destination-Options«-Header, der bis zu zweimal vorkommen darf.

Jeder Header gibt hierbei den nächsten Header (»Next Header«) an (die Aneinanderreihung der Header ist in Abbildung 2.8 dargestellt). Das bedeutet, wenn ein IPv6-Paket, bestehend aus einem IPv6-Header, einem Routing Header und einem TCP-Segment, versendet wird, gibt das Next Header-Feld im Basis-Header an, dass es sich beim nächsten Header um den Routing Header handelt. Der Routing Header gibt wiederum an, dass der nächste Header ein TCP-Header ist.

Der Wert »59« im Feld Next Header gibt an, dass dem aktuellen Header kein weiterer Header folgt.

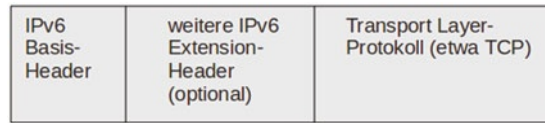


Abbildung 2.8: Aneinanderreihung der IPv6-Extension Headers

2.8.2 Sicherheit von IPv6

Auch wenn es viele Leute glauben, so ist IPv6 trotzdem nicht wesentlich sicherer als IPv4. Einige Details machen das neue Protokoll jedoch etwas sicherer:

Da keine Broadcast-Adressen existieren, kann die Verfügbarkeit von Systemen nicht einfach mit einem Broadcast-Ping überprüft werden. Zudem sind dadurch Denial-of-Service-Angriffe, bei denen ein Angreifer Pakete mit falscher Absenderadresse an Broadcast-Adressen schickt, nun auf diese Weise nicht mehr möglich.

Das Sammeln von Informationen im Netzwerk über IP-Record-Route-Optionen oder das ID-Feld im IPv4-Header fällt ebenso weg, da diese schlicht nicht mehr vorhanden sind.

Des Weiteren sind die Subnetze größer, was einen Netzwerkscan verkompliziert.

2.9 Internet Layer: ICMPv6

So, wie es bei IPv4 der Fall ist, bekam auch IPv6 ein ICMP-Protokoll verpasst. ICMPv6 hat die Protokollnummer 58 und ist generell wie sein Vorgänger ICMPv4 aufgebaut.

Jeweils 8 Bit stellen den ICMP-Type und -Code dar. Es folgen eine 16-Bit Checksum und die typabhängigen Daten.

2.9.1 ICMPv6-Typen

Seit Version 6 des ICMP-Protokolls sind die ICMP-Typen in zwei Bereiche unterteilt: Fehlermeldungen und Informationsmeldungen. Den Fehlermeldungen wurde der Typbereich 0 bis 127, den Informationsmeldungen der Bereich 128 bis 255 zugewiesen. Zudem verfügt die neue ICMP-Version nun auch über Multicast-Fähigkeiten, wie sie das IGMP-Protokoll zur Verfügung gestellt hat, und Möglichkeiten zur automatischen Adressvergabe, Routingkonfiguration und der Neighbour Discovery (ein ARP-Ersatz, näheres hierzu erfahren Sie im RFC 2461 »Neighbour Discovery for IP Version 6«).

Fehlermeldungen

Es gibt folgende ICMPv6-Fehlertypen:

- 1, *Destination Unreachable*: Das Zielsystem ist aus einem Grund, der im Code genauer spezifiziert wird, nicht erreichbar.
- 2, *Paket to Big*: Das Paket ist zu groß für einen Netzabschnitt, weil es größer als die zulässige MTU ist. Diese Meldung wird nur versandt, wenn zusätzlich das »Do not Fragment«-Flag gesetzt ist.

- 3, *Time Exceeded*: Das Hop-Limit erreichte den Wert 0 und das Datagramm wurde verworfen.
- 4, *Parameter Problem*: Der Header des Paketes enthielt Fehler und konnte daher nicht korrekt ausgewertet werden.

Informationsmeldungen

Es gibt folgende ICMPv6-Informationsmeldungen:

- 128 *Echo Request*: Eine ICMP-Echo Anforderung (Vgl. ICMPv4 »Echo Request«).
- 129 *Echo Reply*: Die Antwort auf obige Anforderung (Vgl. ICMPv4 »Echo Reply«).
- 130 *Group Membership Query*: Hierbei handelt es sich um eine Funktionalität, wie sie bereits vom IGMP-Query bekannt ist.
- 131 *Group Membership Report*: Hierbei handelt es sich um eine Funktionalität, wie sie bereits vom IGMP-Report bekannt ist.
- 132 *Group Membership Reduction*: Dieser Typ wird zum Austritt aus einer Multicast-Gruppe verwendet.
- 133 *Router Solicitation*: Anfrage nach einem Router (Vgl. ICMPv4).
- 134 *Router Advertisement*: Router Bekanntmachung (Vgl. ICMPv4).
- 135 *Neighbour Solicitation*: Anfrage nach der MAC-Adresse der anderen Hosts im Netz. Diese Funktionalität ersetzt das Address Resolution Protocol (ARP).
- 136 *Neighbour Advertisement*: Dieser ICMP-Type gibt die MAC- Adresse einer Schnittstelle bekannt und stellt sozusagen ein ARP-Reply dar.
- 137 *Redirect*: Dieser Typ ist ebenfalls schon von ICMPv4 bekannt und wird zur Umleitung von Routen verwendet.

2.9.2 Sicherheit von ICMPv6

Wie auch für das IPv6 gilt, dass es nicht wesentlich sicherer ist als sein Vorgänger, so gilt dies auch für das Protokoll ICMPv6. Nach wie vor können gefälschte ICMP-Pakete versendet werden und so kann beispielsweise die Autokonfiguration von Rechnern (ein DHCP-Ersatz, der in ICMPv6 bereits integriert ist) gefälscht werden. Somit ist es einem Angreifer beispielsweise möglich, den gesamten Netzwerkverkehr eines Hosts über sich zu leiten, und somit zu sniffen, Man-in-the-Middle-Angriffe durchzuführen etc.

Auf ähnliche Weise kann auch ein *ICMPv6 Neighbor Discovery-Request* ausgenutzt und mit falschen Daten beantwortet werden.

Außerdem kann die *ICMPv6 Duplicate Address Detection* angegriffen werden, mit der ein Angreifer nach Belieben Adressen für einen Host, der einen Adresswechsel durchführt, als bereits vergeben signalisieren kann.

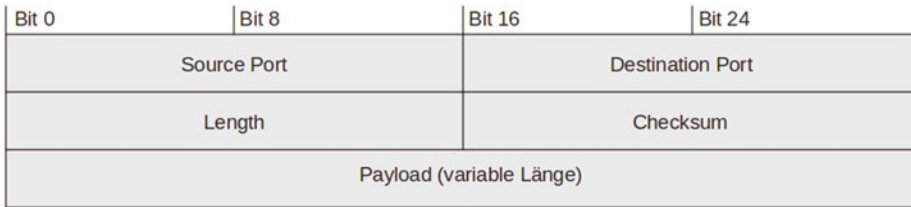


Abbildung 2.9: Der UDP-Header

2.10 Transport Layer: UDP

Zunächst werden wir das verbindungslose Protokoll UDP, später das verbindungsorientierte TCP betrachten. Das heißt, für UDP müssen weder Verbindungen aufgebaut werden, noch haben Verbindungen einen Status.

Bei UDP handelt es sich um ein verbindungsloses Transport-Layer-Protokoll. Die Abkürzung UDP steht für »User Datagram Protocol«. UDP wurde durch RFC 768 spezifiziert. UDP-Pakete werden als Datagramme bezeichnet.

Da UDP nur über minimale Funktionalitäten bezüglich einer Fehlerkorrektur – nämlich eine Checksum – verfügt, bleibt die Prüfung des Datenflusses, falls dieser kontrolliert ablaufen muss, dem Application-Layer überlassen. Aufgrund dieser Eigenschaft verfügt UDP zwar über einen recht kleinen Header, wird aber generell nicht in Applikationen eingesetzt, bei denen eine gesicherte Nachrichtenübertragung gewährleistet sein muss. Einsatzgebiete für UDP sind in der Regel solche, bei denen Datagramme in periodischen Abständen (Aktualisierungs-)Nachrichten übertragen sollen oder beim Verlust einer Nachricht keine Schwierigkeiten auftreten, da Anfragen einfach wiederholt werden (etwa bei DNS oder DHCP).

2.10.1 Der UDP-Header

Der UDP-Header ist, wie bereits angesprochen, nicht sonderlich groß, da er nur die für den Kernel des Betriebssystems wichtigsten Informationen beinhaltet. Der UDP-Header ist in Abbildung 2.9 dargestellt, er hat eine Größe von 8 Bytes (zum Vergleich: Der TCP-Header hat eine Größe von 20 Bytes). Die ersten 32 Bit bestehen aus dem Source-Port und dem Destination-Port. Es folgen die Längenangabe des Payloads und die Checksum zu jeweils 16 Bit. Da jeweils 16 Bit pro Port-Angabe zur Verfügung stehen, was auch beim TCP-Protokoll der Fall ist, können also maximal 65.535 verschiedene Ports adressiert werden, da der Null-Port nicht verwendet wird. Generell gilt: Die ersten (meist 1024) Ports sind in dem Sinne »privilegierte Ports«, als dass sie nur über administrative Rechte gebunden werden können. Die Portanzahl ist dabei betriebssystemspezifisch. Folglich kann ein Webserver an Port 80 nur mit root-Rechten gebunden werden. Unter den meisten gängigen Systemen ist es möglich, die UDP-Checksum zu vernachlässigen. Sie kann mit Kommandos wie »sysctl« abgeschaltet werden, was sich auf stark belasteten Systemen positiv auf deren Performance auswirken kann, allerdings auf Kosten der Verbindungs- und Datenqualität geht.

Die Checksum berechnet sich (und dies ist, wie wir später sehen werden auch beim TCP-Protokoll der Fall) aus dem Protokoll-Header, dem Payload und einem so genannten Pseudo-Header. Der Pseudo-Header beinhaltet die Source- und Destination-Adresse, die Protokoll-Nummer und die Länge von UDP-Header, sowie den Payload.

Typische UDP-Dienste sind das Domain Name System (DNS), (zumindest ältere) Implementierungen des Network Filesystems (NFS), DHCP und das Network Time Protocol (NTP).

2.10.2 Sicherheitsaspekte von UDP

Zunächst einmal ist es für erfahrene Netzwerkprogrammierer äußerst einfach, UDP-Datagramme zu spoofen. Dazu benötigt der Angreifer lediglich Superuser-Access auf einem System und entweder einen Paketgenerator, den er sich irgendwo aus dem Internet lädt, oder einen Compiler bzw. Interpreter für beispielsweise Perl-Code. So können unter falscher Absenderadresse beispielsweise DNS-Anfragen gefälscht werden.

Anmerkung: Praktisch alle Provider (etwa T-Online) haben mittlerweile Spoofing-Schutz in ihre Router integriert, sodass es vielen Skript-Kiddies gar nicht mehr möglich ist, von ihrem Heimrechner aus gespoofte Pakete an Hosts im Internet zu senden. Innerhalb von autonomen Systemen besteht das Problem hingegen weiterhin.

Da UDP im Gegensatz zu TCP keine Sequenz- und Bestätigungsnummern verwendet, ist es für einen Angreifer sehr einfach, eine »Verbindung« zu übernehmen (zu hijacken).

Keep State: Leider gibt es bei UDP-Kommunikationen aus Sicht der Paketfilter ein Problem: Die Software muss zwischen Verbindungsanfragen und Antwortpaketen unterscheiden können. Um dieses Problem zu lösen, merken sich die Paketfilter die Socket-Eigenschaften der ausgehenden Pakete (Source- und Destination-IP sowie Source- und Destination-Port). Ging also bereits binnen einer gewissen Zeitspanne ein Paket dieses Sockets über ein Interface, kann auf ein Antwortpaket geschlossen werden, andernfalls handelt es sich um eine Verbindungsanfrage.

2.11 Transport Layer: TCP

Das »Transmission Control Protocol« (TCP) ist das wohl meistgenutzte Transport-Layer-Protokoll der TCP/IP-Protokoll-Suite. TCP wurde durch RFC 793 beschrieben. Es gibt eine große Anzahl von Anwendungen, deren Protokolle auf TCP aufbauen. Die bekanntesten von ihnen sind wohl die folgenden (die Portangaben sind Standardports; sie können variieren):

- Webserver (HTTP-Protokoll (Port 80), HTTPS (Port 443))
- FTP-Server (FTP-Protokoll, Port 21 (und 20 (DATA)))
- Usenet-Server (NNTP-Protokoll, Port 119)
- Mailsysteme (SMTP-Protokoll (Port 25), POP3-Protokoll (Port 110), IMAP-Protokoll (Port 143), IMAPS (Port 993))
- SSH (SSH-Protokoll, Port 22)
- Telnet (TELNET-Protokoll, Port 23)

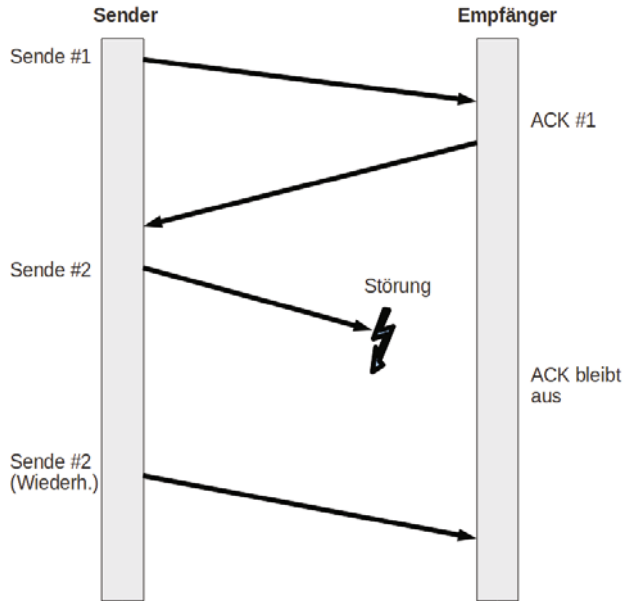


Abbildung 2.10: Ein Beispiel für TCP-Reliability

2.11.1 TCP-Reliability

TCP-Datenpakete (man bezeichnet diese als *Segmente*) werden, im Gegensatz zu UDP-Datagrammen, nicht als einzelne Nachricht (*Message*), sondern als Datenstrom (*Stream*) versendet. Dieser Datenstrom muss kontrolliert werden, sodass die Reihenfolge der Daten in diesem Stream nicht verfälscht wird. Hierfür wird die sogenannte Sequenznummer (*Sequence Number*) verwendet. Jedes Segment kann durch diese Sequence Number in die richtige Position des Streams eingeordnet werden. Die Seite, die das Segment empfängt, bestätigt die empfangenen Daten anschließend mit der sogenannten Bestätigungsnummer (*Acknowledgement Number*). Diese gibt aber auch die nächste, vom empfangenden Host erwartete Sequence Number an. Stimmen die erwartete und die empfangene Sequence Number nicht überein, kann TCP auf ein verloren gegangenes Segment schließen und sendet das verloren geglaubte Segment erneut aus. Auf diese Weise stellt TCP sicher, dass Datensegmente nicht verloren gehen und in der richtigen Reihenfolge ankommen.

In der folgenden Abbildung versendet Host A Daten an Host B. Host B empfängt und bestätigt diese. Darauf sendet Host A erneut Daten. Diese jedoch gehen unterwegs auf Grund von Netzwerkproblemen verloren, sodass Host B keine Bestätigung an Host A für die empfangenen Daten versendet. Daraufhin versendet Host A diese Daten erneut, in der Hoffnung, dass diese nun ankommen.¹²

¹²Die Zahlen hinter den Rauten entsprechen der Paketnummer, sie sind nicht mit der Sequence Number oder der Acknowledgement Number zu verwechseln.

Werden nun noch Sequenznummern und Bestätigungsnummern in die Betrachtung der Verbindung integriert, so gilt folgendes: Ein Paket mit einer Sequenznummer X muss mit einer Bestätigungsnummer $X+n$ bestätigt werden, wobei n das neue Datenoffset ist. Die Bestätigungsnummer ist dabei die für das folgende Paket erwartete Sequenznummer. Die erste Sequenznummer einer Verbindung wird von jedem der beiden Hosts, die miteinander kommunizieren selbst gewählt (randomisiert) und wird als *Initial Sequence Number* (ISN) bezeichnet.

Es lässt sich zum Thema TCP-Reliability klärend anmerken, dass das Ziel für TCP also nicht darin besteht, einzelne Pakete ans Ziel zu bringen, sondern darin, den gesamten Payload zuverlässig zu liefern [33]. Dabei kann es vorkommen, dass die Inhalte mehrerer kleiner, aber verlorengegangener, Pakete anschließend durch ein einziges Paket mit dem Gesamtpayload verschickt werden, wenn dies im Sinne der Reliability durchführbar ist [33].

2.11.2 Sende- und Empfangspuffer

TCP arbeitet mit sogenannten Sende- und Empfangspuffern. In diesen werden, wie sich wohl bereits erraten lässt, die empfangenen bzw. versendeten Daten gespeichert. Doch wozu speichert man diese überhaupt zwischen? Die wichtigsten Gründe hierfür sind, dass TCP Daten nur ab einer bestimmten Anzahl von Bytes im Sendepuffer versendet bzw. nur eine gewisse Anzahl von Daten pro Applikation im Empfangspuffer zwischenspeichert, bevor die Applikation die Daten abholen muss.

Es gibt einige Ausnahmeanwendungen (wie Telnet), die dieses Feature nicht nutzen, doch verursacht dies weitaus höhere Verbindungskosten: Ein TCP-Header ist im Normalfall 20 Bytes groß. Er baut wiederum auf einem IP-Header (ebenfalls 20 Bytes) auf. Und wenn man nun ein einzelnes Zeichen via Telnet versendet, versendet man auch jedes Mal diesen 40 Bytes großen Header. Die Geschwindigkeit von großen Datenübertragungen wäre bei dieser Übertragungsart sehr wahrscheinlich eine Katastrophe. Bei Anwendungen wie Telnet, bei denen sofort jedes eingetippte Zeichen auf dem Bildschirm angezeigt wird, führt an einer solchen Umsetzung allerdings kein Weg vorbei.

2.11.3 Flow-Control

Wie wir weiter unten sehen werden, verwendet TCP im Header ein Feld mit der Titulierung »Window Size«. Dieses teilt dem Empfänger mit, wie viele Daten der Sender im Empfangspuffer noch aufnehmen kann. Der Empfänger sendet daraufhin maximal so viele Daten, wie angegeben wurden, an den Sender zurück. Dieses Feature bezeichnet man auch als »Receive Window Sizing«.

Ein weiterer Flow-Control-Mechanismus nennt sich fast genauso, nämlich »Sliding receive windows«. Hierbei wird nicht abgewartet, bis die Bestätigung eines bereits versendeten Segments eintrifft, sondern es wird – in der Hoffnung, dass eine Bestätigung schon noch kommen wird – bereits vorsorglich das nächste Segment abgeschickt. Dies kann die Verbindungsperformance steigern.

Weitere Features des TCP-Flow-Control-Mechanismus sind die Veränderung des »Congestion Windows«, um den Sendefluss zu kontrollieren, und der »Slow Start«, um den Sendefluss mit einer bestmöglichen Segmentübertragungsgröße einzupendeln. Dabei wird zuerst ein Segment

Bit 0				Bit 8				Bit 16				Bit 24			
Source Port								Destination Port							
Sequence Number															
Acknowledgement Number															
Offset		Reserved Bits				Flags				Window					
Checksum								Urgent Pointer							
Options (nicht notwendig) und Payload (variable Länge)															

Abbildung 2.11: Der TCP-Header

mit einer sehr kleinen Größe versendet und dann die Segmentgröße von Segment zu Segment, bis Probleme auftauchen, immer weiter erhöht. So findet sich die optimale Größe für die Segmente, und der Overhead durch Frames und Header der Protokolle niedriger Layer wird auf ein Minimum optimiert.

2.11.4 Header

Kommen wir nun zum TCP-Header. Einige der oben angesprochenen Informationen werden Sie – zumindest indirekt – im Header wiederfinden.

Wie bei UDP werden die ersten 32 Bit des Headers für die jeweils 16 Bit langen Source- und Destination-Port-Angaben verwendet.

Es folgt eine 32 Bit lange Sequenznummer, die die Position der Payload im Datenstream angibt. Die Sequenznummer fängt je nach Implementierung mit einem Zufallswert oder null an, aufwärts zu zählen. Die Startnummer wird dabei als *Initial Sequence Number* (ISN) bezeichnet und bei gesetztem SYN-Flag im *3-Way-Handshake* übergeben (dazu gleich mehr).

Hinter der Sequence-Number folgt eine ebenfalls 32 Bit lange Bestätigungsnummer (man spricht von der Acknowledgement-Number), die die empfangenen Daten bestätigt.

Dadurch, dass beide Systeme, zwischen denen eine TCP-Verbindung besteht, diese beiden Nummern verwalten, kann festgestellt werden, ob auch alle Datenpakete tatsächlich angekommen sind und welche verloren gingen und demzufolge neu gesendet werden müssen.

Das nächste Feld (»Offset«, 4 Bit) gibt die Headerlänge in 32-Bit-Wörtern an. In der Regel beträgt dieser Wert »5« und wird bei der Verwendung der optionalen Felder (etwa TCP-Timestamps) erhöht.

Bei den nächsten vier Bit handelt es sich um einen reservierten Bereich, der mit Nullen überschrieben wird.

Für die Flags stehen 8 Bit zur Verfügung. Jedes gesetzte Bit hat eine spezielle Funktionalität, wobei zu beachten ist, dass es sich bei TCP um ein *Full-duplex*-Protokoll handelt, was im Zusam-

menhang mit dem Management der Verbindung, wie wir im nächsten Abschnitt sehen werden, wichtig ist:

- 0x01 (FIN): Der Sendevorgang des Kommunikators (Sender) zum Rezipienten (Empfänger) wird beendet.
- 0x02 (SYN): Die Verbindung vom Kommunikator zum Rezipienten wird aufgebaut.
- 0x04 (RST): Verbindungsreset.
- 0x08 (PSH): Die Payload wird direkt an den Applikation-Layer weitergereicht und nicht zwischengespeichert (PUSH).
- 0x10 (ACK): Sofern dieses Bit gesetzt ist, ist die ACK-Number gültig.
- 0x20 (URG): Der Dringlichkeitszeiger (Urgent Pointer) ist gültig.
- 0x40 (ECE) und 0x80 (CWR): Die »Explicit Congestion Notification«- und »Congestion Window Reduced«-Flags sind eine Erweiterung des TCP-Protokolls zum Umgang mit Netzwerküberlastungen. Die »Explicit Congestion Notifications« Bits des IP-Headers (vgl. »IP-Header«) stehen hiermit in Verbindung.

Das Empfangsfenster enthält eine Zahl. Diese Zahl gibt dem Rezipienten des Pakets an, wie viel Payload er in einem Paket an den Kommunikator zurücksenden darf. Kann der Kommunikator beispielsweise noch 1410 Bytes in seinen TCP-Buffer aufnehmen, beträgt der Wert des Empfangsfensters 1410.

Die TCP-Checksum wird auf dieselbe Art und Weise wie beim UDP-Protokoll (nämlich mit einem zusätzlichen Pseudo-Header) berechnet.

Der Urgent-Pointer zeigt auf vorrangig zu behandelnde Daten in der Payload. Er ist jedoch nur gültig, wenn das URG-Flag gesetzt ist.

2.11.5 Grundlegendes zur Datenkommunikation

3-Way-Handshake:

Die TCP-Verbindung wird mittels 3-Way-Handshake eingeleitet. Dieser 3-Way-Handshake wird deshalb so betitelt, weil er drei TCP-Segmente benötigt, um eine Verbindung beidseitig zu initialisieren. Beim ersten Paket jeder Seite wird dabei das SYN-Flag gesetzt. Dies wird von der Gegenseite jeweils mit einem gesetzten ACK bestätigt. Da die Seite, die ein passives Open (Verbindungsöffnen) ausführt, die Bestätigung des SYN-Segments gleich mit in das eigene Segment zum Verbindungsaufbau einbringen kann, werden von dieser Seite keine zwei, sondern lediglich ein einziges Segment versendet.

Da soeben der Begriff »passives Open« eingeführt wurde, soll natürlich auch die Unterscheidung zwischen aktivem und passivem Open nicht fehlen. Ein Host, der von sich aus eine Verbindung einleitet, führt ein aktives Open aus, und der Host, der die Verbindung akzeptiert, führt ein passives Open aus.

Sehen wir uns einmal den Datenverkehr beim Initialisieren einer Verbindung an. Wir verbinden uns vom Host *yorick* mit *eygo.sun*, Port 22, und überwachen auf *eygo.sun* die Datenpakete mittels *tcpdump*:

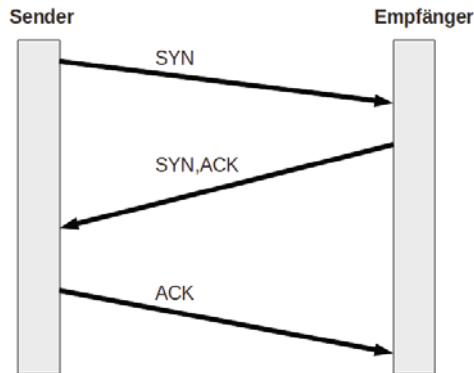


Abbildung 2.12: SYN- und ACK-Flags beim TCP-Handshake

```

eygo# tcpdump -i ne3
tcpdump: listening on ne3
20:15:24.883176 yorick.sun.1181 > eygo.sun.ssh: S
2591281295:2591281295(0) win 16384 <mss 1460,nop,
nop,sack0K> (DF)

20:15:24.883266 eygo.sun.ssh > yorick.sun.1181: S
1965163459:1965163459(0) ack 2591281296 win 16384
<mss 1460,nop,nop,sack0K> (DF)
20:15:24.883542 yorick.sun.1181 > eygo.sun.ssh: . ack
1 win 17520 (DF)
  
```

tcpdump zeigt uns an, welche Flags gesetzt sind. In den ersten beiden Paketen ist das SYN-Flag (S) gesetzt. ACK-Flags werden nicht dargestellt und sind durch *ack <nummer>* gekennzeichnet. Ist kein von *tcpdump* explizit gekennzeichnetes Flag gesetzt, wird ein Punkt ausgegeben – so wie im Paket 3.

Aufmerksamen Lesern wird bereits aufgefallen sein, dass es sich bei den Nummern hinter dem SYN-Flag um Sequenznummern, genauer gesagt um Initial-Sequence-Nummern, handelt. Das zweite Paket gibt die eigene ISN an *yorick.sun* weiter und bestätigt die ISN zudem noch mit einem ACK (Sequenznummer+1).

Datenkommunikation:

Ist die TCP-Verbindung beidseitig eingerichtet, können beide Seiten mit dem Datentransfer beginnen. Dabei bestätigen sich beide Seiten gegenseitig die empfangenen Daten mit den Sequence-Nummern.

Im Falle der obigen Verbindung wurde *telnet.exe* auf »yorick.sun« zum Verbindungsaufbau verwendet. Dabei wird jedes gesendete Zeichen an den Server als Extra-Segment versendet und mit einem PUSH-Flag (P) versehen.

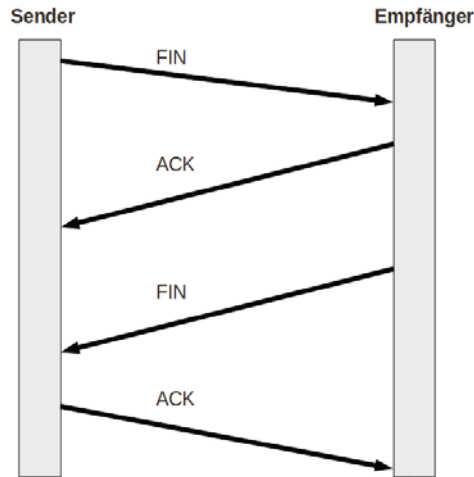


Abbildung 2.13: FIN- und ACK-Flags beim TCP-Verbindungsabbau

Das erste Segment beinhaltet die 22 Zeichen lange SSH-Versionsangabe und ein Newline-Zeichen. Die Bestätigung erfolgt mit einem ACK von *yorick.sun*. Nach der Eingabe von Return (*yorick*) wird die Eingabe seitens *eygo.sun* mit einem *Protocol mismatch* »bestätigt« und, wie wir gleich sehen werden, die Verbindung geschlossen.

```
20:15:24.886291 eygo.sun.ssh > yorick.sun.1181: P
1:24(23) ack 1 win 17520 (DF)
20:15:25.093096 yorick.sun.1181 > eygo.sun.ssh: . ack
24 win 17497 (DF)
20:15:28.593774 yorick.sun.1181 > eygo.sun.ssh: P
1:3(2) ack 24 win 17497 (DF)
20:15:28.594051 eygo.sun.ssh > yorick.sun.1181: P
24:43(19) ack 3 win 17520 (DF)
```

Verbindungsabbau:

Beim Verbindungsabbau sendet die Seite, die das Senden von Daten einstellen will, ein Segment mit gesetztem FIN-Flag an die Gegenseite. Die Gegenseite bestätigt dieses Segment mit einem ACK. Da TCP Full-Duplex ist, kann die Seite, die das FIN bestätigte, noch immer Daten senden. Wir sprechen beim Schließen einer Verbindung analog zum obigen 3-Way-Handshake von »aktivem« und »passivem« Close.

Die Beendigung der obigen Verbindung wurde natürlich ebenfalls protokolliert. Gehen Sie die oben in der Grafik gezeigten Schritte einfach einmal durch.

```
20:15:28.594100 eygo.sun.ssh > yorick.sun.1181: F
43:43(0) ack 3 win 17520 (DF)
```

```
20:15:28.594435 yorick.sun.1181 > eygo.sun.ssh: . ack
44 win 17478 (DF)
20:15:28.595272 yorick.sun.1181 > eygo.sun.ssh: F
3:3(0) ack 44 win 17478 (DF)
20:15:28.595388 eygo.sun.ssh > yorick.sun.1181: . ack
4 win 17520 (DF)
```

2.11.6 TCP: Grundlegende Sicherheitsaspekte

Es gibt verschiedene Angriffe gegen TCP. Es sind sowohl Denial-of-Service- als auch Spoofing- und Hijacking-Attacken möglich:

- Beim **Spoofing** von TCP-Paketen muss der Angreifer im Gegensatz zum UDP-Spoofing neben der Source- und Destination-Portnummer auch die Sequenz- und Bestätigungsnummern kennen. Entweder muss er diese erraten, kann diese bei älteren TCP-Implementierungen errechnen oder hat diese gesniffert. Darauf aufbauend kann wiederum ein Hijacking-Angriff ausgeführt werden.
- Mittels eines **Hijacking**-Angriffs können bestehende TCP-Verbindungen übernommen werden. Eine Möglichkeit ist die einfache Desynchronisation der Verbindung, wodurch beide Kommunikatoren der Meinung sind, von ihrem Gegenüber andere Sequenznummern bekommen zu müssen. Ist dies der Fall, bricht ein sogenannter »Ack-Storm« aus, da jeder seine Bestätigungen sendet. Bei einem kompletten Hijacking-Angriff (es gibt verschiedene Möglichkeiten, diesen durchzuführen) wird die Verbindung entweder übernommen und beendet oder geschlossen und reinitialisiert, oder es werden Daten injiziert und die Verbindung anschließend resynchronisiert. Im Optimalfall bemerken die Opfersysteme den Hijacking-Angriff nicht. Zudem ist es durch TCP-Hijacking möglich, einige Authentifizierungsmaßnahmen (z.B. S/Key) zu umgehen.
- Die sogenannten **Reset-Attacken** setzen das RST-Flag in einem gespooften Paket und injizieren es in eine bestehende Verbindung. Dazu muss zwar die Sequenznummer der Verbindung bekannt sein, jedoch ist bekannt, dass es kein allzu großes Problem ist, diese in etwa zu erraten. Dieser Angriff betrifft besonders Systeme, die eine stetige Verbindung benötigen, etwa BGP-basiertes Routing. Hierfür gibt es nun mindestens zwei Gegenmaßnahmen: Die IETF schlug vor, nur Sequenznummern, die exakt der erwarteten entsprechen, seitens des Empfängers zu akzeptieren (andernfalls geschieht dies in einigen Fensterbereichen). Die zweite Gegenmaßnahme findet sich im OpenBSD-Projekt. Hier werden zufällige Ports für Verbindungen vergeben. Der Angreifer muss in diesem Fall also zusätzlich den Port kennen, um die Verbindung zu beenden.
- Beim **TCP-Syn-Flooding** wird ein Verbindungsstatus des TCP-Protokolls ausgenutzt. Dabei werden so viele (vom Angreifer dann nicht mehr beantwortete) Verbindungsversuche eingeleitet, dass der Server keine neuen Verbindungen mehr entgegennehmen kann, da er sich alle angeforderten Verbindungen merken muss. Man nennt diesen Verbindungsstatus auch »halb offen«. Als Gegenmaßnahme wurden SYN-Cookies eingeführt, wobei die

vom Server gehaltenen Verbindungsinformationen nicht lokal gehalten, sondern in der ISN untergebracht werden. Nachdem die zweite ACK-Nachricht des 3-Way-Handshakes beim Server eintraf, wird ein tatsächlicher lokaler Eintrag in der Tabelle der TCP-Verbindungen hinterlegt; die entsprechenden Verbindungsinformationen werden aus der bestätigten ISN herausgerechnet.

2.12 Application Layer: HTTP

Zur Übertragung von Webseiten über das Internet wird das zustandslose Protokoll HTTP (*hyper-text transfer protocol*) verwendet. Bei HTTP handelt es sich um ein Plaintext-Protokoll, welches verschiedene Anfragetypen (so genannte *Methods*) unterstützt, die vom Client an den Server gestellt werden können:

- GET: Anfrage einer Ressource (optionale Parameter können mit meist stark begrenzter Länge über die Adresse übertragen werden).
- POST: Anfrage mit (textuellem oder binärem) Inhalt seitens des Clients an eine Ressource des Servers (etwa Übertragen von Eingabedaten oder Dateien).
- HEAD: Mit HEAD kann, wie bei GET, eine Ressource angefragt werden. Anstelle der gesamten Response erhält der Client hierbei nur die Metainformationen, also den HTTP Response-Header als Antwort (und keinen Payload).
- OPTIONS: Abfrage, welche Anfragetypen unterstützt werden.
- CONNECT: Verbinden mit einer weiteren Instanz (HTTP-Proxy).
- PUT: Upload von neuen Ressourcen durch den Client.
- DELETE: Löschen einer Ressource auf dem Server.
- TRACE: Zurücksenden einer Anfrage zu Debugging-Zwecken, um diese auf Veränderungen hin zu überprüfen

Die aktuelle HTTP-Version ist 1.1 und wird von allen gängigen Browsern implementiert. Die Schreibweise für die HTTP-Version ist »HTTP/Version«, also etwa »HTTP/1.1«. Version 1.1 erfordert gegenüber der Vorgängerversion 1.0 die Angabe eines »Host:«-Headers, womit *virtuelle Hosts* (das sind mehrere Hosts auf einem Server mit derselben IP-Adresse) möglich sind. Die Vorgängerversion von HTTP/1.0 war übrigens HTTP/0.9 von 1991.

2.12.1 Aufbau des HTTP-Headers

Beim Aufbau des HTTP-Headers wird zwischen Request und Response unterschieden. Der Client sendet dabei immer in einer ersten Zeile folgenden Aufbau an den HTTP-Server: »[METHODOD] [URL] [HTTP-VERSION]«. Die restlichen Header-Bereiche sollen in dieser kurzen

Einführung nicht beachtet werden und sind (bis auf »Host:«) optional. Eine minimale HTTP/1.0-Anfrage sieht also beispielsweise wie folgt aus: »GET /index.html HTTP/1.0«. Für HTTP/1.1 müsste die Anfrage wie folgt aussehen: »GET /index.html HTTP/1.1« gefolgt von einer Zeile »Host: www.example.com«. HTTP-Requests werden mit zwei Zeichensequenzen, die jeweils Carriage Return und Linefeed enthalten (also »\r\n\r\n«), abgeschlossen.

Eine HTTP-URL setzt sich, gemäß RFC 2068, aus den folgenden Bestandteilen zusammen: »http://Host[:Port]/Pfad«, also beispielsweise »http://www.wendzel.de:80/index.html«. Die Angabe des Ports kann im Falle von Port 80 (der Standardport für HTTP) ausbleiben, sodass die Anfrage der URL »http://www.wendzel.de/index.html« völlig korrekt ist.

Der HTTP-Server antwortet auf Anfragen (Requests) mit einer Response der Form: »[HTTP-Version] [Statuscode] [Status-Bezeichnung]«, also etwa: »HTTP/1.1 200 OK« für die erfolgreiche Ausführung (Code 200, Bezeichnung »OK«) eines HTTP/1.1-Requests. Status-Codes im Bereich von 100-199 sind Statusmeldungen (etwa »102 Processing«), im Bereich von 200-299 Erfolgsmeldungen (etwa »200 OK«), im Bereich von 300-399 Umleitungen (etwa »301 Moved Permanently«), im Bereich von 400-499 clientseitige Fehlermeldungen (etwa »400 Not Found« für eine vom Client angefragte Ressource, die nicht auf dem Server existiert) und im Bereich von 500-599 serverseitige Fehlermeldungen (etwa »500 Internal Server Error«).

Bei einer Response werden zudem einige weitere Parameter an den Client geliefert. Dazu zählen etwa Informationen über die Webserver-Software (Produkt, gegebenenfalls dessen Version und geladene Extension-Module), ein Timestamp, Informationen zur Content-Länge, zum Verbindungsverhalten (dazu gleich mehr) und zum Content-Type und seiner Codierung (etwa »text/html, charset=UTF-8«).

Wie gerade erwähnt wird der Client vom Server über den Umgang mit der bestehenden HTTP-Verbindung informiert. Dabei liefert er entweder den Wert »close« oder »keep-alive« zurück. Keep-Alive-Verbindungen werden vom Server nicht sofort nach der Response terminiert, um weitere Verbindungsanfragen ohne erneutes Verbinden zu erlauben (etwa für die in einer HTML-Datei verlinkten/eingebetteten Ressourcen, wie Bilder). Close-Verbindungen werden hingegen direkt terminiert. Einige Beispielanfragen sollen den Aufbau von HTTP-Requests und HTTP-Responses verdeutlichen.

Zunächst sollen mit HTTP/1.0-Methoden abgefragt werden, die der Webserver der Hochschule Augsburg unterstützt. Der Server antwortet mit den entsprechenden Methoden (Zeile »Allow«), Informationen zu seiner Software und weiteren Daten. Die Content-Length ist 0, da keine weiteren Inhalte (etwa Bilder oder eine Webseite) übertragen wurden.

```
swendzel@steffenmobile:~$ telnet www.rz.fh-augsburg.de 80
Trying 141.82.16.40...
Connected to www.rz.fh-augsburg.de.
Escape character is '^]'.
OPTIONS / HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 02 May 2011 13:31:42 GMT
Server: Apache/2.2.9 (Unix) mod_ssl/2.2.9 OpenSSL/0.9.7d DAV/2
mod_fastcgi/2.4.6
```

```
Allow: GET,HEAD,POST,OPTIONS,TRACE
Content-Length: 0
Connection: close
Content-Type: text/html
Connection closed by foreign host.
```

Als nächstes soll dieselbe Anfrage über HTTP/1.1 gestellt werden, wozu der Parameter »Host« angegeben werden muss.

```
swendzel@steffenmobile:~$ telnet www.rz.fh-augsburg.de 80
Trying 141.82.16.40...
Connected to www.rz.fh-augsburg.de.
Escape character is '^]'.
OPTIONS / HTTP/1.1
Host: www.rz.fh-augsburg.de

HTTP/1.1 200 OK
Date: Mon, 02 May 2011 13:31:49 GMT
Server: Apache/2.2.9 (Unix) mod_ssl/2.2.9 OpenSSL/0.9.7d DAV/2
mod_fastcgi/2.4.6
Allow: GET,HEAD,POST,OPTIONS,TRACE
Content-Length: 0
Content-Type: text/html
```

Nun möchten wir mit der GET-Methode die Startseite von *www.wendzel.de* abfragen und erhalten als Antwort (hier gekürzt dargestellt) den HTML-Code der Seite mit einer Gesamtlänge von 10217 Bytes.

```
swendzel@steffenmobile:~$ telnet www.wendzel.de 80
Trying 89.110.146.195...
Connected to www.wendzel.de.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.wendzel.de
```

```
HTTP/1.1 200 OK
Date: Thu, 16 Jun 2011 19:31:08 GMT
Server: Apache
Last-Modified: Fri, 03 Jun 2011 23:16:17 GMT
ETag: "697004a-27e9-4a4d6f0d09240"
Accept-Ranges: bytes
Content-Length: 10217
Content-Type: text/html; charset=UTF-8
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
```

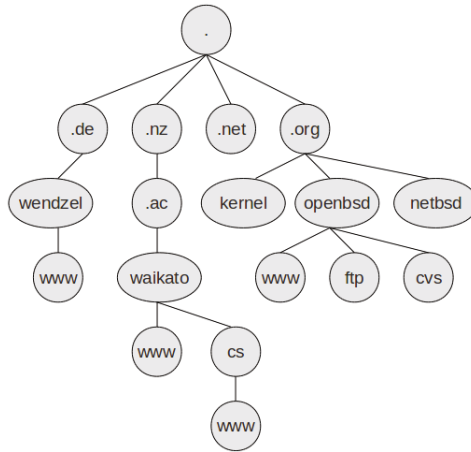



Abbildung 2.14: Hierarchie der DNS-Domains.

```

Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
...
...
Connection closed by foreign host.

```

2.13 Application Layer: Domain Name System

Hauptaufgabe des Domain Name Systems (DNS) ist die Übersetzung von Hostnames in IP-Adressen (und vice versa). DNS ist ein hierarchisches System, dessen Ausgangspunkt als ».« (ein Punkt) bezeichnet wird. Die hierarchische Organisation erfolgt in sogenannten *Domains*, die durch Punkte getrennt werden, etwa *whois.denic.de* (Host *whois* unter der Domain *denic* unter der Domain *de*), siehe Abbildung 2.14.

Hinter der Domain *de* kann ebenfalls ein Punkt stehen, der aber optional ist. Entsprechend ist die Domain »*www.fh-augsburg.de.*« korrekt angegeben. Seit einigen Jahren gibt es auch Umlautdomains (etwa *müller.de*). Domainnamen müssen mit einem Buchstaben beginnen, können maximal 63 Zeichen beinhalten und dürfen Buchstaben, den Bindestrich (»-«) und Zahlen beinhalten. Es können bis zu 127 Domainlevel erstellt werden. DNS-Namen dürfen nur einmalig vergeben werden. Die obersten Domains der Hierarchieebene werden dabei als *Top Level Domains* (TLDs) bezeichnet, wobei verschiedene Arten solcher Top Level Domains existieren [131]:

- Länder: .de (Deutschland), .at (Österreich), .us (Vereinigte Staaten), .nz (Neuseeland), .au (Australien)

- Allgemeines: com (kommerzielle Domains), org (Organisationen), gov (Regierungsorganisationen), mil (militärische Organisationen), edu (Bildungseinrichtungen) usw.
- Seit 2000: .biz (Business), .info (Informationen), .name (Namen von Personen), .pro (freie Berufe), .aero (Luftfahrtindustrie), .museum etc.
- Mit Beschluss der ICANN vom Juni 2011: Unternehmenseigene Top-Level-Domains (diese können von der ICANN nach Prüfung abgelehnt werden und sind äußerst kostspielig).¹³

Die Organisation von DNS ist dezentral: Server sind für einen Bereich (eine so genannte *Zone*) zuständig (*authoritative server*) und können die Informationen anderer Server zwischenspeichern (*DNS-Caching*).

Obwohl DNS dezentral organisiert ist, gibt es in der Hierarchie eine oberste Instanz: Die so genannten Rootserver. Es gibt derzeit 13 solcher Rootserver (durch Anycasting existieren in Wirklichkeit über 100 Rootserver), die in der Vergangenheit Ziel von DoS-Angriffen wurden (durch das Caching der regionalen Server konnten größere Probleme im DNS allerdings vermieden werden). Jeder Client darf diese Server direkt anfragen. Aus Gründen der Stabilität laufen die Rootserver mit unterschiedlichen Betriebssystemen (i.d.R. unixartig, etwa Solaris) und unterschiedlicher Serversoftware (i.d.R. jedoch BIND). Die meisten Root-Nameserver stehen in den USA. Rootserver verteilen die Adressen der DNS-Server für die Top-Level-Domains.

2.13.1 Resource Records

DNS kennt sogenannte Resource Records. Diese sind in vier Klassen organisiert, wobei nur die Klasse *IN* (Internet) für dieses Buch relevant ist. Es können dabei bis zu 65536 verschiedene Resource Records (RRs) in DNS definiert werden. Die wichtigsten sind:

- *A*: IPv4-Adresse für einen Host
- *AAAA*: IPv6-Adresse für einen Host
- *CNAME*: kanonischer Hostname
- *DNSKEY*: öffentlicher Schlüssel einer Zone für DNSSec¹⁴
- *MX*: *Mail Exchange* für die Angabe der Mailserver einer Domain samt ihrer Priorität
- *NAPTR*: Erweiterung der A-Records um zusätzliche Funktionalität, etwa Angabe des verwendeten Protokolls eines Servers
- *NS*: Nameserver einer Domain
- *PTR*: Umwandlung einer IP-Adresse in einen DNS-Name (Gegenstück zu A/AAAA)

¹³Quelle: <http://www.linux-magazin.de/NEWS/ICANN-stimmt-fuer-neue-Generic-Top-Level-Domains>

¹⁴DNSSec kann im Kontext dieses Buches nicht ausführlich besprochen werden. Es sei allerdings gesagt, dass es sich bei DNSSec um eine Erweiterung des DNS handelt, die die Nachrichten-Authentizität und -Integrität sicherstellt. DNSSec kümmert sich also, anders formuliert, darum, dass Nachrichten nicht manipuliert werden und zudem sichergestellt ist, dass Nachrichten tatsächlich vom gewünschten Absender stammen.

- **RRSIG:** Dieser Resource Record ermöglicht die Signatur anderer Resource Records und ist ebenfalls Bestandteil von DNSSec.
- **SOA:** *Start of Authority*, enthält Informationen zur autoritativen Zone eines DNS-Servers (Kontaktinformation zum Administrator, Standard-Lebenszeitspanne der Resource Records, Refresh-Time und weitere Werte).
- **TXT:** Freitext

2.13.2 Resolving

Jeder Rechner beinhaltet einen lokalen Resolver. Unter unixartigen Systemen wird ein Resolver über die Datei `/etc/resolv.conf` konfiguriert, in der die IP-Adressen von Nameservern angegeben werden, die ein Client ansprechen soll, um einen Resource Record abzufragen. Dabei wird bzgl. der Nameserver eine Unterscheidung hinsichtlich ihres Auflösungsverfahrens getätigt:

- **Iteratives Resolving:** Der Nameserver A fragt den ihm bekannten Nameserver B der nächst höheren Instanz nach den gewünschten DNS-Informationen. Sollte der Nameserver B diese Informationen nicht kennen, leitet A die Anfrage an den hierarchisch nächst höheren Nameserver C weiter usw., bis er am Ende einen Rootserver anfragt.
- **Rekursives Resolving:** Nameserver A fragt Nameserver B nach Informationen. Hat Nameserver B diese Informationen nicht, fragt B selbst den nächst höheren Server selbstständig an. Das weitere Vorgehen für den Fall, das eine Antwort nicht erhalten wurde, läuft rekursiv.

2.13.3 Der DNS-Header

Der DNS-Header gliedert sich in folgende Headerbereiche, die in Abbildung 2.15 dargestellt sind: Header Section (das ist der eigentliche Protokollheader), Questions Section (gestellte Fragen nach Resource Records), Answer Section (Antworten für gestellte Fragen), Authority Section (Hinweise zu alternativen, zuständigen DNS-Servern für die gestellte Anfrage) und Additional Section (zusätzliche Hinweise in Form weiterer Resource Records).

Ein UDP-DNS-Request kann maximal 512 Bytes groß sein. Für TCP können Streams versendet werden, die nicht unter diese Beschränkung fallen. Während alle Sections, bis auf die Header Section, nur Resource Records beinhalten, ist die Header Section nach einem bestimmten Schema aufgebaut. Sie enthält neben einer ID zur Zuordnung von Anfragen und Antworten, folgende Bestandteile (ebenfalls in Abbildung 2.15 dargestellt):

- **QR-Bit.** Das QR-Bit gibt an, um welche Art DNS-Nachricht es sich handelt; ein 0er-Bit steht für eine Anfrage (Query) und ein 1er-Bit für eine Antwort (Response).
- **Opcode.** Der Opcode gibt die Art der DNS-Anfrage an. Es handelt sich dabei entweder um eine Standardabfrage (*Standard Query*), eine inverse Abfrage (*Inverse Query*), die für die Umkehrung (also Negierung) des eigentlichen Requests verwendet wird, um eine Abfrage des Server-Zustands (*Server Status Request*), eine Benachrichtigung (*Notification*) oder um eine Aktualisierung (*Update*).

Bit 0	Bit 8	Bit 16		Bit 24	
Identification Number		Q R	Opcode	Flags	Rcode
#Question Resource Records		#Answer Resource Records			
#Authoritative Resource Records		#Additional Resource Records			
Question Resource Records					
Answer Resource Records					
Authoritative Resource Records					
Additional Resource Records					

Abbildung 2.15: Der DNS-Header.

- *DNS-Flags*. Weiterhin enthält der DNS-Header eine Reihe möglicher Flags:
 - AA: Dieses Flag (*Authoritative Answer*) ist gesetzt, wenn der angefragte DNS-Server autorisiert ist, Antworten im Rahmen der angefragten Domain zu liefern. Ein Server, der eine Anfrage für eine fremde Zone beantwortet (für diese Zone also etwa nur als Caching-Server fungiert), muss dieses Flag nicht setzen.
 - TC (truncated): Nur die ersten 512 Bytes der Response sind enthalten und die restlichen Daten sind abgeschnitten.
 - RD (recursion desired): Zuvor haben wir den Unterschied zwischen iterativem und rekursivem Resolving betrachtet. Setzt ein Client in einer Anfrage das RD-Bit, so fordert er ein rekursives Resolving an.
 - RA (recursion available): Da rekursives Resolving nicht immer verfügbar sein muss, kann ein Server mit dem RA-Flag signalisieren, dass er rekursives Resolving unterstützt.
 - Z-Bit (reserviert)
 - Die beiden Flags AD (*Authenticated Data*) und CD (*Checking Disabled*) stehen im Kontext von DNSSec. Mit gesetztem AD-Flag signalisiert ein DNS-Server, dass eine Überprüfung des RRs stattfand – die enthaltenen Informationen sind authentisiert. Das CD-Flag wird hingegen vom Client verwendet, um dem Server zu signalisieren, dass nicht überprüfte Antworten akzeptiert werden (der Client muss jedoch noch eigenständig Überprüfungen der Resource Records durchführen, weshalb er mit gesetztem CD-Flag dem Server die Arbeit ersparen kann) [8].
- *Rcode*. Dieser Wert gibt den Response-Code an. Die IANA definiert eine Liste möglicher Werte für den Rcode¹⁵). Die wichtigsten Werte sind:

¹⁵<http://www.iana.org/assignments/dns-parameters>

- 0 (es lag kein Fehler vor)
 - 1 (*Format error*, die Nachricht ist nicht interpretierbar.)
 - 2 (*Server failure*, ein serverseitiges Problem liegt vor.)
 - 3 (*Non-existent domain*, angefragte Domain existiert nicht.)
 - 4 (*Not implemented*, ein angefragtes Feature ist nicht verfügbar. Es ist beispielsweise möglich, dass kein rekursives Resolving verfügbar ist, der Client aber in einer Anfrage das RD-Flag setzte.)
 - 5 (*Refused*, die Anfrage wurde abgelehnt.)
- *Total Questions*, 16 Bit: Dieser Wert gibt die Anzahl der enthaltenen Anfragen an den Server an.
 - *Total Answers*, 16 Bit: Anzahl der enthaltenen Antworten vom Server an den Client.
 - *Total Authority Resource Records*, 16 Bit: Anzahl der enthaltenen Hinweise auf autorisierte Server in einer Antwort.
 - *Total Additional Resource Records*, 16 Bit: Anzahl der enthaltenen zusätzlichen Resource Records in einer Antwort.

Zusammenfassung

Protokolle des TCP/IP-Modells sind in einer Schichtenarchitektur organisiert. Die Aufgaben der Schichten unterscheiden sich stark, so ist die unterste Schicht etwa für die physikalische Datenübertragung und die darüber liegende Schicht für das Routing verantwortlich. Die dritte Schicht stellt Multiplexing bereit und die oberste Schicht beinhaltet anwendungsspezifische Funktionen. Die wichtigsten Protokolle der TCP/IP-Suite sind das ARP-Protokoll zur Übersetzung zwischen IP-Adressen und Hardware-Adressen, das Internet Protocol (IP) samt Control Protocol (ICMP) sowie die neueren Versionen IPv6 und ICMPv6, die sich auf dem Internet Layer befinden, sowie UDP und TCP auf der Transportschicht, und verschiedene Protokolle der Anwendungsschicht (wie etwa HTTP und DNS). Viele weitere Protokolle existieren, sind aber im Rahmen dieses Buches nicht von nennenswerter Bedeutung.

Tunnel und verdeckte Kanäle im Netz
Grundlagen, Protokolle, Sicherheit und Methoden
Wendzel, S.
2012, VIII, 179 S. 57 Abb., Softcover
ISBN: 978-3-8348-1640-5