Study-Guide: Netzwerk-Grundlagen

Die drei folgenden Kapitel gehen auf Netzwerk-Grundlagen ein. Als Unix-System arbeitet Linux grundsätzlich mit TCP/IP, ein fundiertes Wissen über dieses System ist also von Nöten, wenn Linux ans Netz gehängt werden soll. Grundlagen von TCP/IP TCP/IP Konfiguration und Problemlösung Konfiguration von Linux als PPP-Client

Seite:	-= LinuxLernSystem =- (http://www.lpi-test.de)
Kurs:	LPIC-1 [102]
Buch:	Study-Guide: Netzwerk-Grundlagen
Gedruckt von:	André Scholz
Datum:	Dienstag, 1 November 2005, 10:49 Uhr

Inhaltsverzeichnis

1.112 - Netzwerk-Grundlagen

Die drei folgenden Kapitel gehen auf Netzwerk-Grundlagen ein. Als Unix-System arbeitet Linux grundsätzlich mit TCP/IP, ein fundiertes Wissen über dieses System ist also von Nöten, wenn Linux ans Netz gehängt werden soll.

- Grundlagen von TCP/IP
- TCP/IP Konfiguration und Problemlösung
- Konfiguration von Linux als PPP-Client

1.112.1 - Grundlagen von TCP/IP

Beschreibung: Prüfungskandidaten sollten ein richtiges Verständnis der Netzwerk-Grundlagen demonstrieren können. Dieses Lernziel beinhaltet das Verständnis von IP-Adressen, Netzwerkmasken und ihrer Bedeutung (d.h. Bestimmung einer Netzwerk- und Broadcast-Adresse für einen Host auf Grundlage seiner Subnetzmaske in "dotted quad" oder abgekürzter Notation oder die Bestimmung der Netzwerk-Adresse, Broadcast-Adresse und Netzwerkmaske bei gegebener IP-Adresse und Anzahl von Bits). Dies deckt auch das Verstehen der Netzwerkklassen und klassenloser Subnetze (CIDR) und der für private Netzwerke reservierten Adressen ab. Ebenfalls enthalten ist das Verständnis der Funktion und Anwendung der Default Route. Weiters enthalten ist das Verstehen der grundlegenden Internet-Protokolle (IP, ICMP, TCP, UDP) und der gebräuchlicheren TCP- und UDP-Ports (20, 21, 23, 25, 53, 80, 110, 119, 139, 143, 161).

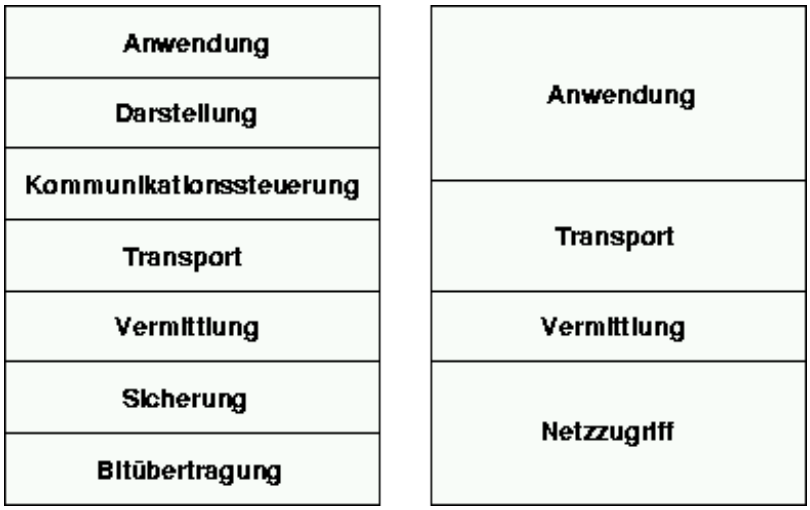
Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- `/etc/services`
- **ftp**
- **telnet**
- **host**
- **ping**
- **dig**
- **traceroute**
- **whois**

Dieses Kapitel ist wenig Linux-spezifisch, sondern bezieht sich auf das Verständnis von TCP/IP im Allgemeinen. Natürlich werden die jeweiligen Linux-Techniken auch mit erwähnt, wo sie anfallen, wichtig ist jedoch das fundierte Verständnis der Technik von IP-Netzwerken, von Netzadressen usw. Ich beginne hier also mit ein paar ausführlichen Abschnitten über diese Themen. Abschließend werden noch die hier aufgeführten Programme besprochen.

Die TCP/IP Protokollfamilie

TCP/IP hält sich nicht an das OSI-Schichtenmodell, es hat ein eigenes Modell, das sich nicht eins zu eins übertragen lässt.



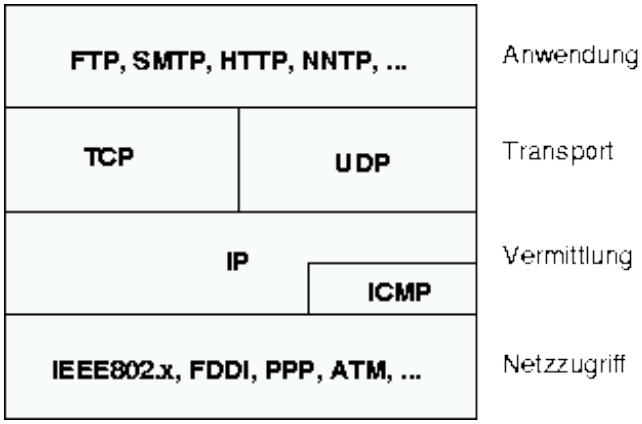
Jede der vier Schichten des TCP/IP Schichtenmodells verfügt über ein bzw. mehrere Protokolle, die den Umgang mit den jeweiligen Daten regeln. Unter Protokollen werden hier Regelsammlungen bezeichnet, die in sogenannten RFCs (Request for comment) veröffentlicht werden.

Das Entscheidende an der Konstruktion von TCP/IP sind die beiden mittleren Schichten, die Transport- und die

Vermittlungsschicht. Hier arbeiten immer die gleichen Protokolle. In der Transportschicht stehen zwei zur Verfügung, eins für verbindungsorientierten Datenverkehr (TCP) und eins für verbindungslosen Datentransfer (UDP). In der Vermittlungsschicht arbeitet nur ein Protokoll in der Datenübertragung (IP), so steht also eine einheitliche und verbindliche Übertragungsform zur Verfügung.

In der obersten und untersten Schicht können so die verschiedensten Protokolle arbeiten, von denen die beiden mittleren Schichten nichts wissen müssen. In der Anwendungsschicht können so Standardprotokolle wie etwa FTP (Dateiübertragung), TELNET (Terminal EmuLation über NET), HTTP (WWW-Seiten), SMTP (E-Mail) oder NNTP (News) arbeiten, aber auch eigens entwickelte Protokolle etwa für ein Schachspiel übers Netz. Jedes dieser Anwendungsprotokolle hat eine Identitätsnummer zugewiesen bekommen, die sogenannte Portnummer. Mit Hilfe dieser Portnummer kann das Transportprotokoll wissen, an welche Anwendung in der Anwendungsschicht es Daten weitergeben soll. Unter Linux ist die Liste aller bekannten Portnummern in der Datei `/etc/services` zu finden.

Die unterste Schicht übernimmt die Steuerung der Netzhardware, also die der Netzwerkkarten o.ä. Diese Schicht kann mit beinahe jedem beliebigen Protokoll arbeiten, wie etwa die IEEE 802.3 802.4 802.5 oder FDDI, ATP aber auch mit PPP oder SLIP auf seriellen oder mit PLIP auf paralelle Schnittstellen.



Um die Funktion des Netzsystems genau zu verstehen ist ein Wissen über die einzelnen Schichten und Protokolle unabdingbar. Anhand einiger Beispiele sollen hier die einzelnen Aufgaben der Schichten erklärt werden.

Die Netzzugriffsschicht mit ihren Protokollen

In dieser Schicht arbeiten, wie oben schon erwähnt, beinahe alle Protokolle von Netzhardware, von einfachen seriellen Verbindungen bis zu Hochleistungssysteme mit Glasfaserkabeln. Ein typisches Beispiel soll hier näher betrachtet werden, das Protokoll IEEE-802.3, das Protokoll des Ethernet. Das hier dargestellte Protokoll ist das der 10Mbit Version von Ethernet, die moderneren Versionen unterscheiden sich in Kleinigkeiten, die aber nicht technisch relevant sind.

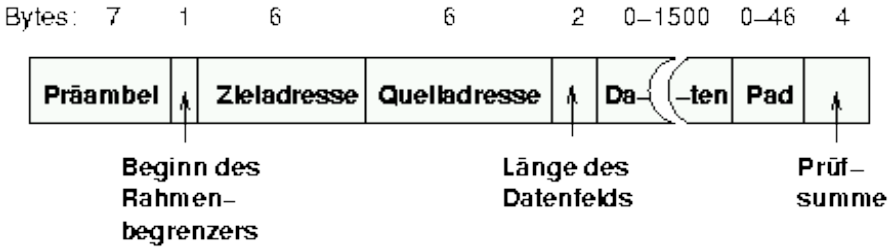
Die Aufgabe des IEEE-802.3 Protokolls ist es, Daten, die von der Vermittlungsschicht kommen, in Pakete zu packen, und diese Pakete dann über die Netzhardware zu verschicken. Umgekehrt nehmen sie Pakete vom Netz entgegen und geben sie gegebenenfalls an die Vermittlungsschicht weiter. Der Inhalt der Daten spielt dabei keinerlei Rolle.

Physikalisch gesehen, ist ein Ethernet eine parallele elektrische Verbindung aller Rechner, die an einem Netzsegment hängen. Das heißt, wenn ein Rechner einem anderen Rechner ein Paket schickt, so bekommen alle angeschlossenen Rechner dieses Paket. Anhand einer (weltweit eindeutigen) Adresse, die jede Netzwerkkarte besitzt, kann dann aber entschieden werden, ob das Paket tatsächlich für diesen Rechner bestimmt ist. Jedes Paket, das über ein IEEE-802.3 Protokoll verschickt wird enthält sowohl diese Adresse des Absenders, als auch die des Empfängers. Alle angeschlossenen Rechner im Netz (genauer gesagt, alle angeschlossenen Netzwerkkarten im Netz) empfangen das gesendete Paket, aber nur die Netzwerkkarte, die tatsächlich die Adresse aufweist, die im Paket als Empfängeradresse angegeben ist, gibt das Paket entpackt an die Vermittlungsschicht weiter. Diese Adressen sind die sogenannten MAC-Adressen.

Die detaillierte Darstellung des IEEE 802.3 Frameformat ist hier einsehbar:

Die interne Struktur des IEEE 802.3 Frames

Der IEEE802.3 Standard (Ethernet) bedient sich auch der Aufteilung des zu transportierenden Datenstromes in Pakete. Diese Pakete werden auch Rahmen (Frames) genannt, weil sie vor und nach den eigentlichen Daten noch Steuerinformationen ablegen. Die folgende Abbildung zeigt das verwendete Rahmenformat:



Jeder Rahmen beginnt mit einer **Präambel** von 7 Bytes, die jeweils die Bitfolge `10101010` enthalten. Durch die Kodierung dieser Bitfolge (Manchester) entsteht eine 10 MHz Schwingung, die 5,6 µs dauert. Dadurch kann sich der Taktgeber des Senders mit dem des Empfängers synchronisieren.

Als nächstes folgt ein Rahmenstartbyte (**Rahmenbegrenzer**) also die Markierung, daß hier der Frame beginnt. Dieses Byte enthält immer die Bitfolge `10101011`

Die beiden folgenden Blöcke, **Zieladresse** und **Quelladresse** enthalten die physikalischen Netzwerkadressen (Ethernetadressen, MAC-Adressen) des Empfängers und Senders. In etwa also das, was ein Briefumschlag enthält, Absender und Empfängerangaben...

Dabei wird intern bei der Zieladresse noch zwischen Einzeladressen, Gruppenadressen und Broadcastadressen unterschieden, also Pakete, die entweder an einen bestimten Rechner gehen, oder an eine bestimmte Gruppe von Rechnern oder an alle Rechner im Netz.

Im nächsten Feld (**Länge**) steht die Angabe, wie lang das folgende Datenpaket ist, gültige Werte sind hier 0 bis 1500. Theoretisch sind also Datenpakete mit der Länge 0 möglich, aus physikalischen Gründen ist aber eine minimale Länge von 46 Bytes erforderlich. Dafür sorgt dann das **Pad** Feld, das Datenfelder mit einer Länge unter 46 Bytes bis auf desen Minimalwert auffüllt.

Im **Datenfeld** zwischen Länge und Pad stehen die eigentlich zu übertragenen Daten. Durch die Angabe der tatsächlichen Länge dieses Feldes im Feld Länge kann der Empfänger des Paketes mit unterschiedlich großen Datenfeldern immer richtig umgehen.

Das letzte Feld des Rahmens ist die **Prüfsumme**, die vom Sender errechnet wurde und sich natürlich auf das Datenfeld bezieht. Der Empfänger kann nun das Datenfeld ebenso überprüfen und, falls er zu einem anderen Ergebnis kommt, eine erneute Zusendung des Pakets beantragen. Dieses Verfahren wird übrigens in der IEEE Norm 802.2 beschrieben, die für alle drei folgenden Normen (802.3, 802.4, 802.5) angewendet wird.

Natürlich gibt es noch viele weitere Protokolle auf der Netztzugriffsschicht, etwa **ppp** und **slip** für die serielle Anbindung an ein IP-Netz, oder andere Hardware-Netzfamilien wie Token Ring. Diese Darstellung ist also nur ein Beispiel von vielen.

Die Vermittlungsschicht und das InternetProtokoll (IP)

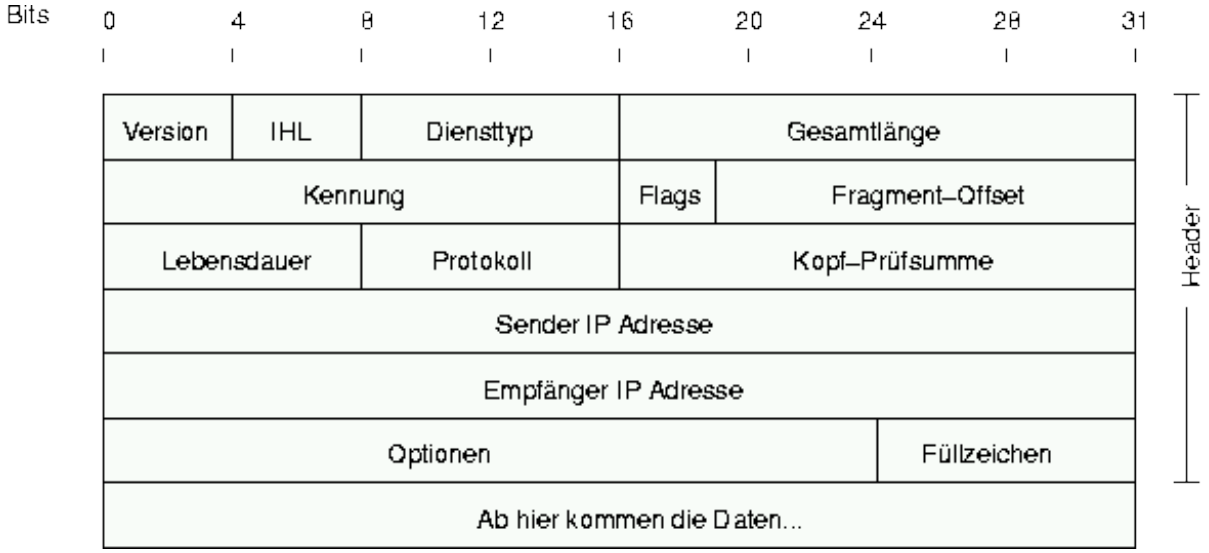
Die Vermittlungs- oder auch Internetschicht hat ein allumfassendes Protokoll, das zum Transport der Daten zuständig ist, das Internet Protokoll (IP). Daneben arbeitet noch ein weiteres wichtiges Protokoll auf dieser Schicht, das für technische Informationen statt für Daten zuständig ist, das Internet Control Message Protokoll (ICMP).

Das Internet Protokoll (IP)

Das Internet-Protokoll ist für die Vermittlung zwischen zwei Rechnern zuständig. Hier wird mit den IP-Adressen gearbeitet, statt mit den physikalischen Adressen im Netzwerk. Die zentrale Aufgabe des IP ist das Versenden und Empfangen von Daten, die für diesen Zweck wieder mal in Pakete aufgeteilt werden müssen. Die Pakete des IP werden Datagramme genannt und beinhalten neben den eigentlichen Daten wieder verschiedenste Steuerinformationen.

Der Aufbau von IP-Datagrammen:

Das Internet Protokoll (IP) packt den Datenstrom, den es aus der Transportschicht bekommt wiederum in Pakete, die aber diesmal mit den logischen IP-Adressen versehen sind statt mit den physikalischen Netzadressen. Die interne Struktur eines solchen Datagramms (Paketes) ist in der folgenden Graphik dargestellt.



Der Aufbau eines IP-Datagramms entspricht also nicht dem Frame System, sondern stellt nur einen Header (Kopfteil) zur Verfügung. Nach dem Header folgen nur noch Daten, keine Frame-Ende-Zeichen o.ä.

Die einzelnen Felder sollen kurz noch dargestellt werden:

Version - 4 Bit
: Dieses Feld enthält die Versionsnummer des IP. Aktuelle Version ist 4, bald kommt 6...

Internet Header Length (IHL) - 4 Bit
: Gibt die Länge des Headers in 32 Bit Worten an. Damit kann ermittelt werden, ob Optionen gesetzt sind. Normalerweise (ohne Optionen) hat dieses Feld den Wert 5.

TOS (Type of Service - Diensttyp) - 8 Bit
: Enthält Flags, die zur Steuerung der Zuverlässigkeit des Netzes dienen. Sie werden automatisch erstellt.

Gesamtlänge - 16 Bit
: Enthält die Gesamtlänge des Datagramms (incl. Kopf) in Byte. Aus der Breite dieses Feldes (16 Bit) ergibt sich so eine maximale Größe von Datagrammen von 65535 Byte. Dieser Wert liegt weit über dem maximalen Wert der einzelnen Netzsysteme, daher reichen 16 Bit aus.

Kennung - 16 Bit
: Jedes IP-Datagramm muß eine eigene Kennung haben, um beim Wiederzusammenbau richtig eingeordnet zu werden.

Flags - 3 Bit
: Flags zur Steuerung der Fragmentierung.

Fragment-Offset - 13 Bit
: Unter Fragmentierung von Datagrammen ist zu verstehen, daß einzelne Datagramme, die durch Gateways in unterschiedliche Netze geroutet werden, dort oft unterschiedlichen Größenbestimmungen unterliegen. IP fragmentiert solche Datagramme und baut sie wieder zusammen. Dieses Feld enthält die Information, an welcher Stelle das Fragment ursprünglich war.

Lebensdauer - 8 Bit
: In Netzen wie dem Internet, in denen viele Datagramme auf unterschiedlchsten Wegen versuchen zum Ziel zu kommen, muß eine maximale Lebensdauer haben, sonst werden sie unendlich oft geroutet. Dieses Feld

enthält eine Zahl, die vom Sender eingetragen wurde. Jeder Gateway muß diese Zahl um eins herunterzählen, wenn er ein Datagramm routet. Ist der Wert Null, so darf kein Router das Datagramm mehr weiterleiten. Übliche Startwerte sind 32 oder 4 (nur lokale Netze setzen so niedrige Lebensdauer an)

Protokoll - 8 Bit

Dieses Feld gibt an, von welchem Protokoll der Transportschicht dieses Datagramm kommt. Das ist wichtig für den Empfänger, weil ja mehrere Netzanwendungen gleichzeitig laufen können. Nur durch diese Information kann das Empfänger IP das Datagramm an das richtige Transportschichtprotokoll weiterleiten. Übliche Werte sind:

- 17 - UDP
- 6 - TCP
- 1 - ICMP

Kopf-Prüfsumme - 16 Bit

Prüfsumme über den Inhalt des Headers, nicht der Daten. Sicherung der Daten wird von der Transportschicht überprüft.

Sender IP-Adresse - 32 Bit

32 Bit IP-Adresse des Senders

Empfänger IP-Adresse - 32 Bit

32 Bit IP-Adresse des Empfängers

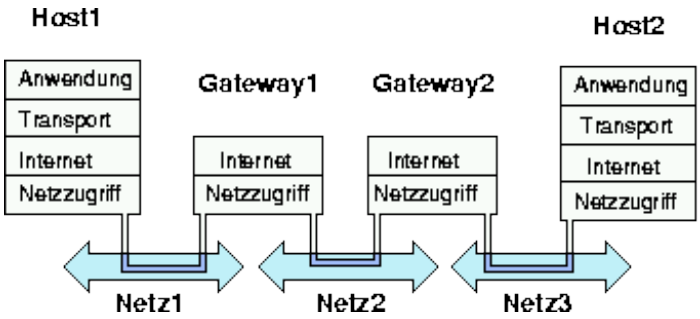
Optionen - <= 32 Bit

Hier können verschiedene Optionen gesetzt werden, die mit Debugging (Fehlersuche) oder Routenprüfung zu tun haben.

Füllzeichen

Füllt den Bereich zwischen den jeweils gesetzten Optionen und dem Ende des 32 Bit Wortes auf.

Wenn Pakete (Datagramme) im Internet verschickt werden oder zumindest eine etwas komplexere Topologie in einem lokalen Netz existiert, so müssen Datagramme oft über Gateways geroutet werden. Das heißt, sie müssen durch verschiedene Rechner hindurch zum Empfänger geleitet werden. Auch diese Aufgabe erfüllt das IP-Protokoll, so daß solche Pakete im Gateway nie über die IP-Schicht hinweg kommen:



Zusammenfassend kann man sagen, das IP hat folgende Aufgaben:

- Es definiert die Datagramm-Struktur, die die Grundlage für allen Datenverkehr im TCP/IP-Netz darstellt.
- Es definiert die IP-Adress-Schematik, von der später noch die Rede sein wird.
- Es bewegt Daten zwischen der Transportschicht und der Netzzugriffsschicht.
- Es routet Datagramme zu externen Rechnern.
- Es teilt Datenströme in Datagramme um sie zu versenden und baut aus empfangenen Datagrammen wieder Datenströme zusammen.

Das Internet Controll Message Protocol

Das *Internet Control Message Protocol* (ICMP) ist ein integraler Bestandteil von IP und dient zur Übermittlung technischer Meldungen, die übers Netz gehen, aber nicht über die IP-Schicht hinauskommen müssen. D.h., daß dieses Protokoll in der Lage ist, Pakete zu schicken, die nicht an ein Transportprotokoll der Transportschicht gebunden sind, sondern eben nur von Vermittlungsschicht zu Vermittlungsschicht gehen.

Die Aufgaben, die dieses Protokoll zu erfüllen hat sind hier kurz aufgelistet:

Flußkontrolle
: Wenn ein Rechner Datagramme so schnell schickt, daß der Empfänger sie nicht rechtzeitig verarbeiten kann, so schickt der Empfänger über ICMP eine Meldung an den Sender, daß die Sendungen vorübergehend stoppen sollen. Nachdem der Empfänger alle anstehenden Datagramme verarbeitet hat, schickt er wieder eine Meldung, daß er wieder empfangsbereit ist.

Erkennen von unerreichbaren Zielrechnern
: Wenn ein Gateway erkennt, daß ein bestimmter Rechner nicht erreichbar ist, so schickt er an den Absender des Paketes eine *Destination unreachable* Meldung über ICMP

Routenoptimierung
: Wenn ein Gateway erkennt, daß er einen Umweg darstellt, so schickt er an den Absender eine Meldung, in der die schnellere Route steht. Der Absender (bzw. seine IP-Schicht) kann dann das nächste Paket schon über den besseren Weg übermitteln.

Überprüfen von erreichbaren Hosts
: Mit Hilfe des *ICMP Echo Message* kann ein Rechner überprüfen ob ein Empfänger ansprechbar ist. Das ping-Kommando nutzt diese Echo-Meldung.

Der Aufbau von IP-Adressen

Das IP-Adressen Format

Jeder Rechner im Netz hat eine (Host) oder mehrere (Gateway) eindeutige IP-Adressen. Streng genommen hat eigentlich jede Netzwerkschnittstelle eine solche Adresse. Dabei handelt es sich um die Adressen, die in den IP-Datagrammen angegeben werden. Aus der Aufteilung des IP-Datagramms geht schon hervor, daß diese Adressen 32 Bit breite Zahlen sind. In der Regel werden diese Zahlen in vier Oktetts dargestellt.

Die hexadezimale IP-Adresse `954C0C04` wird also in seine vier Oktetts (Bytes) aufgeteilt, die durch Punkte getrennt werden. Dadurch entsteht hexadezimal `95.4C.0C.04` oder die bekanntere, dezimale Form: `149.76.12.4`.

IP-Adressen werden zentral vergeben, das heißt, es sollten nicht einfach beliebige Kombinationen verwendet werden. Im Prinzip ist es in reinen lokalen Netzen möglich, beliebige Adressen zu benutzen, sobald ein Anschluß ans Internet besteht, müssen aber zentral vergebene Adressen benutzt werden. Jedes Land der Erde unterhält dazu ein NIC (Network Information Center), das für die Adressvergabe zuständig ist.

Netz- und Hostadressen

Die Aufteilung der IP-Adressen in vier Oktette hat noch einen anderen wesentlichen Grund. Das Internet besteht zum großen Teil nicht aus vernetzten Einzelrechnern, sondern aus Computernetzen, die miteinander über das Internet kommunizieren können. Die IP-Adresse ist immer aufgeteilt in eine Netz- und eine Hostadresse. Diese Eigenschaft spielt für das Routing eine wesentliche Rolle, weil immer davon ausgegangen werden kann, daß IP-Adressen mit der gleichen Netzadresse im lokalen Netz sind.

Je nachdem, welcher Teil der Gesamtadresse als Netz- und welcher als Hostadresse gilt, entstehen unterschiedliche Obergrenzen für die mögliche Anzahl von Netzen bzw. Hosts pro Netz. Weil ja alle Adressen im Internet einmalig sein müssen, kommt es heute bereits zu Engpässen bei der Adressvergabe. Ein neuer Standard (IP6) ist schon in Arbeit, der dann 128 Bit breite Adressen verwenden soll.

Die Frage, welcher Teil der Adresse Netzadresse ist wird durch Adressklassen gelöst. Es existieren fünf Klassen, von denen aber nur drei relevant sind, die anderen beiden dienen der Forschung und Experimentierarbeit. Die Klassen sind:

Netz klasse	Netzadressen breite	Hostadressen breite	Max Anzahl Netze	Max Anzahl Hosts/Netz	Startadresse	Endadresse
A	1	3	126	16777214	1.0.0.0	126.0.0.0
B	2	2	16382	65534	128.0.0.0	191.255.0.0
C	3	1	2097150	254	192.0.0.0	223.255.255.0

Die Adressen von 224.0.0.0 bis 255.255.255.255 sind für zukünftige Verwendungen reserviert, werden aber wohl nie zur Geltung kommen, weil demnächst das IPV6 kommen wird das diese Erweiterungen schon beinhaltet.

Um Konflikte mit lokalen TCP/IP Netzen und dem Internet zu vermeiden, gibt es für jede Klasse einige Adressen, die nicht im Internet geroutet werden. Sie sind ausschließlich für lokale Netze gedacht und sollten auch immer Verwendung finden, wenn es um die Frage der IP-Adressen geht und kein echter Internetanschluß vorliegt. Diese Adressen sind:

Klasse A
10.0.0.0
Klasse B
172.16.0.0 bis 172.31.0.0
Klasse C
192.168.0.0 bis 192.168.255.0

Wo immer lokale Netze ohne direkte Anbindung des ganzen Netzes ans Internet vorliegen, werden Adressen aus diesem Pool verwendet.

Spezielle Adressen

Die Adressen, deren Hostadressenbits alle auf 0 oder 1 gesetzt sind, haben eine Sonderbedeutung. Alle auf Null gesetzt meint das jeweilige Netz selbst, alle auf 1 gesetzt, meint alle angeschlossenen Hosts. Das ist die sogenannte Broadcast-Adresse. So bedeutet die Adresse 149.76.255.255 nicht eine bestimmte Adresse im Netz 149.76.0.0 sondern alle Rechner, die an diesem Netz hängen.

Es gibt auch zwei reservierte Netzadressen mit Sonderbedeutung. Das sind die Adressen 0.0.0.0 und 127.0.0.0 Die erste ist die sogenannte *default route* (voreingestellte Route) und die andere das sogenannte *Loopback*.

Das Netz 127.0.0.0 bedeutet intern im Netz immer das lokale Netz, egal welche reale Adresse es hat. Gewöhnlich steht dann auch noch die Adresse 127.0.0.1 als *loopback interface* des jeweiligen Hosts zur Verfügung. Jedes Paket, das darauf ausgegeben wird kommt sofort wieder an, als käme es aus einem Netz. Damit kann z.B. Software entwickelt werden, für Netze, auf einem Rechner, der nicht an eins angeschlossen ist.

Die Adresse 0.0.0.0 hat eine Bedeutung für das Routing. Jedes Datagramm, das an eine Adresse geschickt wird, zu der keine Route bekannt ist, wird an die *default route* geschickt. Diese voreingestellte Route weist dann zum nächsten Gateway, der schon wissen wird, wohin er das Paket senden soll. Falls er keine Route kennt, die zum entsprechenden Ziel führt, so hat auch er eine default route, an die er das Paket dann schickt...

Subnetze und die Netmask

Ein Teil der Host Adresse kann auch intern als Subnetzadresse gewertet werden um große Netze nochmal zu trennen. So kann also der Host mit der Nummer 12.4 im B--Klasse Netz 149.76 auch uminterpretiert werden, in den Host Nummer 4 im Subnetz 12 des Netzwerks 149.76. Welche Bits der 32 Bit Adresse als Netz- und welche als Hostadresse gewertet werden wird in der sogenannten Netzmaske festgelegt.

Die Netzmaske (Netmask) ist wie die Adresse eine 32 Bit Zahl, die in vier Oktetts aufgeteilt ist. Jedes Bit dieser Maske, das gesetzt ist (1), bestimmt daß das entsprechende Bit in der Adresse ein Teil der Netzadresse ist. Zum Verständnis am Besten mal ein Beispiel:

Die B-Klasse Adresse 149.76.0.0 wird normalerweise gewertet als Adresse, deren erste 2 Bytes als Netzadresse gelten, und deren letzte 2 Bytes als Hostadresse. Die Netmask würde also einfach berechnet:

```
                  Netzadresse       Hostadresse
149.76.0.0 --> 10010101.01001100.00000000.00000000
Netmask        11111111.11111111.00000000.00000000
           --> 255.255.0.0
```

Wollen wir jetzt dieses Netz in mehrere Unternetze aufteilen, so können wir z.B. das dritte Byte als

Unternetzadresse und das letzte Byte als Hostadresse werten. Für die Netmask käme folgendes Bild heraus:

```
                 Netzadresse        Hostadresse
149.76.0.0 --> 10010101.01001100.00000000.00000000
Netmask        11111111.11111111.11111111.00000000
           --> 255.255.255.0
```

Die Netzmaske kann entweder in dieser Form angegeben werden (dotted quad) oder als verinfachte Angabe, indem einer IP-Adresse einfach die Anzahl der Bits angegeben wird, die Netzadressenbits sind. Die folgenden Adressenangaben sind also äquivalent:

Dotted Quad	**Vereinfacht**
192.168.100.1 Netmask 255.255.255.0	192.168.100.1/24
175.12.34.56 Netmask 255.255.0.0	175.12.34.56/16

Schwieriger wird es, wenn wir etwa ein C-Klasse Netz in weitere Unternetze aufteilen wollen. Hier muß dann jede Adresse errechnet werden, je nach der Aufteilung. Die Netmask wäre dann entsprechend auch nicht mehr nur 255 sondern würde sich aus anderen Zahlen zusammensetzen. Nochmal ein Beispiel:

Das C-Klasse Netzwerk 192.168.200.0 soll in vier Unternetze aufgeteilt werden. Um vier Netze anzusprechen benötigen wir zwei Bits der Hostadresse für die Subnetzangabe:

```
                                        Subnetzbits
                                             ||
                         Netzadresse         VV
192.168.200.0 --> 11000000.10101000.11001000.00000000
Netmask           11111111.11111111.11111111.11000000
              --> 255.255.255.192
```

Statt der üblichen 255.255.255.0 haben wir jetzt also 255.255.255.192 als Netzmaskierung (oder - vereinfacht /26 weil die ersten 26 Bits der Adresse jetzt die Netzadresse sind). Damit weiß IP, daß diese Adresse kein normales C-Klasse Netz ist. Allerdings müssen wir jetzt auch bei der Vergabe der einzelnen Rechneradressen aufpassen, daß die verwendeten Zahlen im jeweiligen Netz liegen:

```
Subnetz 0
   Netzadresse   0 -> 00000000
   Hostadressen  1 -> 00000001
                 2 -> 00000010
                 3 -> 00000011
                       ...
   Broadcast    63 -> 00111111

Subnetz 1
   Netzadresse  64 -> 01000000
   Hostadressen 65 -> 01000001
                66 -> 01000010
                67 -> 01000011
                       ...
   Broadcast   127 -> 01111111

Subnetz 2
   Netzadresse 128 -> 10000000
   Hostadresse 129 -> 10000001
               130 -> 10000010
               131 -> 10000011
                       ...
   Broadcast   191 -> 10111111
```

```
Subnetz 3
   Netzadresse 192 -> 11000000
   Hostadresse 193 -> 11000001
               194 -> 11000010
               195 -> 11000011
                        ...
   Broadcast   255 -> 11111111
```

Netzadressen und Broadcast errechnen sich nach der Regel

- Alle Bits der Hostadresse auf 0 ist die Netzadresse.
- Alle Bits der Hostadresse auf 1 ist die Broadcastadresse.

Nur daß eben die Hostadresse jetzt nur noch aus den letzten 6 Bits besteht.

Die Transportschicht (TCP und UDP)

In der Transportschicht arbeiten zwei unterschiedliche Protokolle, TCP und UDP. Der wesentliche Unterschied zwischen beiden Protokollen ist, daß TCP *verbindungsorientiert* arbeitet und UDP *verbindungslos*.

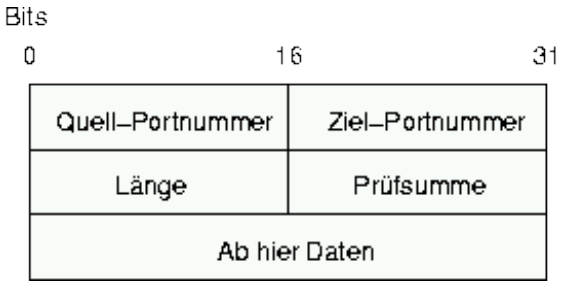
Beiden Protokollen gemeinsam ist die Verwendung sogenannter Portnummern. Dieses System beruht auf der Tatsache, daß es mehrere Verbindungen über ein Transportprotokoll gleichzeitig geben kann. So kann ein Rechner beispielsweise gleichzeitig über FTP Daten von einem anderen Rechner kopieren und via TELNET auf dem gleichen Rechner eingeloggt sein. Das erfordert es, daß das Transportprotokoll in der Lage ist, zu unterscheiden, welche Pakete an FTP und welche an TELNET adressiert sind. Diese Adressierung übernehmen die Portnummern. Die Standard-Internet-Protokolle haben festgelegte Portnummern (definiert in `/etc/services`).

User Datagram Protocol (UDP)

UDP ist ein sehr einfaches Protokoll, das Anwendungen die Möglichkeit bietet, direkten Zugriff auf die Datagramm-Services von IP zu nutzen. Das ermöglicht die Übermittlung von Daten mit einem Minimum an Protokollinformationen.

Das UDP-Message Format:

UDP als datagramorientiertes Protokoll benötigt keinerlei Informationen über Sequenznummern oder ähnliches, eine sehr einfache Headerstruktur genügt hier:



Quellportnummer
: bezeichnet die Portnummer (/etc/services) des Anwenderschichtprotokolls, von dem die UDP-Message abgeschickt wurde.

Zielportnummer
: bezeichnet die Portnummer des Empfängerprotokolls auf der Anwendungsschicht.

Länge
: Hier steht die Länge der gesamten UDP-Message

Prüfsumme
: Eine Prüfsumme über das Datenfeld

UDP wird überall dort verwendet, wo entweder die Datenmenge so klein ist, daß es sich nicht lohnen würde, einen großen Header zu benutzen, weil der größer als die eigentlichen Daten wäre oder wo die Anwendungen selbst noch Überprüfungen des Paketinhaltes vornehmen.

Lohnenswert ist der Einsatz auch dort, wo reine Frage-Antwort Mechanismen auftreten. Dort ist kein verbindungsorientiertes Protokoll nötig, weil ja nach dem Senden einer Frage klar ist, wenn nach einer bestimmten Zeit keine Antwort eingeht, so wird das Paket verlorengegangen sein und die Frage muß nochmal gestellt werden.

Wichtig ist die Feststellung, daß das UDP-Protokoll keinen Datenstrom verarbeitet, sondern Datagramme direkt. Es stellt also keinerlei Mechanismen zur Verfügung um einen Datenstrom in Pakete aufzuteilen und wieder zusammenzubauen.

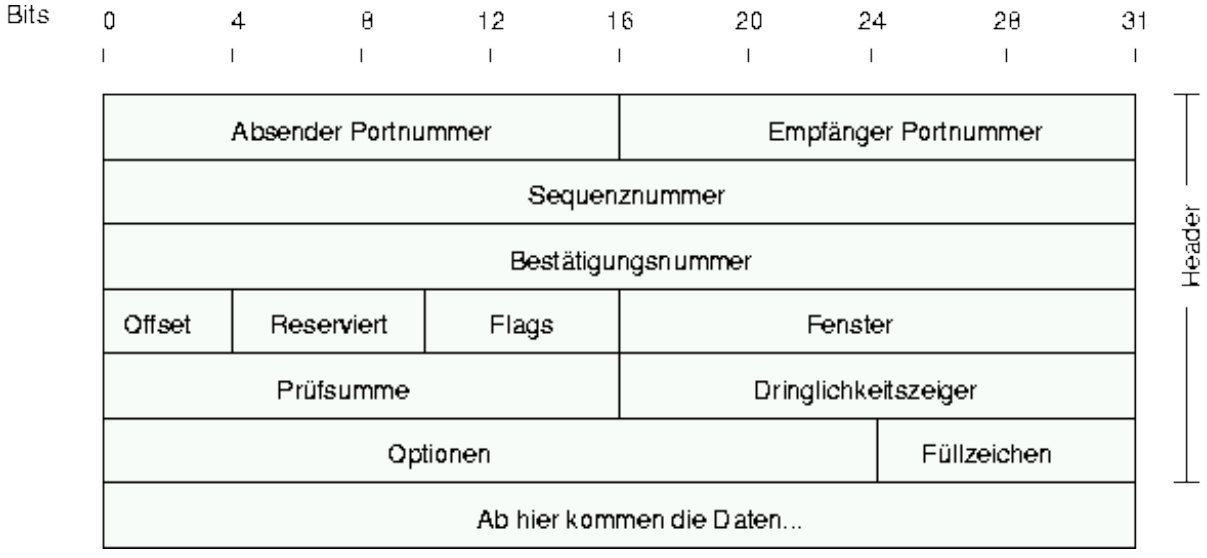
Transmission Control Protocol (TCP)

TCP ist im Gegensatz zu UDP ein verbindungsorientiertes Protokoll, das Datenströme verarbeitet, die von der Anwendungsschicht kommen. TCP garantiert die Versendung von Daten durch eingebaute Handshake-Mechanismen.

TCP bietet einen verlässlichen Datentransfer durch die Verwendung eines Mechanismus, der Datenpakete (sogenannte Segmente) solange an einen Empfänger schickt, bis der eine Bestätigung des Empfangs rückmeldet. Dabei überprüft der Empfänger auch wieder eine Prüfsumme, erst wenn die richtig ist, schickt er das Signal des Empfangs.

Das TCP-Segment Format

Im Gegensatz zu UDP braucht TCP als streamorientiertes Protokoll immer Informationen, an welcher Stelle des Datenstroms das Segment eingefügt werden soll. Außerdem ist Information über die Größe des Empfängerbuffers nötig, um zu verhindern, daß es zu Überläufen kommt. Es ist also ein entsprechend komplexer Header nötig:



Absender Portnummer
: bezeichnet die Portnummer (/etc/services) des Anwenderschichtprotokolls, von dem der Datenstrom abgeschickt wurde.

Empfänger Portnummer
: bezeichnet die Portnummer des Empfängerprotokolls auf der Anwendungsschicht.

Sequenznummer
: Gibt die Position im Datenstrom an, an der dieses Segment eingefügt werden soll.

Bestätigungsnummer

Dient zur Bestätigung des Empfangs eines Segments. Dieses Feld enthält immer die Nummer, die im nächsten Segment als Sequenznummer stehen soll.

Daten-Offset
Gibt die Größe des Headers in 32 Bit Worten an. Ohne Optionen sind es 5. Damit kann genau bestimmt weden, wo der Datenbereich des Segments beginnt.

Flags
Verschiedene Flags zur Kommunikationssteuerung

Fenster
Mit diesem Wert wird die Größe des Buffers angezeigt, der von einem Endknoten für diese Verbindung reserviert wurde. Der sendende Knoten darf keinesfalls mehr Daten als die angegebene Puffergröße senden, ohne auf den Eingang einer Bestätigung zu warten.

Prüfsumme
Eine Prüfsumme über das gesamte Segment (Daten und Header)

Dringlichkeitszeiger
Wird bei besonders dringend zu verarbeitenden Paketen gesetzt. Wenn gesetzt enthält dieses Feld als Wert die Endadresse des Datenfeldes, das als dringlich gilt.

Optionen
Verschiedene Optionen zur Kommunikation wie etwa die maximale Segmentgröße

Füllzeichen
Zum Auffüllen der Optionszeile auf 32 Bit

TCP ist verbindungsorientiert, d.h., daß dieses Protokoll nicht einfach Daten losschickt wie UDP sondern zuerst einen Handshake durchführt, um sich mit dem Empfänger über dessen Bereitschaft zu synchronisieren. Das Prinzip ist einfach, eine TCP Übertragung beginnt immer erst mit der Nachfrage ob der Empfänger bereit ist (SYN). Der Empfänger sendet ein entsprechendes Signal (SYN.ACK) und erst nachdem dieses Signal erhalten wurde startet TCP die Übertragung der Segmente.

TCP ist streamorientiert, d.h., daß es seine Daten als kontinuierlichen Datenstrom ansieht. Durch die Verwendung von Sequenznummern kann das empfangende TCP die Segmente wieder in richtiger Reihenfolge zusammenbauen und sie wieder zu einem Datenstrom formieren.

Die Anwendungsschicht

Auf dieser Schicht laufen die Anwendungen, die übers Netz miteinander kommunizieren. In der Regel handelt es sich hier um Client/Server Paare, also um zwei verschiedene Programme, eins, das einen Dienst anbietet (server), ein anderes, das diesen Dienst in Anspruch nimmt (client).

Damit diese Anwendungen sich nicht gegenseitig stören, hat jede Anwendung eine eigene Portnummer. Sie dient als Adresse, an die die Transportschicht einen Datenstrom oder eine UDP-Meldung weiterleitet.

Die wichtigsten Portnummern sollte man parat haben, um die LPI102-Prüfung abzulegen. Es sind

Port	Protokoll	Transportprotokoll	Beschreibung
20	FTP-Data	TCP	Datenkanal einer FTP-Verbindung.
21	FTP	TCP	Kontrollkanal einer FTP-Verbindung.
22	SSH	TCP oder UDP	SecureShell (Verschlüsselter Login)
23	TELNET	TCP	Terminal Emulation over Network
25	SMTP	TCP	Simple Mail Transfer Protocol
53	DNS	TCP oder meist UDP	Nameserver
80	WWW/HTTP	meist TCP oder UDP	Hypertext Transfer Protokoll
110	POP3	TCP oder UDP	Post Office Protocol zum Holen von Mails
119	NNTP	TCP	Net News Transfer Protocol
139	NetBIOS-SSN	TCP oder meist UDP	Windows Netzwerk Sitzungsdienste

143	IMAP2	TCP oder UDP	Interim Mail Access Protocol (verschlüsselt)
161	SNMP	UDP	Simple Network Management Protocol

Programme in diesem Zusammenhang

Die geforderten Kenntnisse der ganz oben genannten Programme dient zunächst nur dafür, zu wissen, wozu welches Programm benutzt werden kann. Hier jeweils eine kurze Beschreibung über die Programme und ihre Anwendung:

ping

Das Programm **ping** benutzt das ICMP-Protokoll um herauszufinden, ob ein Rechner im Netz verfügbar ist. Dazu sendet es eine ICMP-Meldung (Typ 8) an den Zielrechner. Der Zielrechner gibt dann eine Kopie des Paketes mit dem Typ 0 (*pong*) zurück. Das wieder ankommende Paket wird auf die Standard-Ausgabe geschrieben. **ping** sendet ohne Unterlass weiter, bis es mit Strg-C abgebrochen wird.

Um **ping** von vorneherein auf eine bestimmte Anzahl von Paketen zu beschränken, gibt es den Parameter `-c`, mit dem die Anzahl der zu sendenden Pakete angegeben werden kann:

```
ping -c4 foo.bar.com
```

sendet 4 Pakete an den Rechner `foo.bar.com`. Wenn **ping** mindestens eine Antwort erhält, so gibt es in diesem Fall eine 0 zurück, ansonsten einen Wert ungleich 0. Somit kann **ping** auch in Scripts benutzt werden, um festzustellen, ob ein bestimmter Rechner verfügbar ist.

traceroute

Das Programm **traceroute** ermöglicht es, genau zu bestimmen, über welche Gateways ein Paket zu einem Zielrechner geroutet wird. Es schickt ein Paket an den nächsten Gateway (unsere default route), das das Lebensdauerfeld auf 0 gesetzt hat. Der Gateway schickt daraufhin eine ICMP-Fehlermeldung Typ 11 (`time-exceeded`) an uns zurück. Anschließend wird das selbe Paket mit einer Lebensdauer von 1 geschickt. Jetzt bleibt es beim nächsten Gateway hängen, auch der schickt eine ICMP-Typ 11 Fehlermeldung zurück. So wird die Lebensdauer jedesmal um eins erhöht und das Paket kommt so immer einen Gateway weiter. Aus den Fehlermeldungen der Gateways werden ihre IP-Adressen extrahiert und auf die Standard-Ausgabe geschrieben. Das Ganze wird solange wiederholt, bis der Zielrechner erreicht ist.

Der einzig nötige Parameter für **traceroute** ist der Name oder die IP-Adresse des Zielrechners.

host

Das Programm **host** wird benutzt, um Nameserverabfragen direkt von der Kommandozeile aus vorzunehmen. Es kann über sehr viele Kommandozeilenparameter genau eingestellt werden, wie und was gesucht werden soll. Der einzig notwendige Parameter ist der gesuchte Domainnamen. Die Antwort besteht in diesem Fall aus seiner IP-Adresse.

Wird statt einem Domainnamen eine IP-Adresse eingegeben, so wird umgekehrt der passende Domainnamen ausgegeben, sofern er existiert und ermittelbar ist.

Wenn nicht nur die Information über die IP-Adresse eines Domainnamens gesucht wird, kann der Parameter `-a` angegeben werden, der alle verfügbaren Informationen über einen Rechner anfordert und darstellt.

dig, nslookup, nsquery

Auch diese Programme machen Nameserverabfragen. Sie unterscheiden sich in ihren Fähigkeiten und ihrer Anwendung. So bietet beispielsweise **nslookup** eine interaktive Abfrageshell. **dig** ist in der Lage auch komplexe Probleme zu lösen, so kann zum Beispiel die aktuelle Liste aller Root-Nameserver mit dem Befehl

```
dig . ns
```

erfragt werden oder **dig** kann eine Liste aller Nameserver für eine bestimmte Zone erfragen. An sich werden alle vier Programme (**host, dig, nslookup** und **nsquery**) im Alltag dazu benutzt, explizit einen Nameserver abzufragen.

whois

whois sendet eine Nachfrage an eine RFC-812 Datenbank. In einer solchen Datenbank werden die Daten von Domaininhabern abgelegt. So kann mit **whois** herausgefunden werden, wer eine bestimmte Domain registriert hat, wer der Administrator davon ist usw. Die Menge der Ausgaben wird entweder durch die entsprechende Datenbank vorgegeben oder es wird über Kommandozeilenparameter genau angegeben, welche Information gesucht wird.

Im einfachsten Fall wird **whois** nur mit dem Parameter der Domain aufgerufen, über die Informationen erfragt werden sollen:

```
whois lpi.org
```

gibt als Ausgabe

```
Registrant:
Linux Professional Institute Inc.
78 Leander St.
Brampton, ON L6S 3M7
CA

Domain Name: LPI.ORG

Administrative Contact:
   Silberman, Wilma  nic@lpi.org
   78 Leander St.
   Brampton, ON L6S 3M7
   CA
   (905) 874-4822

Technical Contact:
   starnix, DNS  dns@starnix.com
   175 Commerce Valley Dr. W.
   Thornhill, ON L3T 7P6
   CA
   (416) 410-9342

 ...
```

ftp

Das File Transfer Protokoll dient zur Übertragung von Dateien über das Netzwerk. Es nutzt die TCP-Ports 21 (Kommandokanal) und 20 (Datenkanal) um über TCP/IP Dateien von einem Rechner zum anderen zu kopieren. Auch hier gibt es unter Unix Befehle (**rcp**), die besser auf das System abgestimmt sind, aber auch hier gilt, in heterogenen Netzen hat FTP den Vorteil.

Wie bei Telnet, so gibt es auch hier einen Server (ftpd) und einen Client (ftp, xftp, Netscape), die miteinander kommunizieren. Früher war es üblich, mit der FTP-eigenen Kommandosprache zu arbeiten, heute gibt es graphische Frontends, die FTP wesentlich konfortabler machen.

FTP besitzt zwei Übertragungsmodi, Text und Binärformat. Heute ist es üblich, alle Übertragungen im Binärformat vorzunehmen, es entstehen so weniger Probleme beim Umgang mit Textdateien, die 8 Bit Zeichensätze verwenden (Umlaute...).

Die klassische Art, FTP zu nutzen war interaktiv, mit der FTP-eigenen Kommandosprache. Dabei war fast der Eindruck einer kleinen Terminalsitzung zu erahnen, um sich im Dateibaum zu bewegen und dort Dateien zu kopieren. Die wichtigsten Befehle von FTP sind:

open *Hostname*
: Stellt eine Verbindung zum gewünschten Host her. Dabei wird ein Passwort abgefragt.

close
: Schließt eine bestehende Verbindung.

dir
: Zeigt das aktuelle Inhaltsverzeichnis.

cd *Verzeichnis*
: Wechselt in das angegebene Verzeichnis.

lcd *Verzeichnis*
: Wechselt das Verzeichnis auf der Client Seite (local change dir).

pwd
: Print Working Dir - gibt das aktuelle Verzeichnis an.

get *Datei*
: Kopiert eine Datei vom Server zum Client.

put *Datei*
: Kopiert eine Datei vom Client zum Server.

del *Datei*
: Löscht eine Datei auf dem Server.

binary
: Wechselt in den Binärmodus.

ascii
: Wechselt in den Textmodus.

quit
: Beendet FTP.

Moderne FTP-Clients wie Netscape oder xftp steuern diesen Vorgang über eine graphische Benutzeroberfläche, die den Umgang mit ftp stark vereinfacht.

Eine große Bedeutung im Internet hat anonymes FTP (ohne Username und Passwort), das als Möglichkeit verwendet wird, um Public-Domain-Software downzuloaden.

telnet

Telnet (Terminal EmuLation over NETwork) dient dazu, Zugriff auf einen, am Netz angeschlossenen Rechner in Form einer Terminalsitzung zu liefern. Auf Unix-Systemen läuft auf der Clientseite das Programm telnet, auf der Serverseite ein Daemon namens telnetd. Der Telnet-Service liegt auf dem TCP-Port 23.

Auf Unix-Systemen wird heute das Kommando **rlogin** verwendet, das in etwa die gleiche Funktionalität besitzt, wie Telnet, dabei aber die Unix Eigenheiten besser unterstützt. In heterogenen Umgebungen, die nicht nur aus Unix-Systemen bestehen ist Telnet aber flexibler. Sicherer ist allerdings die Anwendung von **ssh** statt Telnet, da es die gesamte Kommunikation (inklusive der Login-Prozedur und Passwortübergabe) verschlüsselt abwickelt. Telnet überträgt alles, auch das Passwort unverschlüsselt.

Eine Telnetsitzung wird gewöhnlich durch den Client mit dem Befehl

```
telnet Hostname
```

gestartet. Das Programm stellt dann eine TCP-Verbindung mit dem gewünschten Host her und kommuniziert dort mit dem Telnet-Daemon telnetd. Dieser Daemon stellt jetzt ein sogenanntes Pseudo-Terminal (oder auch Network Virtual Terminal - NVT) zur Verfügung. Dieses Terminal lässt sich genauso steuern, wie eine Unix-Konsole, so daß auch bildschirmorientierte Programme via Telnet benutzt werden können (z.B. vi, mc, ...)

1.112.3 - TCP/IP Konfiguration und Problemlösung

Beschreibung: Prüfungskandidaten sollten in der Lage sein, Konfigurationseinstellungen und den Status verschiedener Netzwerkschnittstellen anzusehen, zu ändern und zu überprüfen. Dieses Lernziel beinhaltet die manuelle und automatische Konfiguration von Schnittstellen und Routing-Tabellen. Dies bedeutet im speziellen das Hinzufügen, Starten, Stoppen, Neustarten, Löschen und Rekonfigurieren von Netzwerkschnittstellen. Dies beinhaltet auch das Ändern, Listen und Konfigurieren der Routing-Tabelle und die manuelle Korrektur einer falsch gesetzten Default-Route. Kandidaten sollten auch in der Lage sein, Linux als DHCP-Client und TCP/IP-Host zu konfigurieren und Probleme im Zusammenhang mit der Netzwerkkonfiguration zu lösen.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- `/etc/HOSTNAME` oder `/etc/hostname`
- `/etc/hosts`
- `/etc/networks`
- `/etc/host.conf`
- `/etc/resolv.conf`
- `/etc/nsswitch.conf`
- **ifconfig**
- **route**
- **dhcpcd, dhcpclient, pump**
- **host**
- **hostname (domainname, dnsdomainname)**
- **netstat**
- **ping**
- **traceroute**
- **tcpdump**
- die Netzwerk-Scripts, die während der Systeminitialisierung ausgeführt werden.

Netzwerkkonfiguration ist etwas, was im Normalbetrieb immer automatisch beim Systemstart abläuft. Init-Scripts übernehmen die Konfiguration der Schnittstellen, das Anlegen der Routen und vieles mehr. Trotzdem ist das Wissen um die manuelle Konfiguration wichtig, erstens für Problemlösungen und zweitens, weil damit auf die Schnelle auch eine Umkonfiguration von Netzwerkkarten oder ein experimenteller Aufbau möglich ist.

Notwendig ist also beides, das Wissen um die manuelle und automatische Konfiguration. Im Prinzip ist die zweite Frage kein großer Wurf mehr, wenn die erste beantwortet ist. Da die automatische Konfiguration die selben Befehle und Mechanismen einsetzt, wie die manuelle - nur eben in Scripts - sollte das Verständnis der automatischen Konfiguration kein Problem darstellen, wenn die Befehle aus der manuellen bekannt sind.

Manuelle Konfiguration von Netzwerkkarten

Die erste Grundvoraussetzung für die Konfiguration von Netzwerkschnittstellen ist, daß TCP/IP im Kernel aktiviert ist. Das ist eigentlich immer der Fall, selbst bei Rechnern, die nicht für Netze geplant waren, weil unter Linux viele lokale Dienste ohne diese Fähigkeit nicht funktionieren würden.

Die Darstellung der notwendigen Konfigurationsarbeiten wird sich jetzt auf "normale" Netzwerkkarten (Ethernet) beziehen, die Einstellungen für andere Netzwerkschnittstellen, seien es andere Kartentypen (Token Ring, FDDI, ...) oder andere Schnittstellen wie ppp oder slip, werden mit den selben Befehlen vorgenommen.

Damit die Konfiguration einer Netzwerkkarte überhaupt funktionieren kann, muß der Kernel die Karte zuallererst mal erkannt haben. Das heißt, daß der Kernel die Unterstützung für eine bestimte Netzwerkkarte bereits fest eingebaut haben muß oder daß das entsprechende Modul geladen sein muß. Diese Technik wurde bereits im Abschnitt 1.105.1 besprochen. Wir gehen hier also davon aus, daß das Modul für die zu konfigurierende Netzwerkkarte bereits geladen ist.

Das IP-Protokoll definiert eine abstrakte Schnittstelle *interface* um auf verschiedene Hardware zugreifen zu können. Diese Schnittstellen bieten eine Reihe von standardisierten Operationen an, die alle mit Versand und Empfang von Daten zu tun haben. Durch das Laden von Gerätetreibern (Modulen) werden diese Schnittstellen mit physikalischen Geräten (Netzwerkkarten o.ä.) verbunden. Dadurch entstehen symbolische Schnittstellennamen wie *eth0, eth1* für Ethernetkarten, *tr0, tr1* für Token Ring Karten oder *arc0, arc1* für Arcnet-Karten. Der lokale Loopback heißt *lo*. Selbst serielle Verbindungen ins Internet mit ppp oder slip haben dann Interfacenamen wie *sl0, sl1* oder *ppp0, ppp1*.

Diese Zuweisung geschieht in der Regel schon beim Booten, es können bei einem modularen Kernelaufbau aber auch während der Laufzeit des Kernels noch Gerätetreiber geladen werden. Dann erfolgt natürlich auch die Zuweisung von symbolischen Namen erst während der Laufzeit.

Konfigurieren des Interfaces

Nachdem wir ein funktionsfähiges Interface haben, z.b. *eth0* muß jetzt dafür gesorgt werden, daß dieser Schnittstelle eine IP-Adresse zugewiesen wird. In der Regel geschieht das natürlich automatisch, aber um zu sehen was in den Init-Scripts passiert, ist es wichtig, diese Schritte auch mal von Hand durchgeführt zu haben.

Zum Konfigurieren eines Interfaces gibt es das Programm **ifconfig**, das einem beliebigen Interface IP-Adresse, Netzmaske und Broadcast-Adresse zuweist. Die Anwendung ist erstmal simpel:

```
ifconfig lo 127.0.0.1
ifconfig eth0 192.168.200.55
```

Damit haben wir dem Interface *lo*, also dem lokalen Loopback die ihm zustehende Adresse `127.0.0.1` zugewiesen, die Ethernetkarte *eth0* bekommt die Adresse `192.168.200.55`. Besser wäre noch gewesen, gleich die Netmask und Broadcast-Adresse mit anzugeben, mit den Schlüsselwörtern *netmask* und *broadcast* ist das möglich:

```
ifconfig eth0 192.168.200.55 broadcast 192.168.200.255 netmask 255.255.255.0
```

Somit wäre unser Interface jetzt komplett konfiguriert. Das Programm **ifconfig** kann aber noch mehr. Wird es ohne Optionen aufgerufen, so zeigt es Informationen zu allen bestehenden aktiven Netzschnittstellen an. (mit der Option -a werden auch die Schnittstellen dargestellt, die *down* geschalten sind) Mit

`ifconfig`*Interface*

werden Informationen zum angegebenen *Interface* angezeigt. Neben der Informationsausgabe kann das Programm **ifconfig** noch folgende Optionen verarbeiten:

down
: Schaltet ein Interface schlagartig aus. Es ist danach für den Kernel nicht mehr erreichbar. Vorsicht, bei alten Versionen von **ifconfig** werden Routing Einträge nicht mitgelöscht, es können also noch Routen existieren, denen kein Interface zugeordnet ist.

up
: Schaltet ein abgeschaltetes Interface wieder ein.

pointopoint *ADRESSE*
: für SLIP und PPP, beides Protokolle für serielle Verbindungen. Hier ist das Interface etwas anders zu steuern, weil ja sozusagen nur ein Netz aus zwei Rechnern (Point to Point) besteht.

metric *NUMMER*
: Nur für RIP. RIP summiert alle anfallenden *metric* Werte einer Verbindung um daraus die Kosten zu errechnen (heute selten benutzt) und damit Routenwahl auch hinsichtlich der Kosten zu optimieren.

mtu *BYTES*
: Maximum Transmission Unit - Größte Übertragungseinheit des verwendeten Netzprotokolls. Bei Ethernet (802.3) 1500, bei SLIP 296.

promisc
: Schaltet ein Interface in den *promiscuos mode* in dem alle Pakete des Netzwerks, egal ob an diesen oder einen anderen Rechner geschickt, empfangen werden. Brauchbar zur Analyse des Netzverkehrs, der Fehlersuche, aber auch zum Abhören von Netzleitungen.

Alle diese Optionen werden nach der Nennung des Interfacenamens angehängt, z.B.

```
ifconfig eth0 metric 2 mtu 1024
```

Der Befehl **ifconfig** ist also sowohl zu Konfiguration eines Interfaces, als auch zur Diagnose ein wichtiges Hilfsmittel.

Erstellen der Routen

Nachdem die Interfaces konfiguriert worden sind, müssen noch die Routing-Tables (Routen-Tabellen) erstellt werden. Das geschieht mit dem Programm **route**. Die Tabelle existiert nicht als Datei, sondern wird im Kernelspeicher verwaltet. Das heißt, der Aufruf von **route** muß also bei jedem Start erfolgen.

Die heutigen modernen Versionen von **ifconfig** erledigen bereits das Anlegen der Route ins eigene Netz sobald eine Schnittstelle mit einer IP-Adresse versehen wird. Diese Aufgabe musste bei älteren Systemen auch mit dem Befehl **route** erledigt werden.

Wird **route** ohne Parameter (oder nur mit `-n`) aufgerufen, so zeigt es den aktuellen Routing-Table an. (`-n` zeigt Adressen immer nummerisch, auch wenn Hostnamen bekannt sind)

Ansonsten muß **route** immer entweder mit dem Schlüsselwort `add` oder `del` versehen werden. `add` steht für das Hinzufügen einer Route, `del` für das Löschen einer Route.

```
route add [-net | -host] XXXX [gw GGGG]
          [metric MMMM] [netmask NNNN] [dev DDDD]
```

oder

```
route del XXXX
```

Dabei stehen `XXXX` für die Route (Netz- oder Hostadresse), `GGGG` für die Adresse eines Gateways, `MMMM` für einen numerischen Wert, der als metric-Wert benutzt werden soll, `NNNN` für eine Netzmaskierung und `DDDD` für ein TCP/IP Interface wie etwa *eth0*.

Für einen einfachen Rechner in einem lokalen Netz würde also der folgende Eintrag genügen:

```
route add -net 192.168.200.0
```

Damit wäre eine Route definiert, die alle Pakete, die ans Netz 192.169.200.0 gerichtet sind an die Ethernetkarte schickt. Das weiß das System, weil die Ethernetkarte ja die Adresse 192.168.200.55 hat, also die gleiche Netzadresse. Dieser Befehl ist bei modernen Versionen von **ifconfig** unnötig geworden, weil diese Route eben automatisch bei der Konfiguration angelegt wird.

Nehmen wir an, in unserem Netz ist ein Gateway (z.B. `192.168.200.77`) ans Internet angeschlossen. Wir brauchen jetzt also zwei Routen, eine ins lokale Netz und einen an den Gateway. An diesen schicken wir alle Pakete, die nicht fürs lokale Netz gedacht sind.

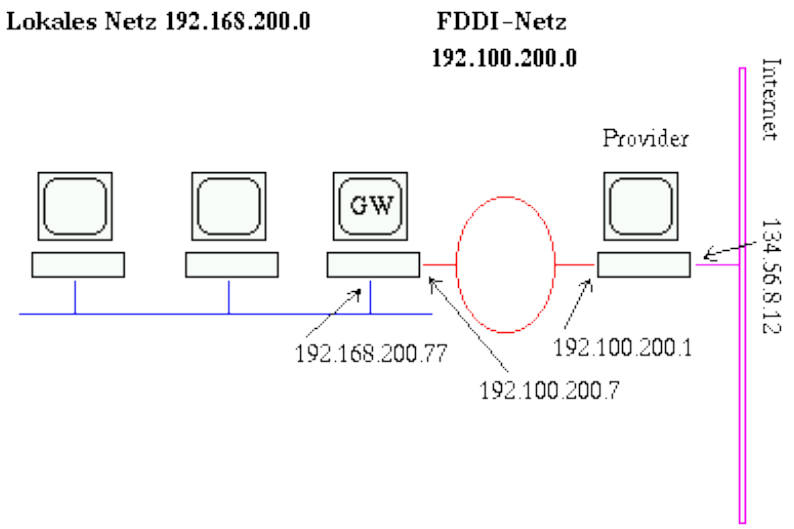
```
route add -net 192.168.200.0
route add default gw 192.168.200.77
```

Alle Pakete, die wir losschicken und die nicht für das lokale Netz bestimmt sind, werden jetzt direkt an den Gateway `192.168.200.77` geschickt. Das Schlüsselwort `default` kann auch mit `-net 0.0.0.0` ausgetauscht werden, der schon bekannten Adresse der *default route*.

Auch in diesem Beispiel ist die erste Zeile bei modernen **ifconfig** Versionen unnötig.

Jetzt bleibt nur noch die Frage, wie die Routing-Tables eines Gateways konfiguriert werden müssen. Nehmen wir

an, der Gateway `192.168.200.77` hat eine FDDI-Karte und hängt direkt am Rechner des Providers. Der Provider hat eine eigene Netzadresse für sein FDDI-Netz. Nehmen wir an, die Provider-Netzadresse ist `192.100.200.0` Der Rechner des Providers (`192.100.200.1`) leitet alle Pakete ans eigentliche Internet weiter. Das heißt, er dient als nächster Gateway ins Internet. Unser Gateway hat eine FDDI-Karte (Interface *fddi0*) mit der Adresse `192.100.200.7`.



Also müssen wir in unserem Gateway drei Routen konfigurieren. Eine ins lokale Netz, eine ins FDDI-Netz (beide werden wieder automatisch durch **ifconfig** angelegt und sind hier nur für ein besseres Verständnis angegeben) und eine Default-Route zum Providergateway. Selbstverständlich müssen hier auch beide Interfaces richtig konfiguriert sein:

```
ifconfig eth0 192.168.200.77 broadcast 192.168.200.255 netmask 255.255.255.0
ifconfig fddi0 192.100.200.7 broadcast 192.100.200.255 netmask 255.255.255.0
route add -net 192.168.200.0
route add -net 192.100.200.0
route add default 192.100.200.1
```

Der Gateway hat jetzt also zwei statische Routen in die jeweiligen Netze, die er verbindet (das geht selbstverständlich auch mit zwei Ethernetkarten) und eine Default Route, die wiederum auf den Gateway des Providers verweist. Wenn jetzt ein beliebiger Rechner in unserem lokalen Netz ein Paket an eine gänzlich unbekannte Adresse schickt (etwa `123.45.67.89`) dann wird sie (weil es keine feste Route dorthin gibt) über die Default-Route geschickt. Diese ist mit unserem Gateway verbunden. Der hat auch keine feste Route zu `123.45.67.89`, schickt das Paket wiederum auf seine Default Route, die mit dem Gateway des Providers verbunden ist. Der gibt das Paket seinerseits ans Internet weiter.

Das Programm **route** ist - wie schon **ifconfig** in der Praxis ein wichtiges Hilfsmittel zur Diagnose. Nachdem die Routen in der Regel beim Starten des Rechners automatisch gesetzt werden, ist die Anwendung zu Diagnosezwecken sehr viel häufiger, als die manuelle Erstellung von Routen.

Das Programm netstat

Dieses Programm gibt Informationen über den Status bestimmter Netzverbindungen zurück. Interessant sind folgende Möglichkeiten:

netstat -r oder -rn Wie route oder route -n Die angezeigten Flags haben folgende Bedeutung:

- **G** Route ist Gateway
- **U** Interface ist UP
- **H** Nur ein Host kann über die Route erreicht werden (PPP/SLIP/loopback)
- **D** Route wurde von ICMP eingetragen
- **M** Route wurde von ICMP verbessert (modified)

netstat -i Gibt die Statistik der Interfaces an, Flags haben folgende Bedeutung:

- **B** Broadcast ist gesetzt
- **L** ist Loopback
- **M** promiscous mode - Alle Pakete werden empfangen
- **O** ARP ist ausgeschaltet
- **P** Point to Point Verbindung
- **R** Running
- **U** Up

netstat -t, -u, -w, -x, -a zeigt die aktiven Sockets für TCP, UDP, RawIP, Unix oder Alle.

Automatische Konfiguration über Init-Scripts

All die oben erwähnten Befehle müssen natürlich nicht jedesmal von Hand eingegeben werden, wenn ein Linuxrechner startet. Sie werden von entsprechenden Init-Scripts gestartet, sobald in einen Runlevel gewechselt wird, der das Netzwerk aktivieren soll.

Die unterschiedlichen Distributionen gehen hier denkbar unterschiedliche Wege. Gemeinsam ist ihnen, daß die Einstellungen für die Netzwerkkarten (Adressen und Routen) in bestimmten Dateien (unter `/etc`) eingetragen werden, die dann von einem entsprechenden Init-Script (z.B. `/etc/init.d/networking`) ausgelesen werden. Das Init-Script startet dann die Befehle **ifconfig** und **route** mit den gefundenen Werten aus den Dateien.

Eine weitere Vereinfachung ist mit den Programmen **ifup** und **ifdown** möglich. Beide Programme sind Frontends für **ifconfig** und **route**. Im Verzeichnis `/etc/network` erwarten diese Programme eine Datei mit Namen `interfaces`, die Informationen über die benutzten Netzwerkschnittstellen beinhalten. In dieser Datei steht beispielsweise:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
      address 192.168.100.123
      netmask 255.255.255.0
      network 192.168.100.0
      broadcast 192.168.100.255
      gateway 192.168.100.20
```

Mit dieser Information kann **ifup** dann entsprechend die Netzwerkschnittstellen mit Adressen versehen und die notwendige Route zum Gateway (default route) setzen.

Einsatz eines DHCP-Servers

Wenn im Netz ein DHCP-Server läuft, dann ist die gesamte Konfiguration der Netzwerkkarten nicht mehr nötig. Ein solcher Server hält alle notwendigen Informationen bereit, die ein Rechner benötigt, um seine Netzwerkkarte zu konfigurieren. Beim Hochfahren des Systems wird ein Broadcast ins Netz gesendet, um einen DHCP-Server zu finden. Der Server antwortet daraufhin mit einem Paket, das alle notwendigen Angaben zur Konfiguration beinhaltet.

Die Prüfungsziele für LPI102 beinhalten nicht die Installation eines DHCP-Servers, sondern nur die Anbindung eines Rechners an einen bestehenden Server.

Es gibt unterschiedliche Programme, die einen Linux-Rechner zum DHCP-Client machen. Die bekanntesten sind

- **dhcpcd**
  Der DHCP-Client-Daemon. Er arbeitet vollkommen selbstständig und durchsucht das Netz nach einem DHCP-Server. Anschließend weisst er jeder gefundenen Netzwerkschnittstelle eine entsprechende IP-

Adresse zu.

- **dhclient**
  Auch dieses Programm arbeitet als Daemon, es bezieht seine Informationen, welche Interfaces über DHCP konfiguriert werden sollen aus der Konfigurationsdatei `/etc/dhclient.conf`.
- **pump**
  Auch **pump** ist ein Daemon, der Netzwerkinterfaces über DHCP konfiguriert. Er kann aber auch als Alternative mit dem *bootp* Protokoll arbeiten, das eine ähliche Aufgabenstellung wie DHCP besitzt, allerdings auch das Booten plattenloser Workstations unterstützt. **pump** bezieht seine Konfigurationsinformationen aus der Datei `/etc/pump.conf`.

Jeder dieser Daemonen wird über ein Init-Script beim Systemstart gebootet und bleibt die ganze Zeit im Speicher. Das ist notwendig, weil DHCP-Server in regelmäßigen Abständen überprüfen, ob ein Rechner noch aktiv ist, um gegebenenfalls eine wieder frei gewordene Adresse neu vergeben zu können.

Konfigurationsdateien für Netzwerke

Neben den Init-Scripts und anderer Netzwerk-relevanten Einstellungen, die von Distribution zu Distribution unterschiedlich sind, gibt es eine Reihe Konfigurationsdateien, die allen Linux-Versionen (und auch allen anderen Unixen) gemeinsam sind. Diese Dateien werden hier jetzt der Reihe nach besprochen.

/etc/HOSTNAME oder /etc/hostname

Diese Datei enthält den Hostnamen des Rechners. Beim Systemstart wird ein Init-Script ausgeführt, das diese Datei ließt und den entsprechenden Namen darin als Hostnamen setzt. Das dazu verwendete Programm heißt ebenfalls **hostname** und ermöglicht es auch, im laufenden Betrieb den Hostnamen zu erfragen (ohne weitere Parameter) oder ihn zu verändern.

Das Programm **hostname** gibt als Ausgabe nur den Hostnamen, ohne Domainnamen an. Soll der volle Name (full qualified domain name) angegeben werden, so muß **hostname** mit der Option `--fqdn` aufgerufen werden. Der Domainname alleine kann mit dem Befehl **dnsdomainname** erfragt werden. Dieser Befehl ist aber nicht in der Lage den Domainnamen zu verändern, da es stark von der Organisation des Netzes abhängt, wo dieser Domainname eingestellt ist. Der Befehl **domainname** gibt statt der echten DNS-Domain den Namen der NIS-Domain an.

/etc/hosts

Die Datei `/etc/hosts` enthält IP-Adressen und die dazu passenden Hostnamen. Überall, wo eigentlich IP-Adressen erwartet werden, können stattdessen auch Hostnamen angegeben werden, wenn diese in der Datei `/etc/hosts` angegeben sind. Die Datei ist also - übertragen gemeint - eine Art Mini-Nameserver für das lokale Netz. Der Aufbau ist einfach, jede Zeile enthält zuerst eine IP-Adresse, dann den dazu passenden Namen und eventuelle Aliase auf diesen Namen.

```
192.168.100.1    marvin.mydomain.com marvin
```

bedeutet also, daß immer, wenn der Name `marvin.mydomain.com` oder der Name `marvin` verwendet wird, dieser Name durch die Adresse `192.168.100.1` ersetzt wird.

/etc/networks

Die Datei `/etc/networks` entspricht der Datei `/etc/hosts`, mit dem Unterschied, daß hier nicht Hostnamen, sondern Netzwerknamen verwendet werden. Diese Netzwerknamen werden mit Netzwerkadressen verbunden.

Im Unterschied zu `/etc/hosts` werden in dieser Datei zuerst die Namen und dann die Adressen angegeben. Eventuelle Aliase werden nach der Adresse angegeben:

```
buero1 192.168.1.0
buero2 192.168.2.0 meinnetz
backbone 192.168.100.0
```

Die angegebenen Adressen müssen Netzwerkadressen sein, also alle Bits des Hostadressenteils auf 0 gesetzt haben.

/etc/host.conf

Die Auflösung von Hostnamen zu IP-Adressen wird von einer speziellen Library erledigt (resolv+). Diese Library ist es auch, die entsprechend Nameserver und die Datei `/etc/hosts` abfrägt, wenn statt einer IP-Adresse ein symbolischer Name gefunden wurde.

Die Datei `/etc/host.conf` ist die Konfigurationsdatei für diese Bibliothek. Die wichtigste Aufgabe dieser Datei ist es, die Suchreihenfolge festzulegen, in der Namen in IP-Adressen aufgelöst werden. Entweder wird zuerst der Nameserver gefragt, und dann die Datei `/etc/hots`, oder umgekehrt. Diese Einstellung wird über die Zeile

```
order hosts,bind
```

erledigt. Das Beispiel gibt an, daß zuerst die Datei `/etc/hosts` und erst dann der Nameserver (bind) gefragt werden soll. Ist umgekehrt gewünscht, daß zuerst der Nameserver gefragt wird, so hieße die Zeile

```
order bind,hosts
```

Weitere Parameter sind:

multi
: Gültige Werte sind `on` und `off`. Wenn `on` gesetzt ist, liefert die Resolverbibliothek alle für den gesuchten Host gültigen Adressen aus `/etc/hosts` zurück, anstatt nur den ersten gefundenen Eintrag. Der Standardwert für diesen Parameter ist `off`, da seine Verwendung auf Installationen mit sehr großer /etc/hosts zu Performanceproblemen führen kann.

nospoof
: Gültige Werte sind `on` und `off`. Wenn dieser Parameter auf `on` gesetzt ist, versucht die Resolverbibliothek, das Spoofing von Hostnamen zu unterbinden, um die Sicherheit von rlogin und rsh zu verbessern. Nachdem für einen Hostnamen die Adresse gefunden wurde, wird geprüft, ob für diese Adresse auch genau dieser Hostname gefunden werden kann. Liegt keine eindeutige Zuordnung vor, schlägt die Namensauflösung fehl.

spoofalert
: Wenn dieser Parameter auf `on` gesetzt und gleichzeitig **nospoof** aktiv ist, protokolliert die Resolverbibliothek Fehler im Zusammenhang mit dem Spoofing-Schutz an Syslog. Der Standardwert hierfür ist off.

Zumeist enthält diese Datei nur zwei Zeilen, die die Reihenfolge definieren und das **multi** auf `on` setzen.

/etc/resolv.conf

Diese Datei wird auch von der Resolver-Library ausgelesen. Die wichtigste Aufgabe ist die Angabe der verwendeten Nameserver. Es können hier bis zu drei verschiedene Nameserver angegeben werden, die dann in der Reihenfolge der Angaben befragt werden, wenn ein DNS-Lookup stattfindet.

Der hierfür verwendete Befehl lautet

nameserver*IP-Adresse*

Dieser Befehl kann bis zu dreimal in der Datei auftauchen.

Ein weiterer Befehl für die vernünftige Arbeit mit dem Resolver ist

search*Domain-Name*

Hier wird die lokale Domain (ohne Hostnamen) angegeben, um Namen, die ohne Domainnamen angegeben wurden, als lokale Namen festzulegen. Genauer gesagt, werden alle Rechnernamen (mit oder ohne Domainnamen) zunächst einmal mit dieser Endung versehen, die dort angegeben wurde. Erst wenn ein Rechner nicht mit der Endung gefunden wurde, wird nach einem Rechner ohne diese Endung gesucht.

Normalerweise enthält die Datei `/etc/resolv.conf` nur diese beiden Einträge.

/etc/nsswitch.conf

Verschiedene Funktionen der C-Standard-Library müssen anhand verschiedener Konfigurationsdateien konfiguriert werden. Traditionell wird das z.B. mit Dateien wie `/etc/passwd` erledigt. Später wurden dann zusätzliche Dienste eingeführt, die auch bestimmte Informationen anbieten, wie etwa NIS/YP, das auch Informationen über Usernamen und UIDs bereitstellen kann, nur eben netzweit und nicht nur lokal. Um festzulegen, welche dieser Dienste in welcher Reihenfolge verwendet werden sollen, existiert die Datei `/etc/nsswitch.conf`. Ihr Mechanismus ist in etwa zu vergleichen mit der Angabe der Reihenfolge in der Datei `/etc/host.conf`, nur daß sie sich nicht nur auf Namensauflösung, sondern auf alle verwendeten Konfigurationsmöglichkeiten bezieht.

Konkret werden die folgenden Datenbanken von `/etc/nsswitch.conf` verwaltet (in Klammern immer die normalerweise verwendeten Konfigurationsdateien):

aliases
: Mail-Aliases von **sendmail** (`/etc/aliases`).

ethers
: Ethernet-Adressen

group
: User-Gruppen (`/etc/group`).

hosts
: Hostnamen und Adressen (`/etc/hosts`).

netgroup
: Netzweite Listen von Usern und Hosts, die für Bestimmungen von Zugriffsrechten wichtig sind.

network
: Netzwerknamen und -adressen (`/etc/networks`).

passwd
: User-Passwörter und andere Informationen (`/etc/passwd`).

protocols
: Netzwerkprotokolle (`/etc/protocols`).

publickey
: Public- und Secret-Keys für Secure_RPC

rpc
: Remote Procedure Call Namen und Portnummern (`/etc/rpc`).

services
: Portnummern von Netzwerkdiensten (`/etc/services`).

shadow
: Shadow-Passwörter (`/etc/shadow`).

In der Datei kann jetzt für jede dieser Datenbanken die Suchreihenfolge angegeben werden, ob zuerst über NIS, dann über die Dateien (files) oder umgekehrt gesucht werden soll.

Troubleshooting mit tcpdump

Wenn Probleme in einem Netz auftauchen, so kann mit Hilfe des Programms **tcpdump** jedes einzelne Paket des Netzes protokolliert und analysiert werden. Dazu wird die Netzwerkkarte in den *promisquous-mode* geschaltet, das heißt, sie gibt jedes Paket, auch wenn es nicht die eigene Mac-Adresse hat, an die Vermittlungsschicht weiter.

tcpdump zeigt jetzt die empfangenen Paket-Header an und ermöglicht so eine Diagnose der aufgetretenen Fehler. Die Anwendung ist ziemlich kompliziert und würde den Rahmen dieser Darstellung sprengen. Es genügt, zu wissen, daß es dieses Programm gibt und daß es folgendermaßen aufgerufen wird.

tcpdump `-i`*Interface*

Damit werden alle Pakete des genannten Interfaces abgehört. Das Interface wird mit seinem symbolischen Namen angegeben, also beispielsweise *eth0*.

tcpdump hat eine eigene Art Abfragesprache, die Befehle ermöglicht wie

```
tcpdump host foo
```

Zeigt nur Pakete an den oder vom Rechner foo.

```
tcpdump host foo and bar
```

Zeigt nur Pakete, die zwischen den Rechnern foo und bar ausgetauscht werden.

Andere Programme

Die in der Liste für dieses Prüfungsziel erwähnten Programme **host, ping** und **traceroute** wurden bereits im Abschnitt 1.112.1 besprochen.

1.112.4 - Konfiguration von Linux als PPP-Client

Beschreibung: Prüfungskandidaten sollten die Grundlagen des PPP-Protokols verstehen und in der Lage sein, PPP für ausgehende Verbindungen zu konfigurieren und zu verwenden. Dieses Lernziel beinhaltet die Beschreibung der Sequenz des Verbindungsaufbaus (bei gegebenem Login-Beispiel) und das Einrichten von automatisch bei Verbindungsaufbau auszuführenden Kommandos. Ebenfalls enthalten ist die Initialisierung und die Beendung einer PPP-Verbindung mittels Modem, ISDN oder ADSL und die Einstellung der automatischen Neuverbindung bei Verbindungsabbruch.

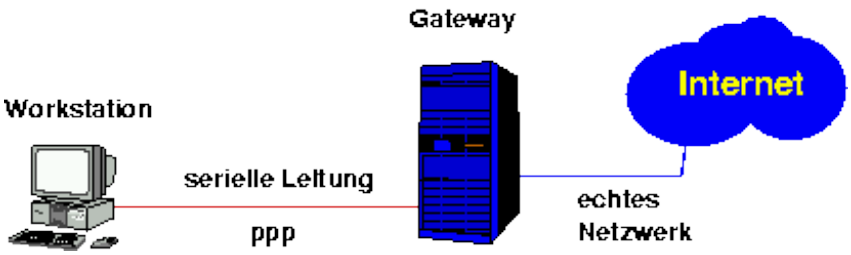
Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- `/etc/ppp/options.*`
- `/etc/ppp/peers/*`
- `/etc/wvdial.conf`
- **/etc/ppp/ip-up**
- **/etc/ppp/ip-down**
- **wvdial**
- **pppd**

Normalerweise werden IP-Verbindungen über Netzwerkkarten hergestellt. Die Protokolle von TCP/IP sind für den Umgang mit Netzwerken geschrieben, sie arbeiten z.B. mit den MAC-Adressen (ARP) und mit der Aufteilung eines Datenstroms in Pakete, wie es bei Netzen üblich ist.

Um zwei Computer statt über Netzwerkkarten über serielle Schnittstellen zu vernetzen, existieren mehrere Protokolle wie **SLIP** (Serial Line IP) und **PPP** (Point to Point Protocol). **SLIP** ist veraltet und wird nur noch sehr selten eingesetzt. Heute wird meist **PPP** benutzt.

Die wirkliche Aufgabenstellung von **PPP** ist es nicht, zwei alleinstehene Computer über eine serielle Leitung zu vernetzen. Heute sind Netzwerkkabel und -karten so billig, daß es sich nicht lohnen würde eine solche Art der Vernetzung zu realisieren. Zumal damit nur zwei Rechner vernetzt werden können, das System also nicht ohne weiteres erweiterbar ist. Die wirkliche Aufgabe von **PPP** ist es, Rechner über serielle Leitungen mit anderen Rechnern zu verbinden, die an ein tatsächliches Netz angeschlossen sind, so daß auch der einzelne Rechner Zugriff auf das echte Netz hat. In der Regel ist das echte Netz das Internet und die serielle Leitung ist eine Modem/ ISDN/ADSL Verbindung zum Provider:



Das **PPP** Protokoll ist ein spezielles Protokoll zur Realisierung einer IP-Verbindung über ein Medium, das nur zwei Kommunikationspartner verbindet. Also etwa serielle oder parallele Verbindungen. Spezielle Versionen von **PPP** sind für ISDN (syncPPP) oder für DSL (PPPoE) verfügbar, das klassische **PPP** ist typischerweise für Modemverbindungen gedacht.

Linux kann natürlich beide Rollen übernehmen, die der Workstation, die über **PPP** an einen Gateway angeschlossen wird (PPP-Client) oder die des Gateways, der eine **PPP**-Verbindung an ein echtes Netz knüpft (PPP-Server). Für die LPI102 Prüfung ist nur die Konfiguration eines PPP-Clients gefragt, diese Konfiguration wird im weiteren Verlauf dieses Abschnitts beschrieben.

Physikalischer Vorgang

Grundsätzlich ist der Vorgang des Aufbaus einer PPP-Verbindung in drei Teile aufzuteilen:

- Aufbau der Modemverbindung
- Loginvorgang
- Aufbau der PPP-Verbindung

Das heißt, zunächst einmal muß das Modem soweit gebracht werden, daß es eine Verbindung zu der bestimmten Nummer aufbaut. Das heißt das Modem muß initialisiert werden (`ATZ`) und die Initialisierung mit einem `OK` beantworten, anschließend muß die Nummer gewählt werden (`ATD` *`Nummer`*) und auf die Antwort `CONNECT` gewartet werden. Wenn diese Antwort empfangen wurde ist der erste Teil der Aufgabe erledigt. Damit haben wir eine bestehende serielle Verbindung zu dem anderen Rechner (Gateway). Auf dem Gateway hat das Modem abgenommen und ein normaler Login-Vorgang über getty wird jetzt gestartet. Das heißt, der Gateway schickt uns jetzt eine Meldung, die wahrscheinlich den Rechnernamen und das Wort `login:` beinhaltet. Diese Meldung wird von uns mit unserem Usernamen beantwortet. Daraufhin schickt der Gateway die Frage nach dem Passwort, die wir mit unserem Passwort beantworten. Wenn alles geklappt hat, ist der Login-Vorgang damit abgeschlossen.

Jetzt muß der eigentliche PPP-Daemon gestartet werden. Das muß auf beiden Seiten der Verbindung geschehen. Der Gateway hat den PPP-Daemon (**pppd**) als Startshell für den Usernamen eingetragen, er startet den Daemon also automatisch. Wir müssen den Daemon jetzt von Hand starten mit

```
pppd Schnittstelle Geschwindigkeit defaultroute
```

Die Schnittstelle ist dabei unsere serielle Schnittstelle, auf der das Modem hängt, die Geschwindigkeit ist meist 38400 (für moderne schnelle Modems) und die Option `defaultroute` sorgt dafür, daß die Verbindung zum Gateway als Default Route eingetragen wird. Der PPP-Daemon initialisiert jetzt die neu entstandene symbolische Netzschnittstelle *ppp0* mit dem Programm **ifconfig** (mit einer dynamischen IP-Adresse, die er von der Gegenstelle bekommt) und legt die default route auf dieses Interface.

Jetzt besteht eine IP-Verbindung zu dem Garteway und unser Rechner hat Zugriff auf das hinter dem Gateway liegende Netz. PPP ist also jetzt aktiv.

Dieser Vorgang kann natürlich automatisiert werden und muß niemals von Hand ausgeführt werden. Diese Darstellung dient nur dazu, den Vorgang zu verstehen, der im nächsten Abschnitt beschrieben wird.

Automatischer Aufbau einer PPP-Verbindung mit chat

Damit der oben beschriebene Vorgang automatisiert werden kann, gibt es das Programm **chat**. Es dient dazu, eine Kommunikation mit dem Modem (oder jeden anderen seriellen Verbindung) zu automatisieren. **chat** erwartet seine Befehle aus einer Scriptdatei, die eine sehr einfache Struktur hat. Jede Zeile eines chat-scripts besteht aus einem Paar:

"Warte auf" "Sende"

Das heißt, **chat** wartet immer auf ein angegebenes Wort, das vom Modem kommt, und sendet nach dem Empfang dann das entsprechende Wort *Sende*. Nehmen wir ein Beispiel. Wir wollen eine Verbindung zu einem Gateway aufbauen, der die Telefonnummer 12345 hat. Unser Username auf diesem Gateway ist `hans`, unser Passwort ist `12sd45gf`. Das entsprechende **chat**-Script würde also folgendermaßen aussehen:

```
''   ATZ
OK   ATD12345
CONNECT  ''
ogin: hans
ssword: 12sd45gf
```

Das bedeutet, wir warten auf nichts und senden ein `ATZ`. Dadurch wird das Modem initialisiert. Das Modem antwortet mit einem `OK`. Sobald wir das `OK` empfangen haben, senden wir den Modembefehl `ATD12345`, der das Modem veranlasst, die Nummer 12345 anzuwählen. Sobald das gegenüberliegende Modem des Gateways abgenommen hat und die beiden Modems ihren Handshake beendet haben, antwortet unser Modem mit dem Wort

CONNECT. Wenn wir das empfangen haben, schicken wir ein Return (''').

Jetzt wird der Gateway eine Loginaufforderung schicken. Wir wissen nicht, was genau er schickt, aber wir wissen, daß die Aufforderung mit den Buchstaben ogin: aufhört. Der Gateway könnte beispielsweise folgendes schicken:

```
Welcome to XYZ-Gateway
Login:
```

Es ist egal, was der Gateway sendet, sobald wir die Buchstaben ogin: empfangen, schicken wir wiederum den Usernamen hans. Der Gateway antwortet mit einer Aufforderung, ein Passwort einzugeben. Wir wissen nicht, ob er das groß oder klein schreibt, aber die letzten Buchstaben, die er schickt werden sicherlich ssword: sein. Sobald wir die empfangen haben, schicken wir das Passwort.

Mit diesem Befehl ist der Anmeldedialog abgeschlossen und die PPP-Verbindung wird aufgebaut.

Um den PPP-Daemon mit diesem Script zu starten, wird der Befehl

```
pppd "chat -f Scriptdatei" /dev/tty2 38400
```

eingegeben. *Scriptdatei* bezeichnet die Datei, die das oben besprochene Chat-Script enthält. Dadurch wird eine PPP-Verbindung automatisch aufgebaut. Der PPP-Daemon **pppd** wird aufgerufen und er erhält als ersten Parameter den Aufruf von Chat, mit der Scriptdatei. Die Angabe /dev/tty2 38400 bezieht sich auf die serielle Schnittstelle, an der das Modem hängt und die dort verwendete Geschwindigkeit.

Die zu verwendenden Optionen werden wir - statt sie auf der Kommandozeile einzugeben - in einer anderen Datei (/etc/ppp/options) angeben, dazu genaueres später.

Wenn der PPP-Daemon erfolgreich einen Verbindungsaufbau erledigt hat, so ruft er ein Shellscript mit Namen /etc/ppp/ip-up auf. Diesem Shellscript gibt er die folgenden Parameter mit:

```
ip-up Interfacename Gerätedate Geschwindigkeit Locale_IP Remote_IP
```

Dieses Script kann jetzt dazu verwendet werden, weitere Befehle auszuführen, die nach dem Verbindungsaufbau gewünscht werden, etwa Firewallregeln oder andere Kommandos. Beim Abbau einer Verbindung wird entsprechend das Script /etc/ppp/ip-down abgearbeitet, das wieder entsprechende Befehle beinhalten kann.

Automatischer Aufbau einer PPP-Verbindung mit wvdial

Die modernere Methode, mit der heute PPP-Verbindungen aufgebaut werden läuft über das Programm **wvdial**. Dieses Programm erledigt alle Aufgaben, von der Anwahl über das Modem, bis hin zum Start des PPP-Daemons. Es ersetzt das Chat-Programm und ermöglicht einen Login, ohne ein Script zu schreiben.

Wen **wvdial** startet, läd es zunächst seine Konfiguration aus der Datei /etc/wvdial.conf. Diese Datei enthält sowohl die Grundeinstellungen für die Schnittstelle, das Modem und die Geschwindigkeit, als auch Informationen über die anzuwählenden Provider (Telefonnummer, Username, Passwort).

Nachdem diese Datei gelesen wurde, initialisiert **wvdial** das Modem, wählt die entsprechende Nummer, authentifiziert den Login und startet den PPP-Daemon.

Die Konfigurationsdatei kann für verschiedene Provider Einträge besitzen. Sie ist aufgebaut wie eine Windows-INI Datei, also mit Sektionen, die durch Überschriften in eckigen Klammern begrenzt werden. Neben der Sektion [Dialer Defaults], die die Grundeinstellungen und einen voreingestellten Providereintrag enthält, kann es beliebig viele andere Einträge in der Form [Dialer *Providername*].

Wird **wvdial** ohne Parameter aufgerufen, so wählt es die Verbindung aus der Sektion [Dialer Defaults], wird es aber mit einem Parameter aufgerufen, der einen Provider beschreibt, so wird der entsprechende Einträge aus der Datei /etc/wvdial.conf benutzt. Der Aufruf

```
wvdial foo
```

benutzt also den Eintrag `[Dialer foo]` aus der Konfigurationsdatei.

Ein Beispiel für eine solche Datei könnte folgendermaßen aussehen:

```
[Dialer Defaults]
Modem = /dev/modem
Baud = 57600
Init1 = ATZ
Dial Command = ATDT
Idle Seconds = 180
Phone = 012345678
Username = foo
Password = bar

[Dialer provider1]
Phone = 0987654321
Username = hans
Password = asdf1234

[Dialer provider2]
Phone = 0918273645
Login Prompt = Weristda:
Username = hmueller
Password = 1082.oir
```

Im Abschnitt `[Dialer Defaults]` werden die entsprechenden Grundeinstellungen gemacht, die für alle anderen Abschnitte auch benutzt werden, wenn dort nichts anderes vereinbart ist. Jeder andere Abschnitt enthält dann noch die notwendigen Informationen über den anzurufenden Provider.

Wie beim Aufbau der Verbindung mit Chat, so wird auch hier beim Aufbau der PPP-Verbindung das Script `/etc/ppp/ip-up` abgearbeitet und beim Verbindungsabbau entsprechend `/etc/ppp/ip-down`.

Im Verzeichnis `/etc/ppp/peers` muß - bei Verwendung modernerer PPP-Daemonen - die Datei `wvdial` liegen, die ein paar Optionen für den PPP-Daemon enthält, die er benötigt, wenn er über **wvdial** gestartet wurde. In der Regel enthält diese Datei nur die Zeilen

```
noauth
name wvdial
replacedefaultroute
```

Falls andere Startprogramme verwendet werden, können sie auch in diesem Verzeichnis jeweils eine Datei besitzen, die die nötigen Optionen für sie setzen.

Die Datei /etc/ppp/options

Damit der PPP-Daemon nicht jedesmal mit viele Kommandozeilenoptionen aufgerufen werden muß, können die gewünschten Optionen in die Datei `/etc/ppp/options` eingetragen werden. Es ist sogar möglich, für jede Schnittstelle (`/dev/ttyS1, /dev/modem,...`) eine eigene solche Datei anzulegen, die dann entsprechend `/etc/ppp/options.ttyS1` usw. heißen muß.

Diese Datei kennt viele verschiedenen Einträge, die wichtigsten sind hier kurz erklärt:

noipdefault
: Der ppp-Client übernimmt nicht seine IP-Adresse aus `/etc/hosts` sondern bezieht seine Adresse vom Server. Diese Option sollte für normale Provider immer angegeben sein.

noauth

Keine automatische Authentifizierung. Das bedeutet, daß die normale Authentifizierung über das Chat-Script benutzt wird.

crtscts
Hardware-Flußkontrolle des Modems wird aktiviert.

lock
Lockdateien werden angelegt, um zu verhindern, daß ein anderer Prozeß das Modemgerät benützt, während wir online sind.

modem
Die Verbindung läuft über ein Modem. Das Gegenteil wäre **local** und wird nur angewandt, wenn zwei Rechner über ein Nullmodemkabel miteinander verbunden sind. Für eine Modemverbindung sollte immer **modem** gesetzt sein.

defaultroute
Die aufgebaute PPP-Verbindung wird als Default Route gesetzt.

persist
Nach einem Verbindungsabbruch wird nicht aufgelegt, sondern versucht, die Verbindung erneut aufzubauen.

maxfail *N*
Die Verbindung wird abgebrochen, wenn *N* mal die Verbindung durch einen Fehler abgebrochen wurde. Ein Wert von 0 schaltet dieses Feature ab.

idle *N*
Die Verbindung wird automatisch beendet, wenn mehr als *N* Sekunden keine Daten über sie gelaufen sind.

Zwingender Abbau einer PPP-Verbindung

Um eine PPP-Verbindung abzubrechen, muß einfach der PPP-Daemon per Signal INT gekillt werden. Da der Daemon seine ProzeßID immer in einer bestimmten Datei ablegt, kann diese Aufgabe automatisch durch ein Script erledigt werden. Die Datei, in die der Daemon seine PID ablegt ist normalerweise `/var/run/`*`interface`*`.pid`, wobei *interface* für die symbolische Schnittstelle (*ppp0, ppp1, ...*) der Verbindung steht. Um also die Verbindung von *ppp0* abzubrechen, kann der Befehl

```
kill -INT `cat /var/run/ppp0.pid`
```

ausgeführt werden. Natürlich kann dieser Befehl auch in einem Script verwendet werden.