TCP/IP - Eine Einführung

Konrad Rosenbaum <konrad@silmor.de>

2007

Inhaltsverzeichnis

Kapitel 1

Überblick

TCP/IP steht für "Transport Control Protocol/Internetwork Protocol". Die TCP/IP Protokollfamilie ist, wie der Name schon sagt, das Kommunikationsprotokoll des Internet. Das eigentliche "Internet-Protokoll" ist dabei IP, TCP ist ein darauf aufsetzendes Datenstrom-Protokoll. Dieses Kapitel beschäftigt sich zunächst nur mit IP, über die nächsten Kapitel werden wir weitere Ebenen des TCP/IP-Protokoll-Stacks abarbeiten.

IP ist paketorientiert. Dies bedeutet, die auszutauschenden Daten werden in einzelne "Datenpakete" zerteilt und unabhängig voneinander über das Netzwerk versandt. Dies hat den Vorteil, dass beliebig viele Applikationen und beliebig viele Hosts sich das selbe Netzwerk (das selbe Kabel) teilen können.

IP ist verbindungslos. Es betrachtet jedes Paket als neue Kommunikation und behandelt es entsprechend. Somit werden Statusinformationen in den Verbindungsknoten (Routern) des Netzwerkes gespart und das Netz kann dynamisch auf Änderungen reagieren.

IP garantiert nichts. Ob ein Paket ankommt, nicht ankommt oder doppelt ankommt hängt ganz und gar von den Umständen im Netz ab. IP an sich gibt keine Garantien. Höher liegende Protokolle, die auf IP aufsetzen, müssen sich selbst um ihre Absicherung kümmern.

1.1 Geschichte

TCP/IP ist aus den Versuchen des ARPANET und später Internet entstanden. Um einen extrem kurzen Abriss zu geben:

1969 bekommt die Firma BBN von der amerikanischen ARPA (später in DARPA – Defense Advanced Research Projects Agency – umbenannt) den Auftrag das ARPANET zu entwickeln. Die initiale Installation des ARPANET umfasste die Grossrechner von gerade einmal vier Universitäten. Das ARPANET läuft zunächst mit dem NCP – dem Network Control Protocol.

1973 Kahn und Cerf entwickeln die ersten Designs für TCP/IP, die auf den Erfahrungen mit NCP aufbauen und das Netzwerk auf beliebige Infrastrukturen portieren sollen.

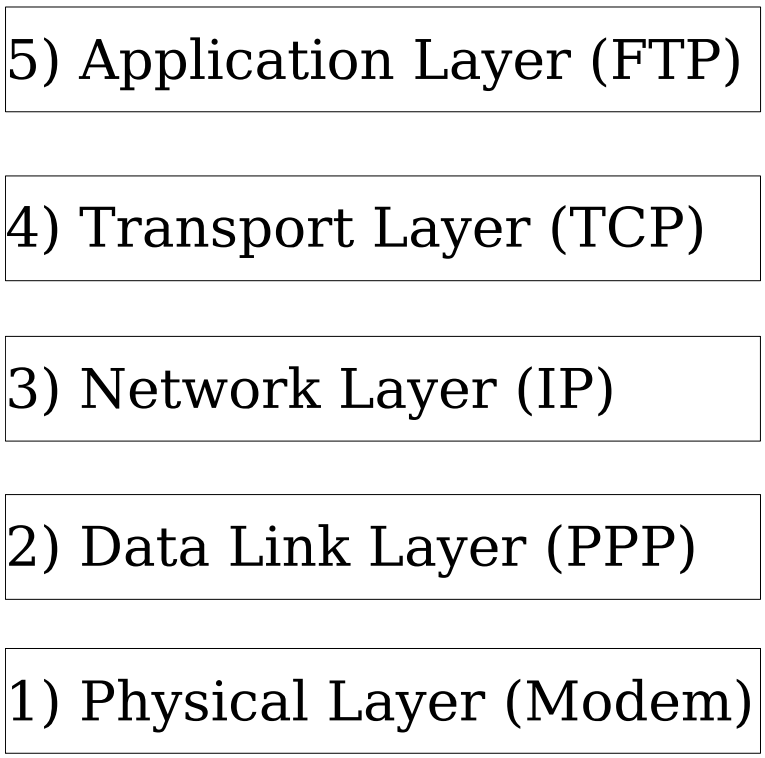

Abbildung 1.1: TCP/IP Protokollstack

1975–1983 Es werden verschiedene Implementationstests und weltweite Verbindungsversuche unternommen. TCP/IP entwickelt sich durch die Versionen 1–3.

1.1.1983 Das inzwischen wesentlich gröSSere ARPANET schaltet komplett auf TCP/IP Version 4 um. Weitere weltweite Netze (insb. Mailbox- und Usenet-Netze) folgen und werden mit dem ARPANET zum Internet zusammengeschaltet.

1.2 Protokollstack

In Grafik 1.1 ist der TCP/IP-Protokollstack zu sehen. Dieser ist so konzipiert, dass die jeweils unteren Ebenen den höher liegenden Dienste zur Verfügung stellen und die höheren Ebenen den unteren Funktionalität hinzufügen.

Die einzelnen Ebenen habe folgende Aufgaben:

Der Physical Layer ist die eigentliche Daten-Transport-Hardware. Sie kann Bits kodieren, auf einem Medium (meist Kupfer- oder Glasfaser-Kabel) transportieren und wieder zu Bits dekodieren. Ein Beispiel wäre das an einen Rechner angeschlossene Modem.

Der Data Link Layer kann Pakete in Bitströme verpacken und auf der anderen Seite diese wieder in Pakete verwandeln, die an die nächste Ebene weitergegeben werden können. Weiterhin besitzt der Data Link Layer Möglichkeiten einzelne Knoten, die an das selbe Medium angeschlossen sind zu identifizieren und individuell anzusprechen (z.B. einzelne Ethernet-Karten über ihre MAC-Adresse).

Häufig werden der Physical Layer und der Data Link Layer zu einer Ebene zusammengefasst.

Der Network Layer beherrscht das Senden, Empfangen und Weiterleiten von Paketen. Diese Ebene kennt die Zuordnung zwischen IP-Adressen und den einzelnen Links (Netzwerkanschlüssen) des Rechners. Sie kann anhand interner Tabellen herausfinden über welchen Link ein ausgehendes Paket gesendet werden muss. Diese Funktionen werden vom Protokoll "IP" abgebildet.

Eine weitere Funktion, die meist dieser Ebene zugeordnet wird, ist die Zuordnung von Link-Adressen (z.B. Ethernet-MACs) zu IP-Adressen auf dem direkt angeschlossenen Medium. Diese Zuordnung wird vom ARP Protokoll ermittelt und verwaltet.

Der Transport Layer bietet Protokolle, die von den Netzwerkapplikationen direkt benutzt werden können. Diese Ebene kann über Netzwerkports zuordnen welches laufende Programm welche Datenpakete erhalten muss. Die wichtigsten beiden Protokolle auf dieser Ebene sind TCP und UDP.

UDP ist ein einfaches Paketprotokoll. Die Applikation hat die Möglichkeit einzelne Pakete an andere Knoten im Netz zu versenden und Pakete zu empfangen. UDP kennt kein Verbindungskonzept (jedes Paket wird für sich behandelt) und gibt auch keine Garantie, dass das Paket ankommt — die Applikation muss sich selbst um Empfangssicherung kümmern.

TCP dagegen bietet ein Datenstromprotokoll. Die Applikation kann (virtuelle) Verbindungen zu anderen (entfernten) Applikationen aufbauen und Datenströme austauschen. TCP kümmert sich dabei um den Erhalt der Verbindung und die Empfangssicherung.

Auf dem Application Layer sind all die Protokolle angesiedelt, die man als Nutzer normalerweise (indirekt) wahrnimmt. Dies reicht vom eMail-Transport, über den Transport von Webseiten bis zu modernen P2P-Protokollen.

Kapitel 2

Grundlagen: Adressierung und Routing

Dieses Kapitel beschäftigt sich mit der Funktion des Network Layer und des Protokolls IP.

2.1 Adressen und Masken

IP-Adressen sind 32bit-Werte[1], die normalerweise als vier Dezimalstellen, die durch Punkte getrennt sind, dargestellt werden (z.B. 192.168.1.1). Jedem Knoten[2] im jeweiligen Netzwerk ist eindeutig[3] eine IP-Adresse zugeordnet, über die er identifiziert werden kann.

Lokal gehört zur Konfiguration eines Knotens nicht nur die IP-Adresse, sondern auch die Netzwerk-Maske. Dies ist eine Bitmaske anhand derer ermittelt werden kann in welchem Subnetz die jeweilige IP-Adresse angesiedelt ist. Die Netzwerk-Maske beginnt immer mit einer zusammenhängenden Reihe von auf 1 gesetzten Bits und endet mit 0-Bits. Eine binäre UND-Verknüpfung von IP-Adresse und Netzwerk-Maske ergibt die allen Knoten in diesem Subnetz gemeinsame Netzwerk-ID. Wenn zum Beispiel die IP-Adresse eines Knotens 192.168.1.1

[1] Dies bezieht sich auf IP Version 4. Mit den 128bit langen IPv6 Adressen beschäftigt sich Kapitel 6.

[2] D.h. jeder Netzwerkkart. Ein Rechner kann durchaus mehrere Karten besitzen.

[3] Man kann Ausnahmen schaffen, indem NAT – Network Address Translation – eingesetzt wird.

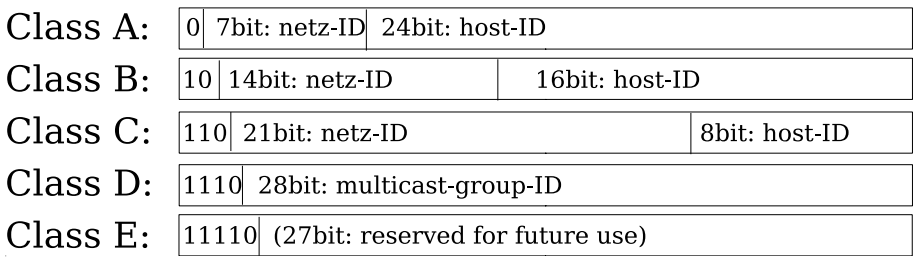

Abbildung 2.1: Adress-Klassen

und seine Netzwerk-Maske 255.255.255.0 ist, dann folget dass die IP-Adressen aller Knoten in diesem Subnetz mit 192.168.1 beginnen.

Seit einiger Zeit[4] können Netzwerkmasken auch einfach mit einem Schrägstrich und der Anzahl der 1-Bits an die IP-Adresse angehängt werden. In unserem Beispiel wäre dies 192.168.1.1/24.

Innerhalb jedes Subnetz gibt es zwei reservierte Adressen: die Adresse, die alle nicht zum Netzwerk gehörenden Bits auf 0 gesetzt hat ist für das Subnetz selbst reserviert und wird im Allgemeinen nicht benutzt. Die Adresse, die alle nicht zum Netzwerk gehörenden Bits auf 1 gesetzt hat (im Beispiel 192.168.1.255) ist die Broadcast-Adresse des Subnetzes und kann benutzt werden, um alle Knoten des Subnetz gleichzeitig anzusprechen[5]. Der Extremfall einer Broadcast-Adresse ist 255.255.255.255. Theoretisch liese sich damit das gesamte Internet ansprechen. Um Paketfluten zu verhindern wird diese Adresse allerdings nie über ein einzelnes Netzwerksegment hinausgeleitet[6].

Per Konvention gibt es zwei weitere reservierte Adressen: Netzwerk plus 1 (192.168.1.1) ist für den Subnetz-Server (z.B. Namensauflösung) reserviert. Broadcast minus 1 (192.168.1.254) für den Router in das nächsthöhere Netzwerk. Oft übernimmt einer der beiden beide Aufgaben.

Netzwerkadressen sind grob in fünf Klassen eingeteilt (siehe auch Abbildung 2.1):

Klasse A bezeichnet Adressen mit einer 8-Bit Netzmaske[7].

Innerhalb dieses Bereiches gibt es zwei reservierte Bereiche: 10.0.0.0/8 darf in privaten Netzwerken eingesetzt werden, ohne die Adressen vorher zu registrieren. Diese Adressen dürfen dann allerdings auch nicht aus dem privaten Netz herausgelassen werden (Provider verwerfen Pakete mit diesen IP-Adressen).

Das Netz 127.0.0.0/8 ist für das lokale Netzsegment reserviert. In der Praxis wird nur eine einzige Adresse aus diesem Bereich verwendet: die Adresse 127.0.0.1 wird benutzt, um den lokalen Rechner selbst anzusprechen — Pakete, die an diese Adresse gesendet werden, kommt immer auf dem selben Rechner an von dem sie gesendet wurden.

Eine weitere spezielle reservierte Adresse ist 0.0.0.0. Diese Adresse kann nicht an einen Netzwerkknoten zugewiesen werden und wird von IP-Implementationen verwendet, um zu signalisieren, dass ein Programm auf allen lokalen Netzwerkinterfaces verfügbar ist.

Die Klasse A umfasst alle Adressen von 0.0.0.0/8 bis 127.255.255.255/8.

Klasse B bezeichnet Adressen mit einer 16-Bit Netzmaske. Sie umfasst die Adressen 128.0.0.0/16 bis 191.255.255.255/16.

Aus diesem Bereich sind die 16 Netzwerke 172.16.0.0/16 bis 172.31.0.0/16 für private Nutzung reserviert.

---

[4] um genau zu sein: seit der "Erfindung" von IPv6

[5] Das bedeutet aber nicht, dass auch alle Knoten reagieren – Windows-Rechner reagieren z.B. nicht auf Broadcast Pings.

[6] Dies läSSt sich sehr gut dafür verwenden alle an ein Netzsegment angeschlossenen Rechner und deren IP-Adressen zu detektieren.

[7] Unabhängig von den hier vorgegeben Adress-Maske-Beziehungen können je nach lokalen Gegebenheiten natürlich abweichende Masken verwendet werden. Die Einteilung in Klassen dient eher der Vergabe von Adressen und als Richtlinie für Administratoren.

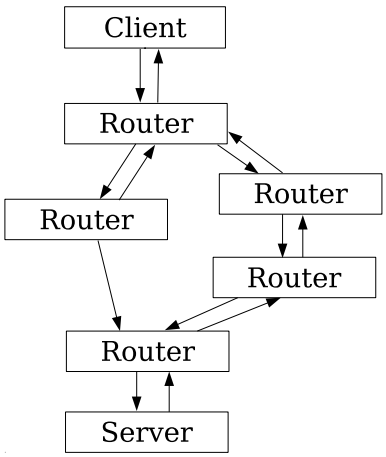

Abbildung 2.2: Beispiel: Netzwerkrouting

```
bash# route -n
Kernel IP routing table
Destination  Gateway      Genmask         Flags Metric Ref Use Iface
217.0.116.25 0.0.0.0      255.255.255.255 UH    0      0     0 ppp0
192.168.2.0  0.0.0.0      255.255.255.0   U     0      0     0 eth1
192.168.1.0  0.0.0.0      255.255.255.0   U     0      0     0 eth0
0.0.0.0      217.0.116.25 0.0.0.0         UG    0      0     0 ppp0
bash#
```

Abbildung 2.3: Beispiel: Routingtabelle

Klasse C bezeichnet Adressen mit einer 24-Bit Netzmaske. Sie umfasst die Adressen 192.0.0.0/24 bis 223.255.255.255/24.

Die 255 Netze 192.168.0.0/24 bis 192.168.255.0/24 sind für private Nutzung reserviert.

Klasse D fasst alle Multicast Adressen zusammen. Multicast-Adressen sind Adressen, die in beliebig viele Netze weitergeleitet und von beliebig vielen Rechnern empfangen werden können. Die kann z.B. von Multimedia-Streaming-Programmen genutzt werden.

Diese Klasse umfasst die Adressen 244.0.0.0 bis 239.255.255.255.

Klasse E umfasst einen Block von Adressen, der für zukünftige Anwendungen reserviert ist: 240.0.0.0 bis 247.255.255.255.

2.2 Routing

Sehr viele Rechner im Netzwerk sind sog. Router (manchmal auch Gateway genannt). Diese Router leiten Pakete von einem Netzwerksegment ins nächste Segment weiter. Damit dies funktioniert braucht jeder Router eine Entscheidungstabelle anhand derer er entscheiden kann auf welches Segment ein Paket

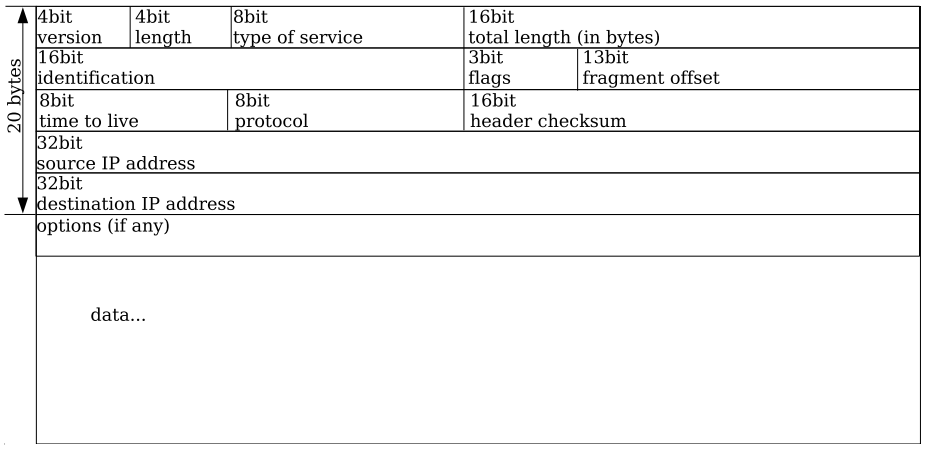

Abbildung 2.4: Aufbau eines IP-Pakets

als nächstes gesendet werden muss (oder ob es verworfen werden sollte). Diese Tabelle wird "Routingtabelle" genannt.

Eine Routingtabelle enthält Einträge, die dem Router sagen welcher Adressbereich (IP-Adresse plus Netzmaske) auf welches Segment (Interface-Name) gesendet werden soll. Bei einem typischen Linux-Router kann das zum Beispiel so aussehen, wie in Abbildung 2.3. Die Einträge einer Routingtabelle sind üblicherweise von der spezifischsten zur unspezifischsten Netzmaske sortiert und werden von oben nach unten abgearbeitet.

In Abbildung 2.3 ist die erste Zeile eine direkte Zuordung eines einzelnen Hosts zu einem Interface — in diesem Fall ist das dadurch begründet dass eine PPP-Verbindung exakt zwei Rechnerknoten miteinander verbindet[8]. Die beiden mittleren Zeilen sind Routingregeln für zwei angeschlossene Ethernet-Segmente. Die letzte Zeile ist die sog. Default-Route: mit einer Netzwerkmaske von 0.0.0.0 erfasst sie alle Pakete, die noch nicht von einer anderen Route erfasst wurden und sendet sie via PPP-Interface an den angegeben nächsten Router (hier als Gateway bezeichnet).

Kommt auf diesem Beispielrouter ein Paket für 192.168.1.5 an, dann wird dieses auf die dritte Routingregel passen und auf das Ethernetsegment der Karte "eth0" gesendet. Soll das Paket dagegen an 193.99.144.85 (www.heise.de) gesendet werden, so wird es lediglich auf die default Route passen und per PPP an den Gateway des Providers, der wiederum über die erste Regel erreichbar ist, weitergeleitet werden[9].

2.3 Aufbau von IP-Paketen

Abbildung 2.4 zeigt schematisch den Aufbau eines IP-Paketes – eine Zeile entspricht hier 32 Bit. Die einzelnen Felder haben folgende Bedeutung:

[8] In diesem ganz speziellen Fall mein DSL-Modem mit dem DSLAM beim Provider.

[9] Nicht sichtbar ist hier, dass dieser Router die Adressen der internen Netzwerke auf die Adresse des PPP-Interface übersetzen wird bevor ein Paket das interne Netz verläSSt — er wird also nicht versuchen private Adressen im Internet sichtbar zu machen. Diese Übersetzung ist allerdings nicht zum Verständnis von Routing notwendig.

Version enthält die Protokollversion von IP. Dieses Feld ist immer auf den Wert 4 gesetzt[10].

Header length enthält die Länge des gesamten IP-Headers in Einheiten von 32 Bit. Sollten keine Optionen verwendet werden ist dieses Feld auf 5 gesetzt (20 Bytes[11]).

Type of service (TOS) war vorgesehen, um optimiertes Routing zu ermöglichen. Dieses Feld wird nicht (mehr) genutzt und ist immer auf 0 gesetzt. Sogenannte TOS-Router benutzen meistens Heuristiken, um das Protokoll und seine Routing-Anforderungen zu ermitteln.

Total length enthält die gesamte Länge des Paketes in Bytes. Dadurch kann ein Paket (inklusive aller Header) maximal 65535 Bytes lang werden. Dieses Feld wird benötigt, da es einige Netzwerkmedien gibt, die zusätzliche Bytes an zu kurze Pakete anhängen[12].

Identification ist eine (kurzzeitig) eindeutige Kennung für ein Paket, die z.B. benutzt werden kann, um doppelt ausgelieferte Pakete zu filtern.

Time to live gibt die maximale Lebenszeit des Paketes in Sekunden an. Sollte ein Paket weniger als eine Sekunde auf einem Router verweilen wird trotzdem der Wert um 1 herabgesetzt. Dieser Wert existiert, um Pakete nicht endlos in Schleifen laufen zu lassen, falls ein Fehler in einer Routingtabelle auftritt. Erreicht dieser Wert Null, wird das Paket verworfen.

In der Praxis wird von kaum einer Implementation der Sekundenzähler implementiert und einfach nur bei jedem Weiterleiten dieser Wert um 1 herabgesetzt. Daher wird dieser Wert auch oft als Hop-Limit bezeichnet.

Protocol gibt das Protokoll der in diesem Paket verpackten Daten an. Für TCP wäre dies z.B. 6, für UDP 17 und für ICMP[13] 1.

2.4 Werkzeuge

Die meisten Betriebssysteme bringen einige Standardwerkzeuge zur Diagnose mit:

`route` kann benutzt werden, um sich die lokale Routingtabelle ausgeben zu lassen und sie eventuell zu manipulieren.

[10] Für IP Version 6 entsprechende auf den Wert 6.

[11] In RFCs und anderen Fachtexten wird oft der Begriff "Octet" verwendet, der eingeführt wurde, als noch nicht (nahezu) alle Rechnerarchitekturen 8-Bit Bytes verwendeten.

[12] Dies ist z.B. bei Gigabit Ethernet der Fall, bei dem ein Paket eine bestimmte Länge nicht unterschreiten darf, da sonst die Kollisionserkennung versagt.

[13] Mit diesem Protokoll werden Fehlermeldungen auf IP-Ebene ausgeliefert

`ping` sendet ein ICMP echo-request Paket an den Zielhost. Sollte das Paket ankommen (Host existiert, Firewall hat das Paket nicht verschluckt, Host ist korrekt konfiguriert), dann wird dieser mit einem ICMP echo-reply Paket antworten, was von Ping dann angezeigt wird.

`traceroute` (Windows: `tracert`) sendet Pakete zunächst mit einer TTL von 1 ab, was dazu führt dass bereits der nächste Router einen Fehler per ICMP zurücksendet. Bei jedem Schritt wird die TTL nun um 1 erhöht, bis der Zielrechner gefunden ist. Auf diese Weise läSSt sich verfolgen welchen Weg ein Paket durch das Netz bis zum Ziel nimmt.

`telnet` kann benutzt werden, um testweise Verbindungen zu TCP-Programmen auf anderen Rechnern aufzubauen und so zu prüfen, ob eine Verbindung überhaupt möglich ist oder ob ein Fehler auf einer höheren Ebene in einer komplexen Applikation zu suchen ist.

Kapitel 3

TCP & UDP

TCP und UDP sind die beiden wichtigsten Transportprotokolle für Applikationsdaten. Neben ihnen existieren weitere Protokolle für Applikationsdaten, die allerdings kaum eingesetzt werden.

Auf der selben Ebene, wie TCP und UDP bewegt sich auch das Fehlerprotokoll ICMP. Eine explizite Behandlung von ICMP würde allerdings für dieses Script etwas zu weit gehen. Der geneigte Leser sei auf die einschlägige Literatur verwiesen.

3.1 UDP

UDP ist ein sehr einfaches Paket-Sende-Protokoll, das IP lediglich um Ports erweitert, so dass mehrere Nutzer/Programme auf dem selben Host UDP gleichzeitig verwenden können, indem der Zielport als Identifikation des Empfängerprogramms dient.

Dies bedeutet insbesondere, dass UDP keine Garantie gibt, dass ein Paket tatsächlich ankommt (oder nicht doppelt ankommt).

In Abbildung 3.1 ist die Einordnung und der Aufbau eines UDP-Paketes zu sehen. Wie man sieht werden die Header der niedrigeren Protokolle unmittelbar

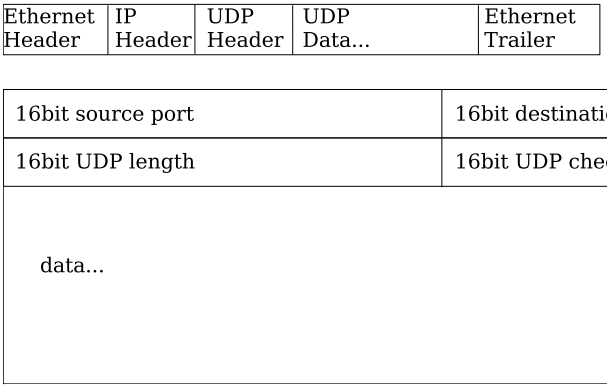

Abbildung 3.1: UDP: Einordnung in ein Ethernet-Paket und Paketaufbau

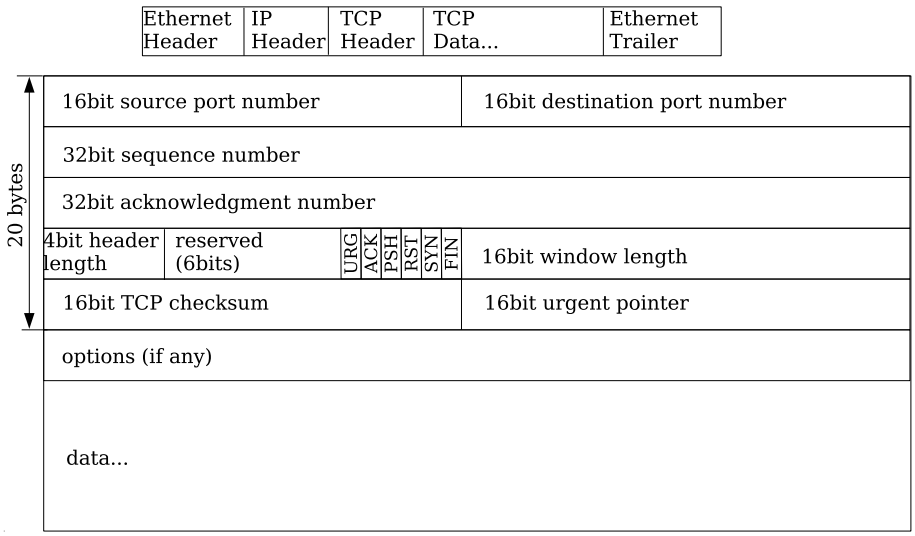

Abbildung 3.2: TCP: Einordnung in ein Ethernet-Paket und Paketaufbau

vor die Header der höheren Protokolle gestellt. Nach den Headern beginnen die eigentlichen Applikationsdaten, denen je nach physical layer ein Trailer folgen kann, der das Paket auf eine Mindestlänge bringt. Der eigentliche UDP-Header enthält lediglich Absender- und Empfänger-Port sowie eine Längenangabe der Daten und eine einfache Checksumme.

3.2 TCP

TCP ist ein Datenstromprotokoll, das virtuelle Verbindungen oberhalb von IP aufbauen kann. Ebenso wie UDP benutzt es 16bit Port-Nummern, über die Quell- und Ziel-Programm identifizierbar werden. Eine bestehende Verbindung wird dabei über die Kombination von Quell-IP-Adresse, Quell-Port, Ziel-IP-Adresse und Ziel-Port eindeutig identifiziert. Im Gegensatz zu UDP besitzt TCP die Fähigkeit nicht oder defekt angekommene Daten zu wiederholen, doppelt angekommene Daten zu verwerfen und die Pakete in die richtige Reihenfolge zu bringen bevor die Daten an den Empfänger ausgeliefert werden. Für Sender und Empfänger entsteht dabei der Eindruck mit einem zusammenhängenden Datenstrom zu arbeiten, während TCP sich um die Aufteilung in Pakete und deren Zusammensetzung beim Empfänger kümmert.

In Abbildung 3.2 ist die Einordnung und der Aufbau des TCP-Headers zu sehen. Die einzelnen Felder haben dabei folgende Bedeutung:

Source und destination port identifizieren zusammen mit Quell- und Ziel-IP-Adresse die Verbindung zu der dieses Paket gehört.

Die sequence number beginnt beim Verbindungsaufbau mit einer zufälligen Zahl und wird danach um die Anzahl der bereits (vor diesem Paket) gesendeten Bytes hochgezählt. Dadurch ist es dem Empfänger möglich die exakte Reihenfolge der Pakete und eventuell noch fehlende Pakete zu ermitteln.

Die acknowledgement number wird auf die sequence number des zuletzt erfolgreich und vollständig erhaltenen Bytes gesetzt. Sollte die Bestätigung eines Bytes/Pakets für eine längere Zeit (normalerweise 30s) ausbleiben versucht der sendende Host diese Daten noch einmal zu übermitteln, da er davon ausgeht, dass die Daten verlorengegangen sind.

Header length enthält die Länge des TCP-Headers in 32-Bit Worten (typischerweise 5).

Das reserved Feld ist immer auf 0 gesetzt und bleibt zukünftigen Erweiterungen von TCP vorbehalten.

Window size wird von den Kommunikationspartnern benutzt um dem jeweils anderen zu signalisieren wieviel Platz noch im Empfangspuffer für diese Verbindung vorhanden ist.

Die Flags von TCP werden benutzt um die Verbindungskontrolle und verschiedene Features zu steuern:

URG (Urgent) zeigt an, dass ein einzelnes Out-of-Band Byte im Paket enthalten ist. Der Urgent-Pointer wird benutzt um auf dieses Byte zu verweisen. Einige Protokolle benutzen dieses Feature, um Verbindungsresets durchzuführen oder Meldungen über den Zustand der Verbindung weiterzureichen.

ACK (Acknowledgement) wird gesetzt, um empfangene Bytes zu bestätigen. Die acknowledgement number ist in diesem Fall auf die zuletzt erfolgreicht empfangenen Bytes gesetzt.

PSH (Push) signalisiert dem Empfänger, dass er die Daten so schnell wie möglich an die Applikation ausliefern soll und nicht noch auf weitere Daten warten soll.

RST (Reset) wird genutzt um der Gegenseite zu signalisieren, dass die Verbindung zu diesem Paket nicht mehr besteht oder (als Antwort auf SYN) nicht aufgebaut werden kann.

SYN signalisiert dem Empfänger dass eine neue Verbindung gewünscht wird.

FIN signalisiert dem Empfänger dass die Verbindung beendet werden soll.

3.2.1 3-way Handshake

Abbildung 3.3 zeigt schematisch den Aufbau einer TCP-Verbindung. Dabei bittet der Client zunächst den Server um eine Verbindung (1), indem er ein TCP-Paket mit gesetztem SYN-Flag mit den gewünschten Ports sendet. Ist der Verbindungsaufbau möglich (es gibt eine Applikation auf dem Zielport) bestätigt

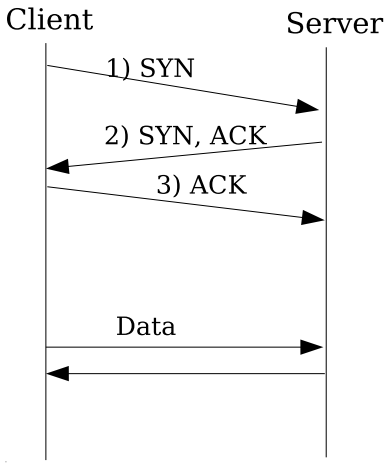

Abbildung 3.3: TCP Verbindungsaufbau

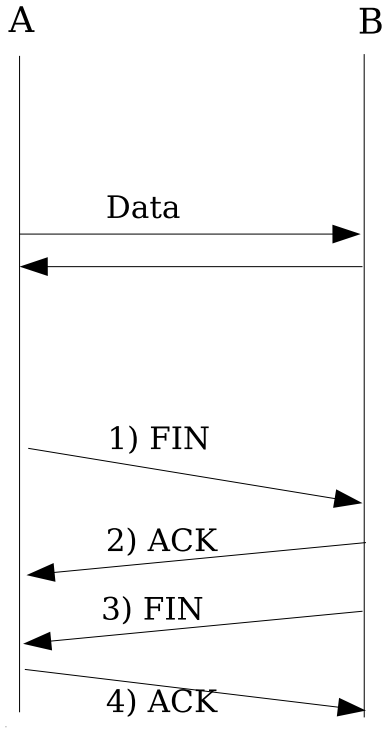

Abbildung 3.4: TCP Verbindungsabbau

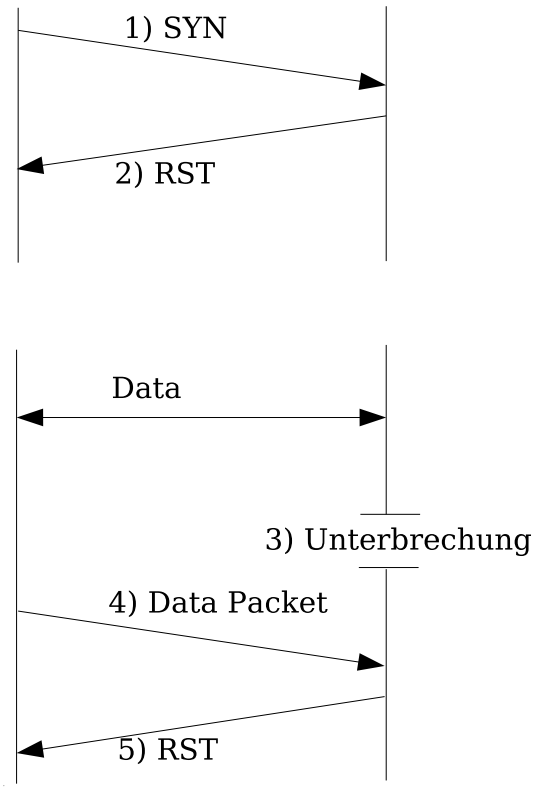

Abbildung 3.5: TCP Reset

der Server den Aufbau (2) und bittet seinerseit um Verbindung, indem ein Paket mit gesetzten SYN- und ACK-Flags gesendet wird. Als letztes bestätigt der Client den erfolgreichen Verbindungsaufbau (3) mit einem Paket bei dem ACK gesetzt ist. Ab diesem Zeitpunkt können beide Daten auf dieser Verbindung senden und empfangen.

Abbildung 3.4 zeigt den Abbau einer TCP-Verbindung. Jeder der beiden Kommunikationspartner kann den Abbau einer Verbindung initiieren. In diesem Fall beginnt A mit dem Abbau indem er ein Paket mit FIN-Flag sendet (1). B bestätigt den Empfang durch ein Paket mit ACK-Flag (2). Ab diesem Zeitpunkt kann A keine Daten mehr senden. Theoretisch kann B noch Daten senden, die von A empfangen werden, in der Praxis wird aber sofort der Verbindungsabbau von B beendet: B sendet ebenfalls ein FIN-Paket (3) und A bestätigt dieses (4). Damit ist die Verbindung beendet und es können keine Daten mehr ausgetauscht werden bis eine neue Verbindung aufgebaut wird.

3.2.2 TCP Resets

Das RST-Flag von TCP hat zwei Anwendungen: die Ablehnung einer Verbindung und das Signalisieren einer unterbrochenen Verbindung. Beide sind in Abbildung 3.5 zu sehen.

Sollte ein Client eine Verbindung zu einem Server-Port aufbauen wollen (1), auf dem keine Verbindung möglich ist, dann lehnt der Server die Verbindung

mit einem Paket ab, bei dem RST gesetzt ist.

Sollte eine Verbindung nur bei einem der Kommunikationspartner unterbrochen worden sein (Übermittlung von FIN war nicht möglich oder der Rechner wurde neu gebootet), dann wird das nächste Datenpaket auf dieser Verbindung mit einem RST-Paket beantwortet.

In beiden Fällen führt der Empfang eines RST-Paketes zum sofortigen Abbruch der Verbindung.

Kapitel 4

Überblick Internet-Standard-Protokolle

Dieses Kapitel listet nur einige Applikations-Protokolle exemplarisch und recht unvollständig auf. Es ist unmöglich die vielen tausend existierenden Protokolle in aller Ausführlichkeit zu behandeln.

Dieses Kapitel soll nur einen Eindruck der Möglichkeiten geben.

4.1 DNS

Das Domain Name System ist eine global verteilte hierarchische Datenbank, die Informationen zu im Netz stehenden Rechnern dem Namen dieser Rechner zuordnen kann. Die Hauptanwendung von DNS ist die Übersetzung von Hostnamen (z.B. www.heise.de) in IP-Adressen (z.B. 193.99.144.85).

Eine Anfrage nach Informationen wird bei DNS immer zunächst an den für den fragenden Rechner zuständigen DNS Server geleitet. Dies kann ein lokal fest konfigurierter DNS-Server oder ein vom Provider zugewiesener Server sein. Kann dieser die Anfrage beantworten, weil er für diesen Netzabschnitt zuständig ist oder die Antwort noch im Cache hat, dann wird er die Antwort sofort zurückgeben. Kann er die Frage nicht beantworten, so wird er die Frage ein seinen übergeordneten Server weiterleiten usw.

Es gibt einige lokale Alternativen zu DNS: traditionell besitzen alle TCP/IP-fähigen Rechner ein Hosts-File, in dem die direkte Zuordnung zwischen IP-Adresse und Name gelistet ist. Dies ist bereits für mittlere Firmennetze nicht mehr praktikabel. Auf Unix-Systemen kann dieses Hosts-File zentral per NIS/YP verwaltet und automatisch an alle Rechner im Netz verteilt werden. Windows besitzt einen eigenen Name-Service (WINS), der Windows-Netzwerknamen auf IP-Adressen zuordnen kann. All diese Alternativen skalieren nicht über mittelgrosse Netze hinaus.

4.2 Telnet

Telnet ist ein einfaches Terminal-Protokoll, dass eine Shell eines entfernten Rechners auf einem lokalen Terminal anzeigen kann. Da dies das erste derartige Pro-

tokoll war funktioniert es vollkommen unverschlüsselt und benutzt lediglich eine einfach TCP-Verbindung, um Terminal-Daten direkt zu transportieren – für die korrekte Interpretation der Steuersequenzen ist das lokale Terminalprogramm (bei Windows häufig in den Telnet-Client integriert) zuständig.

Es gibt inzwischen bessere Alternativen (z.B. SSH) für die Terminal-Arbeit, so dass der Telnet-Client fast nur noch für Diagnosen (Test von TCP-Verbindungen) benutzt wird.

4.3 SSH

Die Secure SHell ist ein verschlüsseltes Terminalprotokoll. Die Interpretation der Steuersequenzen obliegt nach wie vor dem lokalen Terminal, SSH bietet jedoch neben der eigentlichen Terminal-Verbindung einige zusätzliche Features, wie alternative Authentifikationsprotokolle (public key, Kerberos, etc.pp.), Port- und X11-Tunneling, sowie Wrapper-Protokolle, wie scp (secure remote copy) und sftp (FTP-ähnliches Protokoll über SSH abgesichert).

4.4 HTTP

Das HyperText Transfer Protocol ist das Protokoll mit dem das World-Wide-Web und inzwischen auch einige andere Dienste funktionieren.

Die erste Version von HTTP (0.9) erlaubte nur einen Request pro Verbindung, transportierte keine zusätzlichen Headerdaten und kannte nur die GET-Methode, mit der sich eine Seite vom Server holen läSSt.

Typischer Ablauf einer HTTP 0.9 Kommunikation (C> ist Client, S> ist Server):

```
C> GET /mypage.html
S> <html>
S> <head>
S> <title>Meine Seite</title>
S> </head>
S> <body>
S> <h1>Meine Seite</h1>
S> Hier steht nix!
S> </body>
S> </html>
```

Mit Version 1.0 des Protokolls kamen zusätzliche Header, zusätzliche Methoden und optional mehrere Requests pro Verbindung. Version 1.1 nahm einige wenige Verbesserungen an 1.0 vor und machte mehrere Requests pro Verbindung zum Standard.

Typischer Ablauf einer HTTP 1.0 Kommunikation:

```
C> GET http://myhost/mypage.html HTTP/1.0
C> Connection: close
C>
S> HTTP/1.1 200 Ok
S> Date: Sat, 23 Dec 2006 16:14:27 GMT
```

```
S> Server: Apache/2.0.55
S> Content-Length: 109
S> Connection: close
S> Content-Type: text/html; charset=iso-8859-1
S>
S> <html>
S> <head>
S> <title>Meine Seite</title>
S> </head>
S> <body>
S> <h1>Meine Seite</h1>
S> Hier steht nix!
S> </body>
S> </html>
```

4.5 SSL/TLS

SSL – Secure Socket Layer – wurde von Netscape entwickelt, um abgesicherte HTTP-Verbindungen aufbauen zu können. Z.B. für Bankgeschäfte oder online-Einkäufe. Version 3 von SSL wurde von der IETF[1] als TLS 1.0 standardisiert. TLS steht für Transport Layer Security und wird heute von sehr vielen TCP-basierten Protokollen zur Absicherung der Verbindung genutzt.

SSL/TLS ziehen eine zusätzliche kryptographische Schicht zwischen TCP und die Applikation. Es benutzt die recht komplexen X.509 Zertifikate zur Authentifikation der Kommunikationsteilnehmer.

Es gibt im Wesentlichen zwei Varianten, wie TLS heute eingesetzt wird: einige Protokolle reservieren einen eigenen Port für gesicherte Verbindungen (HTTPS, POP3S), andere schalten im laufenden Betrieb per Kommando auf TLS um (IMAP4, SMTP).

4.6 SMTP

Das Simple Mail Transfer Protocol gehört zu den ältesten Protokollen des Internet. Es sorgt für den Transport von eMail von einem Host zum nächsten. Es ist nur für den Versand von Mail geeignet, für das Abrufen gespeicherter Mail existieren andere Protokolle, wie POP3 oder IMAP.

Neuere Versionen von SMTP bieten die Möglichkeit der Nutzerauthentifikation und der Absicherung der Verbindung via TLS. Diese Features haben jedoch keinen Einfluss auf die transportierte Mail, in den meisten Implementationen, die diese Features anbieten, sorgen sie lediglich dafür dass bestimmte Mails überhaupt akzeptiert werden.

Ein typischer SMTP Dialog sieht so aus:

```
S> 220 zaphod.local ESMTP Exim 4.50 Sat, 23 Dec 2006 17:26:45 +0100
C> HELO localhost
S> 250 zaphod.local Hello konrad at localhost [127.0.0.1]
C> MAIL from:konrad@localhost
```

[1] Internet Engineering Task Force

```
S> 250 OK
C> RCPT to:konrad@localhost
S> 250 Accepted
C> DATA
S> 354 Enter message, ending with "." on a line by itself
C> Subject: so ein Quark!
C> From: Alter Ego <auch-konrad@localhost>
C> To: Ich Selba <konrad@localhost>
C>
C> Deine Beispiele sind echt schwach!
C> Mach mal was lustiges!
C> .
S> 250 OK id=1Gy9id-0008PE-NG
C> QUIT
S> 221 zaphod.local closing connection
```

4.7 POP3, IMAP

POP3 – Post Office Protocol version 3 – ist ein recht einfaches Protokoll zum Abholen von auf einem Server gespeicherter eMail. POP3 ist an sich unverschlüsselt, kann jedoch über SSL/TLS getunnelt werden (und nennt sich dann POP3/S).

Da POP3 nur ein einfaches Postfach kennt ist die übliche Vorgehensweise von Mail-Clients, die Mail vom Server abzuholen und dann dort zu löschen. Aufgrund dieser Beschränkung ist POP3 heute weitgehend von IMAP abgelöst

IMAP (insb. IMAP4) ist ein moderneres Postfach-Protokoll. IMAP kann eine beliebig tiefe Hierarchie von Post-Ordnern auf dem Server verwalten. Der Client kann Mails aus diesen Ordnern abholen, sie in andere Ordner verschieben, löschen, modifizieren oder neue Mails anlegen. Die Mails und Ordner bleiben dabei i.d.R. auf dem Server liegen. Mit Hilfe von Sieve-Scripten kann ein IMAP-Server auch ankommende Mails vorfiltern und in vordefinierte Ordner verschieben.

Eine typische POP3-Konversation ist folgend dargestellt. IMAP ist ein komplexeres Protokoll, bietet aber ähnliche ASCII-Befehle.

```
S> +OK myhost Cyrus POP3 server ready
C> USER konrad
S> +OK Name is a valid mailbox
C> PASS secret
S> +OK Maildrop locked and ready
C> LIST
S> +OK scan listing follows
S> 1 4743
S> 2 2562
S> 3 2890
S> 4 1713
S> 5 4809
S> 6 4308
S> 7 4990
S> 8 6050346
```

```
S> 9 9983
S> 10 3350
S> 11 15572
S> 12 955
S> .
C> RETR 12
S> +OK Message follows
S> Return-Path: <cyrus@p15139323.pureserver.info>
S> Received: from p15139323.pureserver.info ([unix socket])
S>         by p15139323.pureserver.info (Cyrus v2.1.17-IPv6-Debian-2.1.17-1)
S>      with LMTP; Sat, 23 Dec 2006 17:41:48 +0100
S> X-Sieve: CMU Sieve 2.2
S> Return-path: <konrad@localhost>
S> Received: from localhost ([127.0.0.1] ident=konrad)
S>         by p15139323.pureserver.info with smtp (Exim 3.35 #1 (Debian))
S>         id 1Gy9w7-0005Aa-00
S>         for <konrad@localhost>; Sat, 23 Dec 2006 17:41:45 +0100
S> Subject: so ein Quark!
S> From: Alter Ego <auch-konrad@localhost>
S> To: Ich Selba <konrad@localhost>
S> Message-Id: <E1Gy9w7-0005Aa-00@p15139323.pureserver.info>
S> Date: Sat, 23 Dec 2006 17:41:45 +0100
S> X-Spam-Checker-Version: SpamAssassin 3.0.3 (2005-04-27) on
S>         p15139323.pureserver.info
S> X-Spam-Level:
S> X-Spam-Status: No, score=-1.1 required=3.0 tests=ALL_TRUSTED,BAYES_50,
S>         TO_MALFORMED autolearn=ham version=3.0.3
S>
S> Deine Beispiele sind echt schwach!
S> Mach mal was lustiges!
S> .
C> QUIT
S> +OK
```

Kapitel 5

Programmierung

Auf Systemebene hat sich heute weitgehend die aus Berkeley stammende Socket-API für die Netzwerkprogrammierung durchgesetzt. Dieses Kapitel beschreib einige der Grundlagen zur Programmierung mit Sockets unter Unix/Linux. Zu den Unterschieden der WinSock-API wird am Ende noch etwas gesagt.

Dieses Kapitel kann nur als Startpunkt dienen, sollte der Leser in die Verlegenheit geraten Sockets direkt einsetzen zu müssen, dann sei er auf die reichlich vorhandene Literatur zum Thema hingewiesen, die Feinheiten wie Socket-Optionen und asynchrone Kommunikation ausführlicher beschreibt.

Nahezu jede höhere Klassenbibliothek bringt allerdings heute einfacher zu nutzende Klassen/Funktionen mit (so z.B. Java oder Qt).

5.1 Hostnamen

Netzwerkprogrammen werden üblicherweise Hostnamen und keine IP-Adressen übergeben, wir müssen also zunächst den Namen auflösen. Dazu bieten Unixoide Systeme die Funktion `gethostbyname` bzw. `gethostbyname2`.

```
#include <netdb.h>

struct hostent*hent;
hent=gethostbyname(?localhost?);
if(hent==0)
  exit(1);
if(hent->h_addrtype!=AF_INET)
  exit(1);
/* hent->h_addr_list ...*/
```

Die hier benutzte Struktur `hostent` enthält drei für uns interessante Elemente: `h_addrtype` enthält ob es sich um eine IPv4 (AF_INET) oder IPv6 (AF_INET6) Adresse handelt. Auf einem normal konfigurierten System wird das immer IPv4 sein, da für IPv6 inzwischen andere Funktionen existieren. Das Element `h_length` wird in diesem Fall immer den Wert "4" enthalten, der signalisiert, dass eine 32bit-Adresse verwendet wird. Schliesslich das Element `h_addr_list` enthält eine Null-terminierte Liste aller zu diesem Namen gefun-

denen Adressen (das Alias `h_addr` kann genutzt werden, um nur auf das erste Element zuzugreifen).

```
struct hostent {
      char    *h_name;        /* official name of host */
      char    **h_aliases;    /* alias list */
      int     h_addrtype;     /* host address type */
      int     h_length;       /* length of address */
      char    **h_addr_list;  /* list of addresses */
}
#define h_addr h_addr_list[0] /* for backward compatibility */
```

Für IPv6 kompatible Programme sollte die Funktion `getaddrinfo` eingesetzt werden:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

struct addrinfo hints;
struct addrinfo *result;
int ge;
/*retrieve IP from host name*/
memset(&hints,0,sizeof(hints));
hints.ai_family=PF_INET6;
ge=getaddrinfo(hostname,0,&hints,&result);
if(ge!=0){
  fprintf(stderr,"Unable to resolve address: %s.\n",gai_strerror(ge));
  return 1;
}
if(result==0){
  fprintf(stderr,"Name resolution returned NULL result.\n");
  return 1;
}
if(result->ai_addr==0){
  fprintf(stderr,"Name resolution did not return an address.\n");
  return 1;
}
if(result->ai_family!=PF_INET6){
  fprintf(stderr,"Name resulution returned address info of "
    "non-inet6 type %i.\n",result->ai_family);
  return 1;
}
```

5.2 Sockets nutzen

Eine Socket kann sehr einfach über den Aufruf `socket` erzeugt werden. Damit besitzt man eine Socket, kann aber noch nichts damit anfangen. Ein TCP-Client-Programm muss noch mit `connect` eine Verbindung zu einem Server herstellen, ein Server muss seine Socket noch mit `bind` an einen Port binden und mit

`listen` in den Server-Mode versetzen. Bei UDP machen beide Seiten ein `bind` und können dann bereits senden und empfangen.

Die Client Seite würde demzufolge etwa so aussehen:

```
#include <sys/types.h>
#include <sys/socket.h>

struct sockaddr_in sa;
/*create socket*/
sock=socket(PF_INET,SOCK_STREAM,0);
if(sock<0){
  fprintf(stderr,"Failed to retrieve socket: %s\n",strerror(errno));
  return 1;
}
/*connect it to host/port*/
sa.sin_family=AF_INET;
memcpy(&sa.sin_addr,hent->h_addr_list[0],hent->h_length);
sa.sin_port=htons(port);
if(connect(sock,&sa,sizeof(sa))!=0){
  fprintf(stderr,"Unable to connect: %s.\n",strerror(errno));
  return 1;
}
```

Mit der Konstante PF_INET wird dem socket-Aufruf gesagt, dass es sich um IPv4 handeln soll. Dementsprechend würde PF_INET6 für IPv6 benutzt. SOCK_STREAM sagt ihm, dass es TCP sein soll. Connect wird eine Struktur vom Typ sockaddr_in[1] übergeben, die die Daten (IP, Port) des Zielrechners enthält, connect kehrt zurück, sobald die Verbindung besteht oder klar ist, dass sie nicht zustande kommt.

Die Serverseite muss dagegen per bind und listen eine Server-Socket aufbauen:

```
/*create socket*/
sock=socket(PF_INET,SOCK_STREAM,0);
if(sock<0){
  fprintf(stderr,"Failed to retrieve socket: %s\n",strerror(errno));
  return 1;
}
/*make sure we can bind (re-use the port even if we just closed the program using it)*/
val=1;
setsockopt(sock,SOL_SOCKET,SO_REUSEADDR,&val,sizeof(val));
/*bind it to port*/
sa.sin_family=AF_INET;
sa.sin_addr.s_addr=htonl(INADDR_ANY);
sa.sin_port=htons(port);
if(bind(sock,&sa,sizeof(sa))){
  fprintf(stderr,"Failed to bind socket: %s\n",strerror(errno));
  return 1;
}
```

[1] sockaddr_in6 für IPv6

```
/*make it listen: server socket*/
if(listen(sock,5)){
  fprintf(stderr,"Cannot turn socket to listen: %s\n",strerror(errno));
  return 1;
}
```

In diesem Fall enthält die an bind übergebene Struktur die gewünschten Daten des lokalen Servers, der gerade erzeugt werden soll. Der Aufruf `listen` versetzt diese Socket dann nur noch in den Server-Modus in dem sie Verbindungsanfragen von Clients entgegen nehmen kann.

Ports, die benutzt wurden sind normalerweise für eine Weile nach ihrer Benutzung gesperrt, damit nicht versehentlich Pakete älterer Verbindungen eine neue Verbindung stören. Der Aufruf von `setsockopt` sagt in diesem Fall dem Betriebssystem, dass die Wiederverwendung des Ports Absicht ist und zugelassen werden soll, was bei Server-Sockets unkritisch ist.

Eine so aufgebaute Server-Socket kann allerdings noch nicht zur Kommunikation genutzt werden. Sie kann lediglich neue Verbindungen annehmen. Für jede angenommene Verbindung muss eine neue Socket erzeugt werden, die dann genutzt werden kann. Dies passiert mit dem Aufruf `accept`, der solange blockiert, bis eine neue Verbindung anliegt und dann eine gültige Socket zur Kommunikation zurückgibt:

```
/*endless loop: wait for new connections*/
for(;;){
  int hdl=accept(sock,0,0);
  if(hdl<0){
    fprintf(stderr,"Ooops. Unable to accept: %s\n",strerror(errno));
    continue;
  }
  do_some_communication(hdl);
}
```

Beide Seiten benutzen grundsätzlich die selben Funktionen zur Kommunikation. Dies sind `read` zum Empfangen von Daten und `write` zum Senden von Daten. Ist das Ende der Kommunikation erreicht gibt `read` den Wert 0 zurück, ansonsten die Anzahl der gelesenen Bytes. Mit `close` oder `shutdown` kann die Socket schliesslich geschlossen und die Kommunikation beendet werden.

5.3 Asynchrone Sockets

Es ist möglich Sockets mit Hilfe des Systemaufrufs `fcntl`[2] in den asynchronen Modus zu versetzen. Danach kann `select` benutzt werden, um auf Ereignisse zu warten. Die `connect` und `accept` Aufrufe benutzen das "READ" Event um auf sich aufmerksam zu machen.

5.4 WinSock

Während WinSock 1.1 noch eine direkte Kopie der BSD-Socket-API war, hat sich bei WinSock 2.0 einiges geändert, um die API näher an den unter Windows

[2]Kommandos GET_FL/SET_FL und Flag O_NONBLOCK.

üblichen Modus zu bringen. Einige der wichtigsten Unterschiede sind:

WSAGetLastError muss aufgerufen werden anstatt errno direkt zu verwenden (Microsoft begründet dies mit der angeblich Thread-Unsicherheit von errno).

WSA_* Fehlercodes anstatt der von Unix gewohnet E* Codes.

send und recv werden statt write und read verwendet, da sich Windows-Subsysteme die Betriebssystem-Aufrufe nicht teilen können.

Windows-Callbacks müssen registriert und verwendet werden um asynchron zu kommunizieren.

Kapitel 6

IP Version 6

Die Entwicklung einer neuen IP Version wurde hauptsächlich durch die Adressknappheit des heutigen Internet vorangetrieben. Weitere Teilnehmer an der Standardisierung haben andere interessante Features eingebracht:

Längere Adressen von 128 Bit bieten mehr Flexibilität bei der Adressvergabe und dem Aufbau von Netzwerken.

Flow Control ermöglicht gleichmäSSigeres Routing von Datenströmen (wie Video).

Mobile Netzknoten können bei IPv6 von einem Netzsegment zum nächsten weitergereicht werden.

Autoconfiguration ermöglicht es dass ein IPv6-Rechner sich problemlos in ein beliebiges Netzsegment einklinken kann.

6.1 IPv6 Header

Abbildung 6.1 zeigt den schematischen Aufbau eines IPv6-Headers.

version ist immer auf "6" gesetzt.

Traffic class kann später genutzt werden, um Routerverhalten zu beeinflussen

Flow Label kann benutzt werden, um zusammengehörige Pakete zu markieren

Payload Length Länge des Paketes in Bytes minus 40 byte IP-Header

Source/Destination Address 128 bit Ursprungs-/Ziel-Adresse des Paketes

Es fällt auf, dass das von IPv4 bekannte Checksummenfeld fehlt. Mit modernen Gigabit-Netzen ist die Checksummen-Korrektur bei jedem Hop zu aufwändig geworden, um noch mit der Kommunikationsgeschwindigkeit mitzuhalten. TCP/UDP Checksummen schlieSSen jetzt einige IP-Felder mit ein (z.B. source/destination address), die sich während des Transports nicht ändern.

Es gibt verschiedene Optionen, die an den IPv6-Header angehängt werden können:

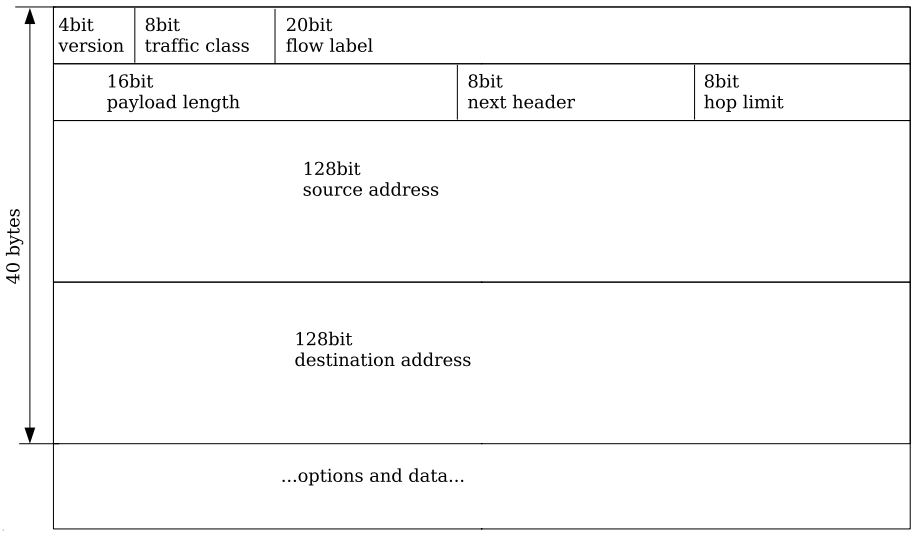

Abbildung 6.1: IPv6 Header

Hop-by-hop Options Aktionen, die jeder Router durchführen muss

Destination Options Aktionen, die der Zielhost durchführt

Routing vorgegebene Route des Pakets, hier kann der Sende-Host festlegen, welche Punkte im Netz das Paket passieren muss.

Fragment-Header markieren Pakete, die zerteilt werden können oder bereits zerteilt sind. Der Zielrechner setzt diese dann wieder zusammen. Anders als bei IPv4 muss ein Paket bei IPv6 explizit markiert sein, damit es fragmentiert werden kann.

Authentication (AH) IP/Sec[1]: identifiziert den Absender

Encapsulating Security Payload IP/Sec: verschlüsselte Daten

6.2 Adressen und Masken

IPv6 verwendet 128 Bit (16 bytes) lange Adressen, die in Gruppen zu jeweils 16 Bit durch ":" getrennt dargestellt werden. Dabei darf die längste Gruppe von 16-Bit-Nullen als "::" abgekürzt werden. Die Netzmaske wird grundsätzlich mit "/" abgetrennt als Anzahl der Bits angegeben, die zum Subnetz gehören. Beispiele:

fd4d:849e:34dd:1:2e0:81ff:fe2e:b6d1/64 ist eine Site-Local Adresse mit einem Netzwerk-Anteil der ersten 64 Bit und einer Interface-ID von 64 Bit (die letzten 64 Bit).

fe80::2e0:81ff:fe2e:b6d1/64 ist eine Link-Local Adresse mit ebenfalls 64 Bit Netzwerkmaske. In diesem Fall befinden sich zwischen "fe80" und "2e0" nur Nullen, die durch "::" abgekürzt werden.

[1] IP/Sec ist ein standardisiertes VPN-Protokoll

::1/128 ist die localhost IP-Adresse von IPv6, bei der alle 128 Bit fest vorgegeben sind.

IPv6 definiert sogenannte Adress-Scopes, die angeben wo eine Adresse sichtbar ist:

Interface Local gilt nur im Netzwerkinterface selbst. Pakete mit diesen Adressen werden nie aus dem Rechner herausgesendet.

Link Local gilt nur im lokalen Netzsegment. Diese Adresse werden i.d.R. nur zur Autoconfiguration benutzt, für den normalen Betrieb werden site local oder global Adressen benutzt.

Site Local gilt für eine Organisation und wird nicht über diese hinaus gesendet. Diese Adressen entsprechen grob den privaten Adressen von IPv4.

Global gilt weltweit und kann im gesamten Internet zur Kommunikation eingesetzt werden. Diese Adressen müssen (evtl. über einen Provider) bei der IANA beantragt werden.

Es gibt im Moment einige definierte Adressbereiche, die eingesetzt werden:

:: (Alle Bits sind 0) Any, ungültige Adresse. Diese wird von Implementationen genutzt, um eine Socket an alle Interfaces zu binden.

::1/128 Loopback. Pakete an diese Adresse kommen an den Sende-Host selbst zurück.

fe80::/64 Link-Local-Adressen, die genutzt werden, um die eigentliche IP-Adresse des Interface vom Router zu erfragen.

ff00::/8 Link-Local Multicast (RFC4489)

2000::/3 aktuelle öffentliche IPs, die bei der IANA oder lokalen Registries (RIRs) beantragt werden können

fc00::/7 site local unicast (RFC4193), die in eigenen Netzen benutzt werden können.

Multicasts sind Adressen mit mehreren Empfängern. Diese können benutzt werden, um die lokale Netzwerk-Umgebung zu inspizieren oder Live-Streams zu verbreiten. Einige der vordefinerten lokalen Multicast-Adressden sind diese:

ff01::/96 Node Local Multicast

ff02::/96 Link Local Multicast:

- **ff02::1** All Nodes, ein ping an diese Adresse bringt eine Antwort alle angeschlossenen Rechner zurück.
- **ff02::2** All Routers, ein ping an diese Adresse bringt eine Antwort aller an den Link angeschlossenen Router zurück (normalerweise sollte das nur einer sein).

ff05::/96 Site Local Multicast

6.3 Autoconfiguration

Bei IPv6 sind die Router für die Konfiguration ihres Netzwerksegmentes zuständig. Jeder Router muss (schon um seine Funktion zu erfüllen) wissen welches Netzwerksegment an welchem Interface anliegt. Wenn ein neuer Client-Rechner an das Netzwerk angeschlossen wird, sendet er automatisch eine Router-Solicitation-Message, die von einem Router mit einem Router-Advertisement beantwortet wird. In diesem Router-Advertisement informiert der Router den Client über die zu benutzende Netzwerk-ID, das ist der Teil der IP-Adresse, der für alle Clients dieses Segments identisch ist. Die IP-Adressen, die bei IPv6 normalerweise an ein End-Segment vergeben werden haben eine 64-Bit-Netzmaske. Die restlichen 64 Bit (EUI-64 genannt) kann jeder Rechner selbst aus der jeweiligen Hardware-Adresse seiner Netzwerkkarte berechnen. Bei Ethernet-MACs funktioniert das so:

1. HW MAC nehmen (z.B. 00:E0:81:2E:B6:D1)
2. Bit 7 flippen (02 E0 81 2E B6 D1)
3. FFFE in die Mitte einschieben (02 E0 81 FFFE 2E B6 D1)
4. in IP einsetzen (fd4d:849e:34dd:1:2e0:81ff:fe2e:b6d1)

Die Link Local Adressen kann ein Host noch vor der Autoconfiguration selbst berechnen, indem er fe80::/64 mit seiner EUI-64 kombiniert (im Beispiel: fe80::2e0:81ff:fe2e:b6d1).

Literaturverzeichnis

[INet] Hafner, Lyon: Die Geschichte des Internet (ISBN 3-932588-59-2)

[Illu] Stevens: TCP/IP Illustrated vol. 1 (ISBN 0-201-63346-9)

[Prog] Stevens: Unix Network Programming vol. 1 (ISBN 0-13-490012-X)

[RFC] IETF RFC Archiv `http://www.ietf.org/rfc`

[lww] Linux – Wegweiser für Netzwerker `http://www.oreilly.de/german/freebooks/linag2/inhalt.htm`

Copyright

©Konrad Rosenbaum, 2006/7

Die Texte, Folien und Programmquellen in diesem Paket sind unter der GNU FDL 1.2 und parallel der GNU GPL 2.0 – oder jeweils neueren Versionen der Lizenzen – geschützt.

Als spezielle Ausnahme von der GNU FDL enthält dieses Werk keine invarianten Sektionen (invariant sections). Ableitungen von diesem Werk dürfen keine invarianten Sektionen hinzufügen. Es ist jedoch erlaubt Teile des Werks oder das Werk als Ganzes zu Dokumenten hinzuzufügen, die bereits invariante Sektionen enthalten, solange am Text selbst vorgenommene Änderungen gleichzeitig unter der GNU FDL ohne invariante Sektionen oder der GNU GPL (oder beiden) freigegeben werden.

Die Quellen zu diesem Script, Programmquellen und Folien können per Subversion heruntergeladen werden:

```
svn co https://silmor.de/svn/misc/tcpip-vortrag/trunk
```

Die Webseite zum Vortrag ist: `http://silmor.de/48`