Kurzeinführung in TCP/IP

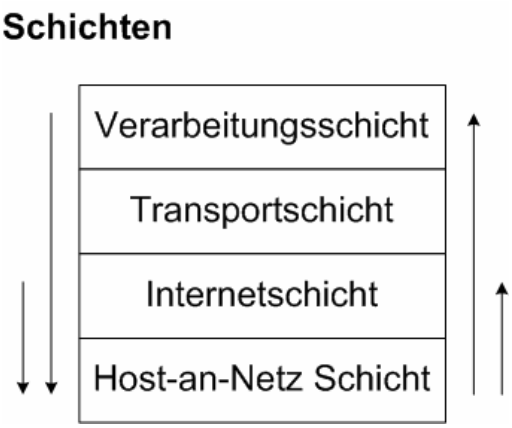
1. Historisches

Während des kalten Krieges 1969/68 wurde das ARPANET entwickelt unter militärischer Leitung entwickelt. Es lief mit wenigen Rechnern und die Kommunikation wurde über das NCP (*Network Control Protocol*) geregelt.
Erst 1972 wurde es der Öffentlichkeit vorgestellt. Damals wurden darin folgende Ziele verfolgt: Kompabilität, Sicherheit, Fehlertoleranz und Unabhängigkeit.
1976 wurde der Grundstein für TCP/IP gelegt um NCP abzulösen.
Bis 1983 erfolgte dann die vollständige Umstellung auf TCP/IP.
Erst später beschäftigte sich die ISO(International Standard Organisation) mit Netzwerken und Standardisierungen. Daher konnte sich auch das ISO-OSI Modell nie gegen TCP/IP durchsetzen, da sich TCP/IP schon international etabliert hatte, praktisch implementiert war und wunderbar funktionierte, wohingegen ISO-OSI nur neue theoretische Richtlinien vorgab.
1982 enstand dann das Internet, welches über einen TCP/IP-Gateway lief.

2. TCP/IP Referenzmodell

Wegen Komplexitäts- und Kompabilitätsgründen, folgte eine Aufgabenverteilung auf mehrere Protokollschichten. Diese Schichten stellen je eine Aufgabe der Netzarchitektur dar.
Beim TCP/IP-Referenzmodell gibt es vier Schichten: Verarbeitungsschicht, Transportschicht, Internetschicht und Host-an-Netz Schicht.

Die Verarbeitungsschicht verkörpert die Anwendungen, die miteinander kommunizieren wollen.
Auf der Transportschicht soll eine transparente *End-to-End* Übertragung stattfinden.
Die Internetschicht hat die Aufgabe, Pakete über jedes beliebige Netz zu schicken und auf einen Weg zum Ziel zu leiten.
Die Host-an-Netz-Schicht ist nicht genauer spezifiziert. Es wird nur festgelegt, dass der Host sich über ein Protokoll am Netz anschließen muss, um IP-Segmente verschicken zu können.
Im folgenden wird auf dieses Referenzmodell aufgebaut.



Vergleiche Quelle [1] Seite 54

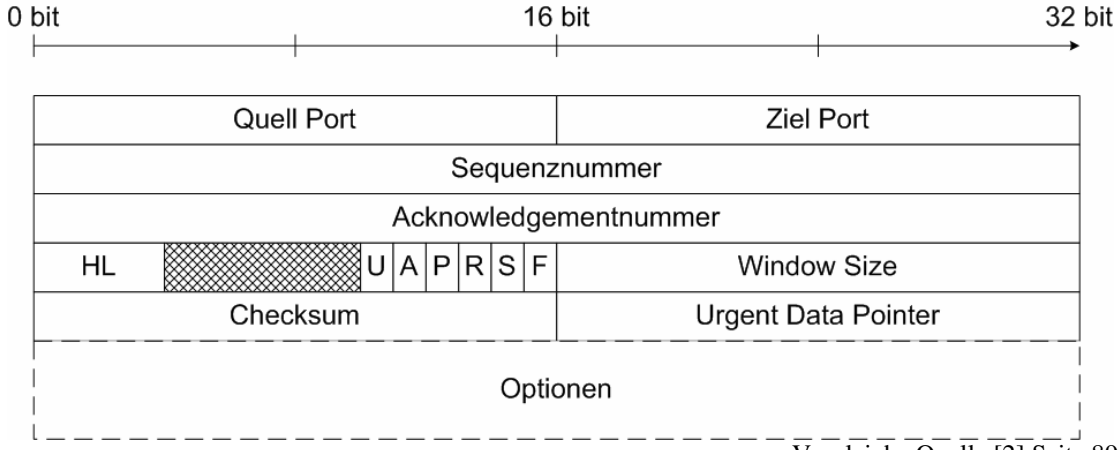
3. Protokolle der Transportschicht

Protokolle der Transportschicht stellen eine transparente end-to-end Übertragung zur Verfügung und bilden eine Schnittstelle für Anwendungen zum Netz.

3.1. TCP

Das *Transmission Control Protocol*, kurz TCP, stellt ein Transportprotokoll dar, das in erster Linie eine sichere Punkt-zu-Punkt Übertragung verfolgt. Dies wird durch einen Verbindungsaufbau gewährleistet. Hiermit wird eine direkte Kommunikation zwischen zwei Anwendungen ermöglicht (z.B. TELNET).

3.1.1. Header



Vergleiche Quelle [2] Seite 89

Quell- und Zielport dienen dem Demultiplexing der Daten zur Anwendung.
Die Sequenznummer gibt an, das wievielte Byte an Nutzdaten übertragen wird.
Die Acknowledgementnummer gibt an, welches Byte als Nächstes erwartet wird.
Die Headerlänge gibt die Länge des Headers in 32 Bit Wörtern an, max. 60 Byte Header.
Die ungenutzten 6 Bit dienen als Reserve, für neue Probleme, unerkannte Fehler.
URG-Flag (urgent) ist gesetzt, wenn es sich um eine dringende Nachricht handelt.
ACK-Flag (acknowledgement) ist gesetzt, wenn es sich um eine Bestätigung handelt.
PSH-Flag (push) ist gesetzt, wenn die Daten sofort der Anwendung bereitgestellt werden sollen (ist z.B. bei Telnet permanent gesetzt).
RST-Flag (reset) ist gesetzt, wenn ein Fehler auftritt.
SYN-Flag (synchronize) und FIN-Flag (final) werden gesetzt, um eine Verbindung aufzubauen oder zu beenden (s.u.).

Window Size gibt an, wie viele Bytes vom Puffer auf Empfängerseite noch frei sind.
Bei einer korrekten Übertragung und Zustellung, ergibt die Berechnung der Prüfsumme Null.
Der *Urgent Data Pointer* verweist auf das Ende von dringenden Daten in einem TCP-Segment. Offset, für das Ende der dringenden Daten, ist Sequenznummer + Urgent Pointer.
Das Optionsfeld kann zur Vereinbarung einer maximalen Segmentgröße dienen, oder ein Skalierfaktor für das *Window Size* enthalten.

3.1.2. Nachrichtenaustausch

Auf der Senderseite, werden die Daten in TCP-Segmenten verschickt. Zu jedem TCP-Segement gehört eine Sequenznummer (SEQ#), und ein Timer, der beim Versenden gestartet wird. Wird eine Acknowledgementnummer (ACK#) empfangen, die größer als SEQ# ist, werden alle SEQ# kleiner ACK# als empfangen markiert, und deren zugehörige Timer gestoppt. Läuft ein Timer aus bevor er gestoppt wurde, wird das Paket nochmal gesendet und der Timer erneut gestartet. Treffen dreimal dieselben ACKs ein, wird das Segment (mit SEQ# = ACK#) sofort gesendet (fast retransmission) und der Timer neugestartet.
Auf der Empfängerseite wartet man 500 ms, bis ein ACK gesendet wird, falls es noch keine unbestätigeten Segmente gibt. Wird ein Segment empfangen, das noch nicht erwartet wird, werden sofort 2 ACKs für das erwartete Segment abgeschickt. Wird eine solche „Lücke" aufgefüllt oder geschlossen, wird sofort ein ACK für das nächste zu erwartende Segment gesendet. Es werden nur Segmente bestätigt, von denen alle vorherigen Segmente eingetroffen sind.

3.1.3. Verbindungsaufbau/Ende

Um eine Verbindung aufzubauen, sendet der Client ein SYN Signal an den Server. Der Server antwortet ebenfalls mit einem SYN. Hierbei werden SEQ# und ACK# intitialisiert und wie oben beschrieben eingesetzt. Im folgenden Graphen steht C für Client, und S für Server

C(SYN)→S(SYN,ACK)→C(ACK, evtl. erste Daten)→…

Falls beide Seiten gleichzeitig eine Verbindung aufbauen und auf ein gesendetes SYN ebenfalls nur ein SYN (ohne ACK) erhalten, verhalten sich beide Seiten wie der Server im letzten Fall. Dies geschieht, um einen Deadlock zu verhindern.

C/S(SYN) →C/S(SYN,ACK)→C/S(ACK, evtl. erste Daten)

Um die Verbindung zu schließen, sendet der Client ein FIN Signal, der Server bestätigt dies, bereitet sich auf das Schließen der Verbindung vor (Puffer leeren, ...) und sendet danach ein FIN Signal an den Client. Dies wird vom Client bestätigt, der nach Ablaufen eines Timers die Verbindung endgültig schließt. Der Server schließt die Verbindung von seiner Seite, nach Erhalt der Bestätigung.

C(FIN)→S(ACK)→S bereitet Ende vor→S(FIN)→C(ACK)→S aus, C nach timeout aus

3.1.4. Flusskontrolle (*Windowing*)

Damit keine Daten verloren gehen, weil der Puffer des Empfängers voll ist, sendet dieser immer die Größe seines freien Puffers mit (*Window Size*).

Der Nagle-Algorithmus soll vermeiden, dass der Sender zu kleine Pakete sendet: alle Daten im Puffer werden erst dann gesendet (dann aber möglichst komplett), wenn alle vorherigen gesendeten Pakete bestätigt wurden. Danach wird der Puffer wieder gefüllt, während darauf gewartet wird, dass alles bestätigt wird.

Wenn *Window Size* Null ist, kann der Sender keine Segmente mehr senden. Mit zwei Ausnahmen: Header ohne Daten (FIN, RST, ACK) und Urgent-Segmente, die meist Interrupts auslösen. Dies soll Verklemmungen vermeiden.
Beim *Silly-Window-Problem* wird auf der Empfängerseite, bei vollem Puffer, 1 Byte im Puffer frei. Darauf wird eine 40 Byte große Fensteraktualisierung gesendet und der Sender sendet ein 41 Byte Segment mit 1 Byte Daten.
Clarks Lösung greift auf der Empfängerseite ein, indem nur dann Fensteraktualisierungen gesendet werden sollen, wenn mindestens die maximale Paketgröße empfangbar ist, oder der halbe Puffer frei ist.

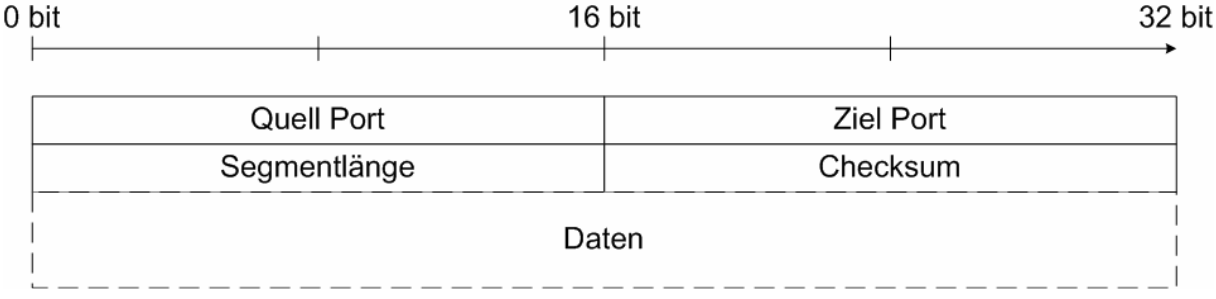
3.1.5. Überlastkontrolle

Um das Netz nicht zu überlasten, sendet der Sender maximal soviel Daten, wie der Wert von *Congestion-Window* (CW) zulässt. CW ist maximal so groß wie *Window Size* (WS). Es wird noch ein Schwellwert (th) benötigt, der mit 64 KB initialisiert wird. CW wird mit der maximalen Paketgröße initialisiert.
Der Sender verdoppelt solange CW, bis ein Timeout, oder 3 doppelte ACKs (3 ACKs mit der selben ACK-Nummer) auftreten. Dies nennt man „*Slowstart*“ (exponentiellen Wachstum). Erreicht CW den Schwellwert, so wächst CW nur noch linear, je um die Größe des max. Segments.
Bei 3 doppelten ACKs wird CW halbiert und th auf CW gesetzt und mit linearem Wachstum fortgefahren.
Bei einem Timeout wird th auf CW halbe gesetzt und CW wird auf den Startwert (i.d.R. maximale Paketgröße) zurückgesetzt und wieder *Slowstart* angewandt.

3.2. UDP

Das *User Datagram Protocol*, UDP, ist im Gegensatz zu TCP verbindungslos und unsicher. Es leistet das Minimale um die Kriterien der Transportschicht zu erfüllen. UDP ist einfacher und schneller. Es muss weniger Speicher belegt werden, um Daten zu empfangen/senden zu können. Dadurch ist UDP für kurze Mitteilungen und Multimedia(livestreams) geeignet. Der Verzicht auf einen Verbindungsaufbau ermöglicht somit auch Broadcasts und Multicasts.

3.1. Header



Vergleiche Quelle [2] Seite 96

Quellport und Zielport erfüllen dieselbe Aufgabe wie bei TCP.
Die Segmentlänge gibt die Gesamtlänge des UDP-Segments (Header und Daten) in Bytes, an.
Die Prüfsumme kann bei UDP ignoriert werden. Dies geschieht, wenn sie auf Null gesetzt ist. Eine Berechnung der Null führt nun dazu, das sie in die größtmögliche Zahl ($111...1_{(2)}$)

überführt wird. Eine korrekte Übertragung und Zustelllung werden also an der maximal größten Zahl erkannt ($65.535_{(10)}$).

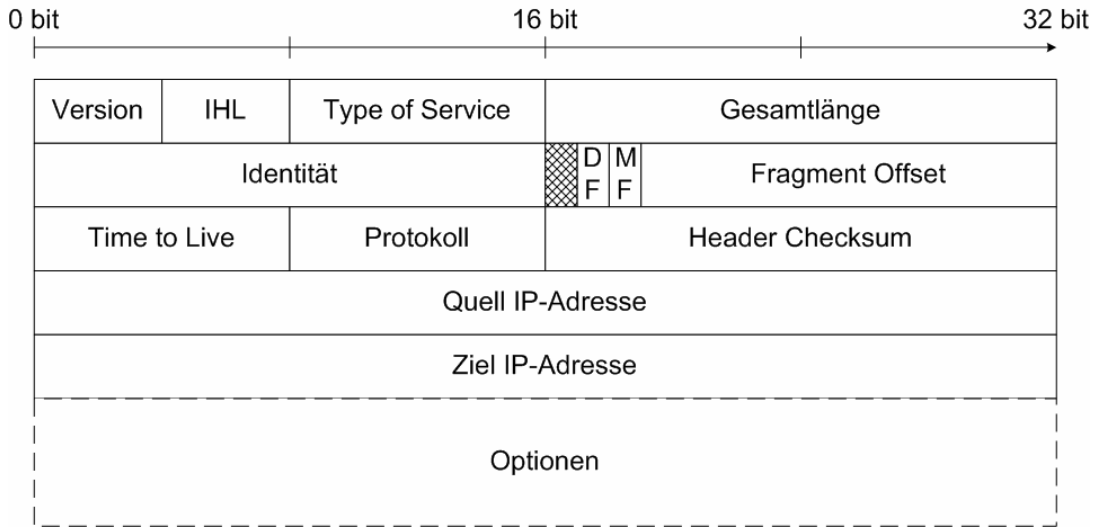
4. Protokolle der Internetschicht

Protokolle der Internetschicht leiten Pakete in und durch das Netz zum Ziel. Sie werden bei jedem Router eingesetzt. Neben den hier angesprochen Protokollen exitsieren noch mehrere Routingprotokolle.

4.1. IPv4

Das Internetprotokoll (IP) Version 4 verfolgt eine Anpassung der Datagramme an das jeweilige Netz durch Fragmentierung und enthält Informaionen zur Zustellung des Pakets (IP-Adresse, nächsthöheres Protokoll).

4.1.1. Header



Vergleiche Quelle [2] Seite 68

IHL steht für *IP Header Length* und gibt die Länge des *Headers* in 32 Bit Worten an.
Das *Type of Service* Feld besteht aus 3 Bits *Precedence*, welche die Priorität des Datagrams angeben sollen. Desweiteren besteht es aus 4 Flags D, T, R, C, die angeben ob auf Verzögerung (**d**elay), Datendurchsatz (**t**hroughput), Zuverlässigkeit (**r**eliability) und Kosten (**c**ost) geachtet werden soll.
Die Gesamtlänge gibt die Größe des ganzen Datagrams mit maximal 65.535 Byte an.
Die Identifikation wird benötigt, um zusammengehörige Fragmente zu erkennen.
Ist das Flag DF (*don't fragment*) gesetzt, darf das Datagramm nicht fragmentiert werden. Ist das MF (*more fragments*) Flag gesetzt, folgen weitere Datagramme mit der selben ID. Beim letzten Datagram eines Fragments ist MF nicht gesetzt.
Der Fragmentoffset gibt an, bei welchem Byte ein Fragment beginnt. Dies wird gemessen mit einem Vielfachen von 8 Byte.
Time-to-Live (TTL) gibt an, wieviele Sekunden ein Datagramm „überleben" soll. Bei längerem Aufenthalt auf Routern und nach jeder Teilstrecke wird TTL gesenkt. Dies soll dafür sorgen, dass fehlgeleitete Pakete nicht ewig herumgeistern.
Das Protokoll-Feld gibt an, an welches Protokoll das Datensegment weitergegeben werden soll.
Mit der *Header Checksum* lässt sich feststellen, ob der *Header* korrekt übermittelt wurde.
Im folgenden möchte ich noch einige offiziellen Optionen ansprechen: Die *Security Option* legt fest, wie geheim ein Datagramm ist und soll so eine Route durch bestimmte Länder vermeiden. Die *Strict-Source-Routing Option* gibt eine genaue Route an. Die *Loose Source*

Routing Option gibt nur einige Punkte an, die durchlaufen werden müssen. *Record Route* zeichnet die verwendete Route auf. Bei *Time Stamp* wird noch zusätzlich die Zeit aufgezeichnet.

4.1.2. IP-Adressierung

Die IP-Adresse spaltet sich in Typ, Netz und *Host-IDs*. Die ersten Bits bestimmen einen eindeutigen Typ. Dieser bestimmt dann, wieviele Bits nur Adressierung des Netzwerkes und des einzelnen *Hosts* zur Verfügung stehen.

	32 Bit		
Typ A	0	7 Bit Netz	24 Bit Host-ID
Typ B	10	14 Bit Netz	16 Bit Host-ID
Typ C	110	21 Bit Netz	8 Bit Host-ID
Typ D	1110	28 Bit Multicastadresse	
Typ E	11110	Für künftige Nutzung reserviert	

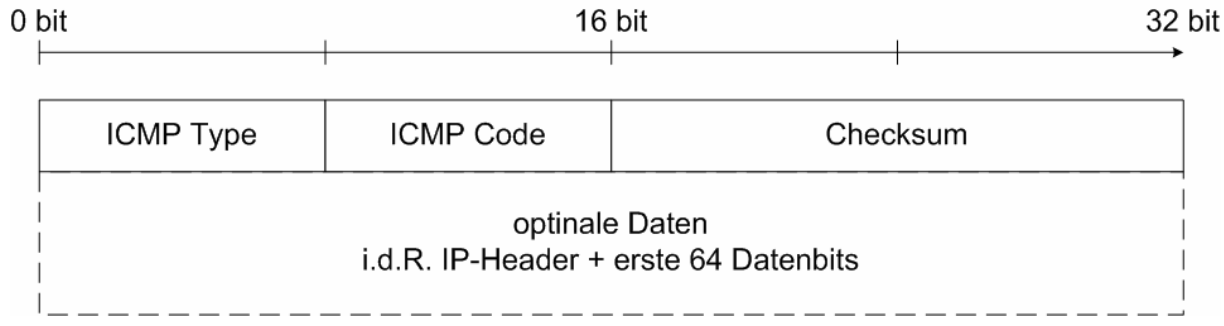
Mit der Zeit wird ein Mangel an IP-Adressen unvermeidlich. Viele Unternehmen wählen ein Klasse B Netz, weil ihnen 256 Hosts in einem Typ C Netz auf Dauer zu knapp sind. Die möglichen 65.535 *Hosts* werden sie aber häufig nicht ausnutzen. Da zusätzlich immer mehr IP-Adressen zu *Routing*-Problemen führen, hat man CIDR, das klassenlose Routing entwickelt. Hierbei wird eine Maske über die IP-Adresse gelegt und so die Netzadresse erkannt. So gehen fürs erste die IP-Adressen nicht aus. Zum besseren *Routing*, hat man außerdem eine Einteilung der Typ C Netze (nicht aller) nach Kontinenten vereinbart.
Folgende IP-Adressen gelten immer:

0.0.0.0	ist *localhost*,
00...0|Host	ist an *Host* im lokalen Netz (z.B. 0.0.0.123, Host = 123),
255.255.255.255	ist *Broadcast* im lokalen Netz,
Netz|11...1	ist *Broadcast* in Netz, (z.B. x.y.z.256, Netz = x.y.z)
127.x.y.z	ist Schleife, wird zu Testzwecken verwendet

4.2. ICMP

Das *Internet Control Message Protocoll* (ICMP) soll das Internet überwachen. Es hilft bei Fehlern Rückschlüsse auf die Ursache zu ziehen und bietet auch Möglichkeiten zum Test eines Netzes (ping).

4.2.1. Header



Vergleiche Quelle [2] Seite 77

ICMP-Type legt einen Meldungsart fest, die durch den ICMP-Code spezifiziert wird.
Die Prüfsumme bezieht sich auf die gesamte ICMP Nachricht.
Das Datenfeld enthält i.d.R. den IP-Header des IP-Datagramms, welches die ICMP-Nachricht verursacht hat, einschließlich dessen ersten 64 Datenbits.

4.2.2. Nachrichten

Folgende Nachrichten sind typisch: *echo request/reply*, *destination host unreachable*, *destination network unknown*, *router advertisement*, TTL *expired*, IP *header bad* und *source quench*. Wird *source quench* an TCP weitergereicht, wird er wie ein *timeout* bei der Überlastkontrolle behandelt.

4.3. ARP/RARP

IP-Adressen sind logische Adressen, die nicht zwangsläufig eindeutig sein müssen und sich ändern können. In der Host-an-Netz Schicht gibt es daher noch feste physikalische Adressen. Diese Adressen sind von Werk aus eingearbeitet und einmalig.
Für die Übersetzung der logischen IP-Adressen in die echten physikalischen Adressen ist das ARP zuständig (*Adress Resolution Protocol*). Wenn Nachrichten von bis dato unbekannten physikalischen Adressen eintreffen, werden sie zusammen mit ihrer IP in einer Tabelle gespeichert. Beim Senden wird dann in dieser Tabelle nachgeschaut, wie die zugehörige physikalische Adresse heißt und zur Übertragung verwendet. Ist die physikalische Adresse unbekannt, so wird diese über einen *Broadcast* im Netz ermittelt. Nur der Host mit der zutreffenden IP antwortet. So wird dann seine physikalische Adresse bekannt.
Die Übersetzung von physikalischer Adresse in IP wird vom RARP (*Reverse Address Resolution Protocol*) übernommen. Hierbei wird ein *Request* ins Netz ausgesendet und ein RARP-Server antwortet. Nachdem eine Antwort angenommen wurde, werden alle anderen ignoriert. Man benötigt RARP, wenn z.B. eine plattenlose Workstation ihre IP nicht kennt.

5. Zusammenfassung

TCP/IP hat sich wegen seiner praktischen Einsetzung und Relevanz gegen theoretisch „bessere“ Standards, wie z.B. ISO-OSI durchgesetzt. Die meisten Protokolle bestehen schon seit 20, 30 Jahren und wurden seitdem kaum geändert. Dies zeigt, wie gut sie funktionieren und konzipiert waren. Perfekt sind sie nicht. Dies zeigt die Entwicklung von IPv6, und die fehlende Überlastkontrolle in UDP, die manchmal zu Problemen führt.

6. Quellen

Bei weiteren Intresse verweise ich auf fogende Quellen.

- [1] Computer Netzwerke: 3.revidierte Auflage, Andrew S. Tanenbaum, Prentice Hall, 1996, ISBN: 3-8272-9568-8
- [2] TCP/IP-Grundlagen: Protokolle und Routing, Gerhard Lienemann, Verlag Heinz Heise 1996, ISBN: 3-88229-070-6
- [3] Computer Networking: A Top-Down Approach Featuring the Internet, James F. Kurose, Keith W. Ross, Addison Wesley Longman, 2001, ISBN: 0-201-47711-4
- [4] TCP/IP: Clearly Explained, 3rd Edition, Pete Loshin, Academic Press, 1999, ISBN: 0-12-455826-7