

INHALT

Teil 1: Netzwerkprogrammierung unter Unix	1
1. Netztechnik	1
1.1 Begriffe	1
1.2 OSI-Modell, Protokoll	3
1.3 Dateneinkapselung	6
1.4 Anordnung der Bytes	7
1.5 Paketvermittlung	7
1.6 Adressen	8
2. Kommunikationsprotokolle: TCP/IP	9
2.1 IP	10
2.2 TCP und UDP	11
3. Client/Server Systeme	13
3.1 Das Client-Server Modell	13
3.2 Ein Client-Server Beispiel	13
4. Das Netzwerk-API: Berkeley Sockets.	14
4.1 Einführung	14
4.2 Übersicht	15
4.3 Socket-Adressen	17
4.4 Grundlegende Systemaufrufe	19
socket	19
bind	20

INHALT

connect	21
listen	22
accept	22
send, sendto, recv und recvfrom	23
close	24
Byte-ordnungsroutinen	25
Byte-Operationen	25
Adreß-Umwandlungen	26
Teil 2: Aufgaben.....	Ü1
Literaturverzeichnis	Lit1
Anhang: Einige praktische Netzwerk-Tools.....	A1
Ping	A1
Netstat	A1
Traceroute	A2
Telnet	A2

Teil 1: Netzwerkprogrammierung unter Unix

1. Netztechnik

1.1 Begriffe

Computersysteme lassen sich durch ein Kommunikationssystem, dem sogenannten **Computer-Netz**, miteinander verbinden. Die angeschlossenen Endsysteme (wie PCs, Unix-Arbeitsstationen oder Großrechner) werden als **Hosts** oder **Knoten** bezeichnet. Die Endsysteme werden über eine eindeutige Adresse (**Internet-Adresse**) identifiziert.

LAN steht für lokales Netz (**local area network**). Entfernungen bis zu einigen Km (z.B. Uni-Campus) werden dabei überbrückt.

Ein Langstreckennetz, oder **WAN** (von **wide area network**), verbindet Computer in verschiedenen Städte und Länder.

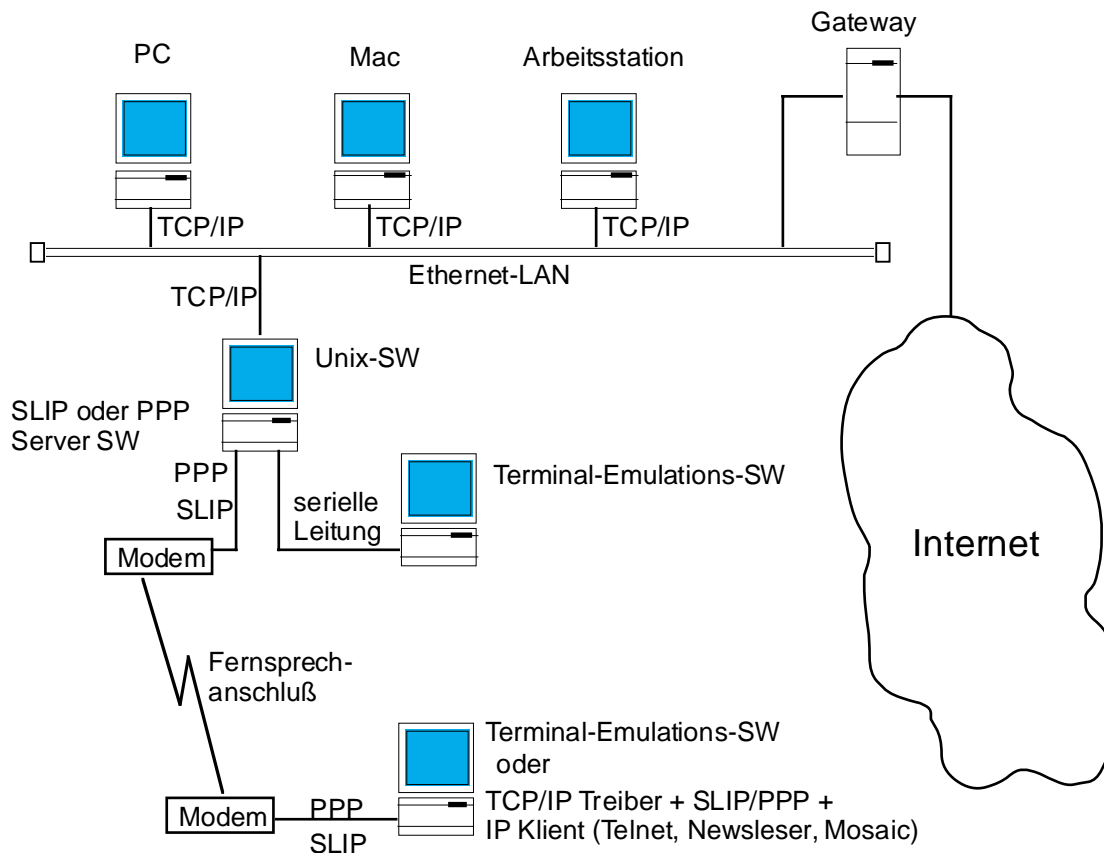


Bild 1.1 Anschlußmöglichkeiten an das Internet

Ein **Internet** oder Internet-Netz ist ein Netz, welches 2 eigenständige Netze miteinander verbindet (z.B. DARPA und BITNET). Dadurch können die Rechner des einen Netzes mit denen des anderen Netzes kommunizieren. Die Verbindung zweier physikalisch unterschiedlichen Netze läuft über eine Trennstufe, das **Gateway**. Man spricht auch von einem **Router**; der Ausdruck "Router" verdeutlicht daß hier bereits eine Entscheidung über die Richtung der weiterzusendenden Nachricht erfolgt.

Ein Gateway muß für jedes der angeschlossenen Netze eine Schnittstellenkarte (Netzadapter) besitzen. Der Ausdruck Gateway wird auch für besondere Software verwendet: ein Post-Gateway setzt zum Beispiel elektronische Post aus einem Format in ein anderes Format um.

1.2 OSI-Modell, Protokoll

Das **OSI-Modell** kann als ein Hilfsmittel verstanden werden, bestehende Netze und ihre Protokolle zu beschreiben und miteinander zu vergleichen: die Komplexität der Funktionen eines Netzwerks ist leichter zu bewältigen wenn man es als organisiert in verschiedene Ebenen oder Schichten betrachtet. Dabei baut eine obenliegende Schicht auf die Funktionalität der darunterliegenden Schicht auf.

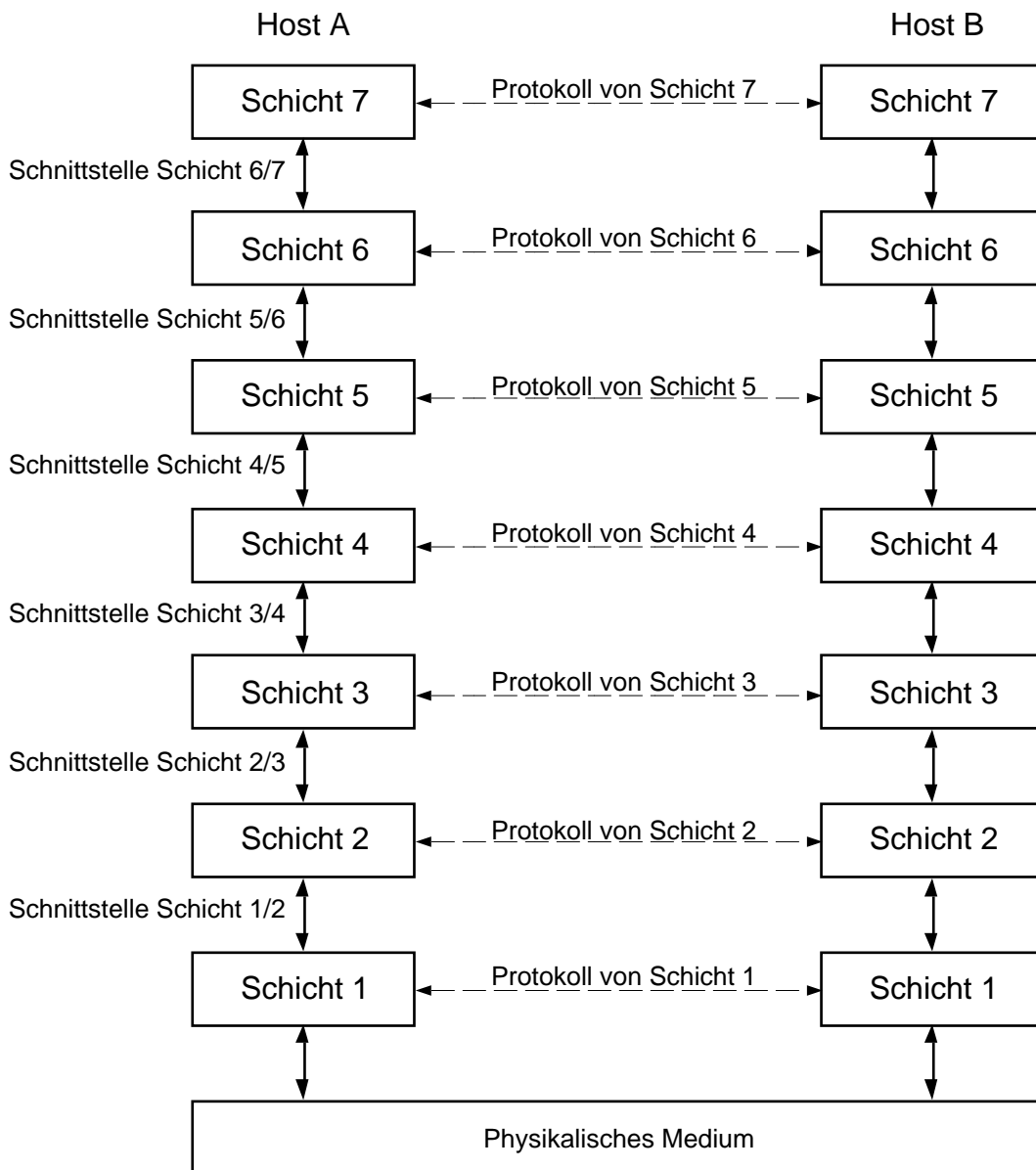


Bild 1.2: Schichten, Protokolle und Schnittstellen.

Eine (gedachte) Kommunikation gleichnamiger Schichten kann dabei von der Kommunikation mit den darunterliegenden Schichten abstrahieren. Oben stehendes Bild

zeigt die 7 Schichten des ISO/OSI Referenzmodells. Die Schichten selbst haben in der ISO/OSI-Norm bestimmte Aufgaben bekommen, diese zeigt das nächste Bild.

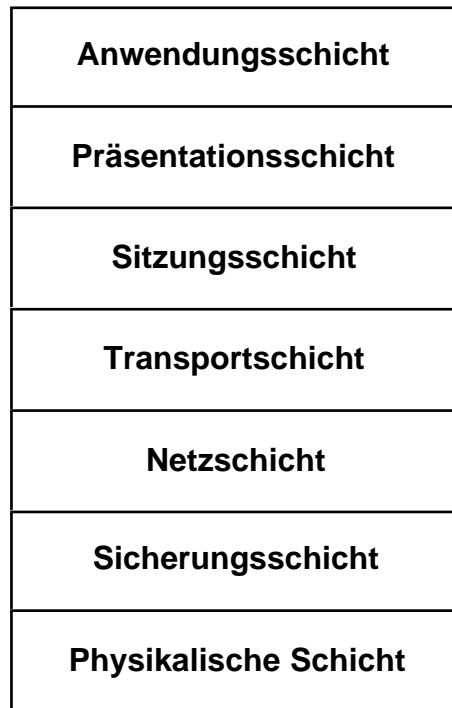


Bild 1.3: Aufgaben der Schichten im OSI-Modell.

Eine Vereinfachung zeigt das nächste Bild: das 4-Schichten-Modell. Bei dieser Betrachtung sind die oberen 3 Schichten in die sogenannte Prozeßschicht zusammengefaßt. Außerdem sind die unteren 2 Schichten zur Datenübertragungsschicht zusammengefaßt.



Bild 1.4 Das vereinfachte 4-Schichten-Modell

Als **Protokoll** bezeichnet man eine Menge von Regeln und Definitionen zwischen den Kommunikationspartnern. Da diese Protokolle sehr komplex sein können, werden Sie im

OSI-Modell in sieben Schichten aufgeteilt (siehe oben). Kein Netz ist jedoch exakt nach dem OSI-Schichtenmodell implementiert.

Der TCP/IP-Protokollsatz z.B. ergibt folgende Schichteneinteilung im vereinfachten 4-Schichten-Modell:

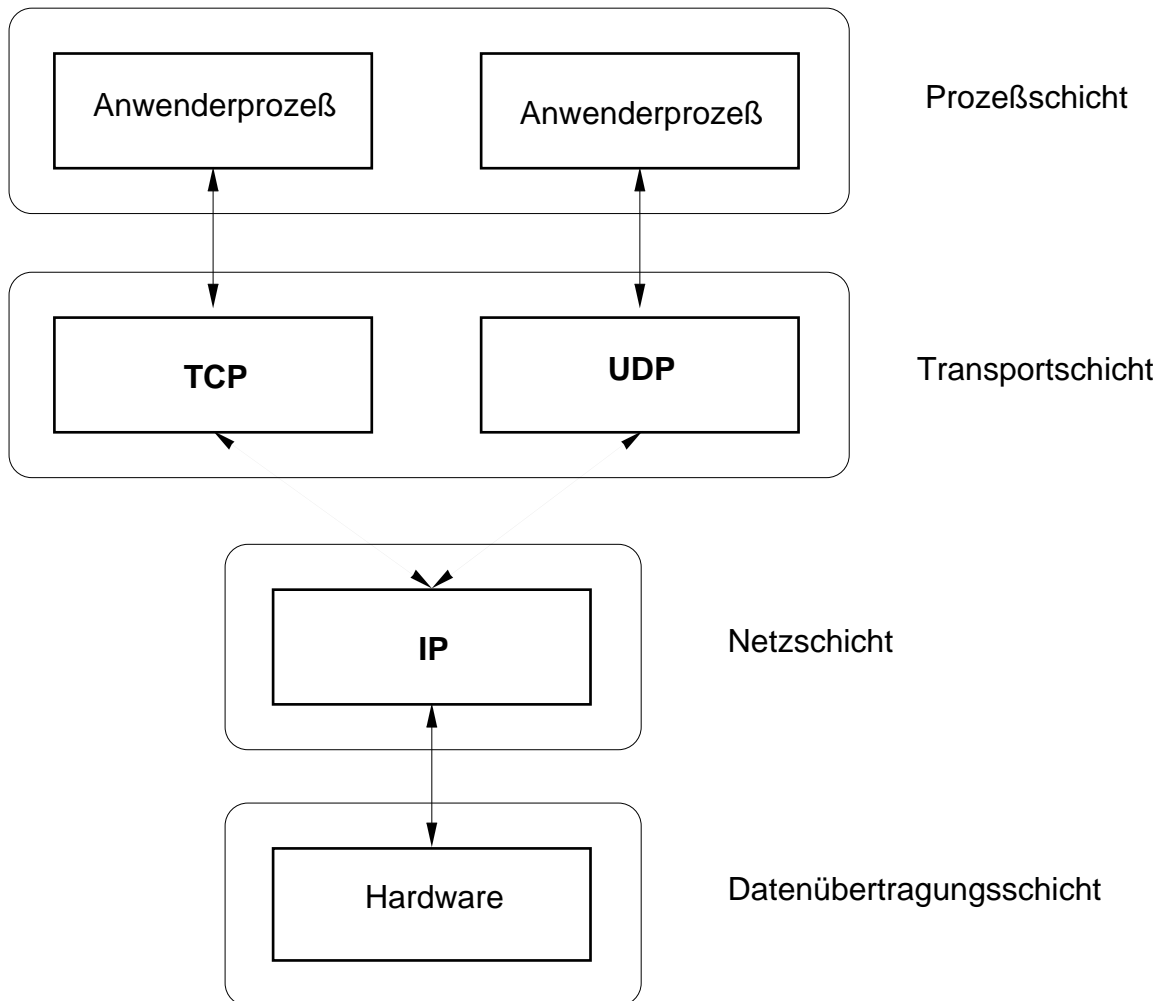


Bild 1.5 Der TCP/IP-Protokollsatz im 4-Schichten-Modell

1.3 Dateneinkapselung

Wenn ein Anwenderprozeß ein Datenpaket über das Netz schicken will, durchläuft dieses Paket die unter der Prozeßschicht liegenden Schichten. Das ursprüngliche Datenpaket wird in jeder Schicht um spezifische Informationen ergänzt:

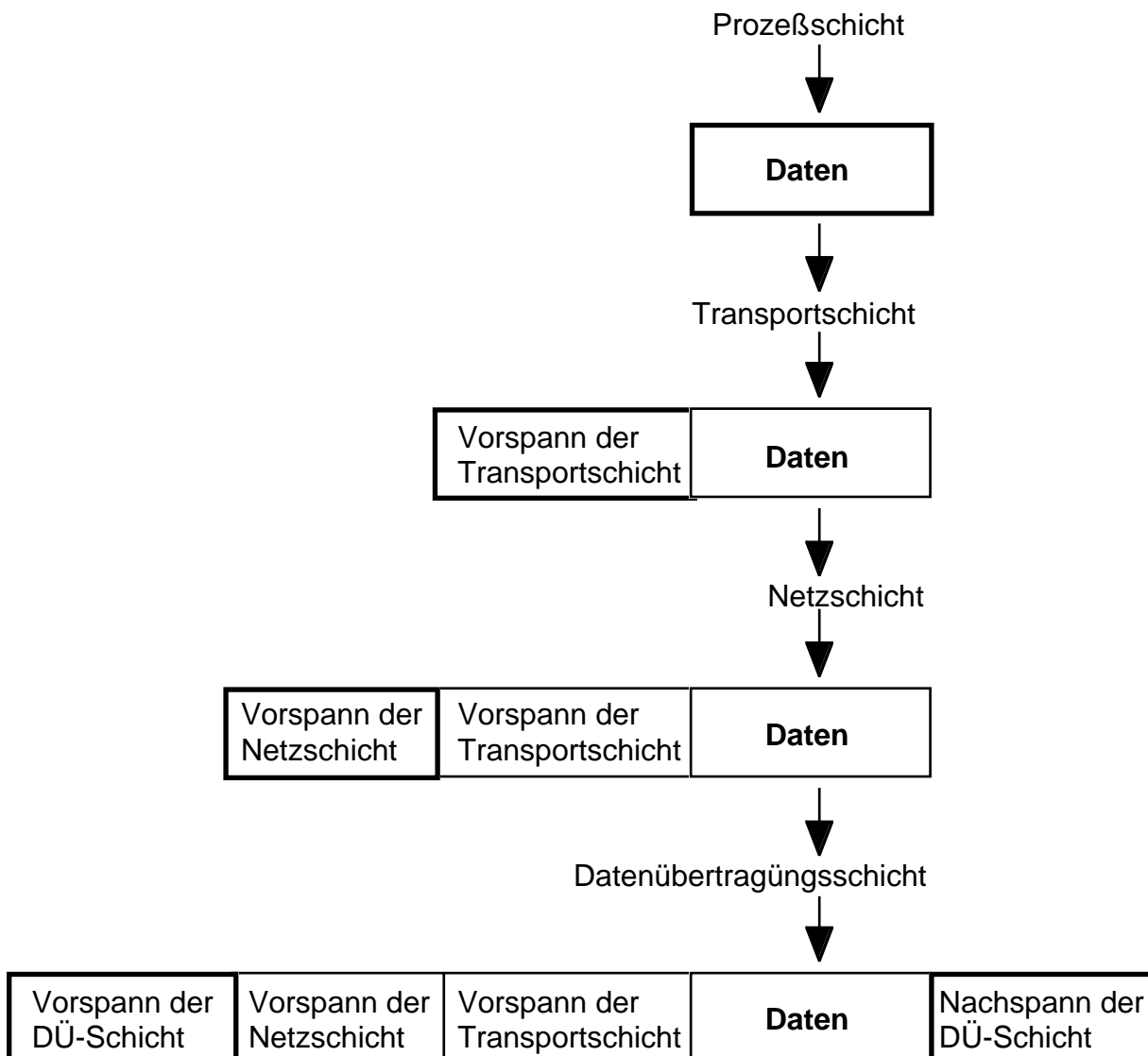


Bild 1.5 Dateneinkapselung

1.4 Anordnung der Bytes

Leider speichern nicht alle Computersysteme die einzelnen Bytes von Mehrbytegrößen in derselben Reihenfolge. Man unterscheidet die Little-Endian und die Big-Endian Anordnung.

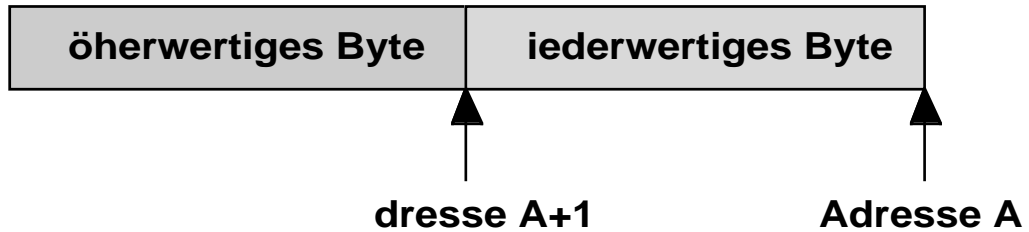


Bild 2.1 Little-Endian-Anordnung der Bytes bei einer 16-Bit-Größe

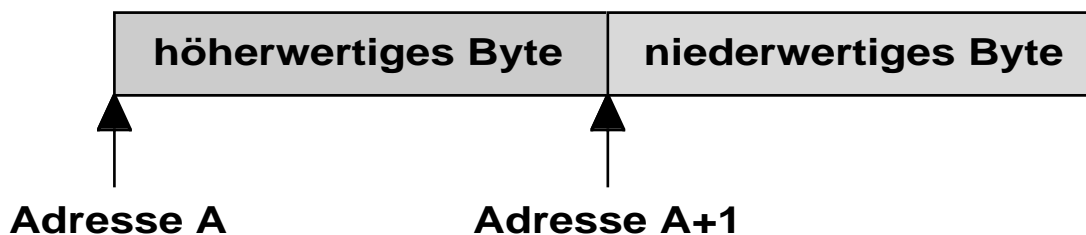


Bild 1.6 Big-Endian-Anordnung der Bytes bei einer 16-Bit-Größe

Deshalb wird für ein Netzprotokoll eine bestimmte Netz-Byte-Anordnung festgelegt. Bei TCP/IP Protokollen ist dies das Big-Endian-Format für 16- und 32-Bit-Integer-Werte (die Protokolle verwalten nur Integergrößen). Das Protokoll hat keinen Einfluß und keine Kontrolle über das Format der Daten, die von einer Anwendung über das Netz übertragen werden. Das Protokoll legt nur für die von ihm verwalteten Felder das Format fest.

1.5 Paketvermittlung

Die Datenkommunikation lässt sich in zwei Grundtypen unterscheiden: in **Leitungsdurchschaltevermittlung** (circuit-switching) und in **Paketvermittlung** (packet-switching).

Das öffentliche Telefonnetz ist das klassische Beispiel für ein Netz mit Durchschaltevermittlung. Für WANs werden oft Telefonleitungen *gemietet*. Eine Verbindung zwischen den beiden Teilnehmer kommt bei einer solchen Standleitung sofort zustande.

Das Internet ist üblicherweise ein paketvermitteltes Netz. Die zu übertragende Information wird in Teile zerlegt, und diese sogenannte Pakete werden jeweils übertragen.

Ein Paket ist die kleinste zu übertragende Einheit und wird selbständig durch die Netze zum Zielsystem übertragen. Dazu muß ein Paket die Zieladresse enthalten, um zum Empfänger gesendet zu werden. Die meisten Protokolle legen auch fest, daß die Adresse des Senders enthalten sein muß.

Bei einem Netz mit Durchschaltevermittlung kann man ab dem Zeitpunkt des Zustandekommens dieser Vermittlung die volle Kapazität des Übertragungskanals nutzen. Bei einem paketvermitteltes Netz muß die Bandbreite des Übertragungskanals zwischen mehrere Computersysteme aufgeteilt werden.

1.6 Adressen

Die Endpunkte einer Kommunikation sind 2 Anwenderprozesse, einer auf jedem der beteiligten Systeme. Bei einem Internet können die beiden Systeme sich in unterschiedlichen Netzen befinden. Diese sind durch ein oder mehrere Gateways miteinander verbunden. Dadurch ist eine Adressierung auf drei Stufen erforderlich.

- Ein bestimmtes auszuwählendes Netz muß angegeben werden.
- Jeder Host eines Netzes muß eine eigene Adresse besitzen.
- Jeder Prozeß, der auf einem Host abläuft, muß einen eigenen Bezeichner haben (auf diesem Host)

Üblicherweise besteht eine Hostadresse aus einer Netz-ID und einer Host-ID dieses Netzes. Die Identifikation eines Anwenderprozesses auf einem Host wird häufig durch einen vom Protokoll zugewiesenen Integerwert übernommen.

Beim TCP/IP-Protokoll wird beispielsweise ein 32-Bit Integerwert für die Hostadresse verwendet. Damit wird die Netz-ID und die Host-ID in kombinierter Form angegeben. Unter TCP und UDP werden zwei 16-Bit Portnummern verwendet um einen Anwenderprozeß zu kennzeichnen.

TCP/IP definiert eine Menge von *bekanntes Adressen*, um die Dienste, die ein Host zur Verfügung stellt, zu kennzeichnen. So wird zum Beispiel der Dateitransfer-Dienst FTP über die Portnummer 21 adressiert (siehe weiter unten).

2. Kommunikationsprotokolle: TCP/IP

Da das TCP/IP Protokoll im ARPANET über zehn Jahre vor dem OSI-Modell entstanden ist, ist es nicht verwunderlich daß es nicht voll der Schichteneinteilung dieser Norm entspricht.

Das Vermittlungsprotokoll IP ist verbindungsunabhängig und wurde für die sichere Verbindung zwischen der riesigen Zahl an lokalen Netzwerken und Weitverkehrsnetzen innerhalb des ARPA Internet geschaffen.

Das Transportprotokoll TCP (Transmission Control Protocol) ist verbindungsorientiert. Es ähnelt in seinem allgemeinen Aufbau dem OSI Transportprotokoll, aber es unterscheidet sich von ihm in allen Formate und Einzelheiten.

Da in den ersten zwanzig Jahren des ARPAnet kein Mensch ein Sitzungs- oder Darstellungsprotokoll benötigt hat, gibt es auch keines. Es gibt mehrere Anwendungsprotokolle, aber die sind nicht wie ihre OSI-Pendants aufgebaut. Es gibt, auf TCP/IP aufbauend, die Dienste elektronische Post, Dateitransfer (FTP) und den Fernanschluß (TELNET).

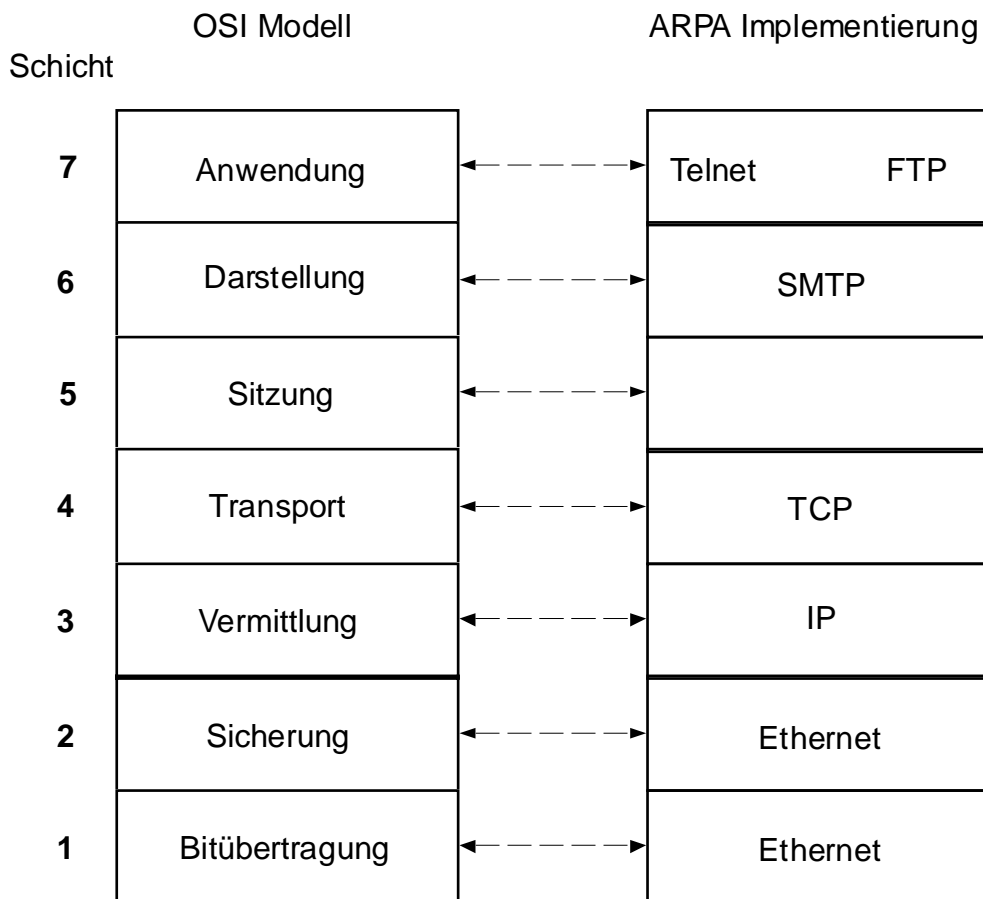


Bild 2.1: Die Schichten des ARPAnet im OSI-Modell.

2.1 IP

Die IP-Schicht unterstützt ein verbindungsloses und nicht zuverlässiges Übertragungssystem. Es ist verbindungslos, da es jedes IP-Datagramm (Nachrichtenpaket) für sich und unabhängig von anderen betrachtet. Sollen Datagramme einander zugewiesen werden, so muß diese Zuweisung in den oberen Schichten erfolgen. Jedes IP-Datagramm enthält die Adresse des Senders und des Empfängers. Dadurch kann für jedes Datagramm die Abfertigung und die Zielsuche eigenständig erfolgen. Die Bezeichnung "nicht zuverlässig" bedeutet, daß IP nicht garantiert, daß alle Pakete ankommen. Ob alles angekommen ist, dafür ist das Transmission Control Protocol (TCP) zuständig.

IP-Vorspann

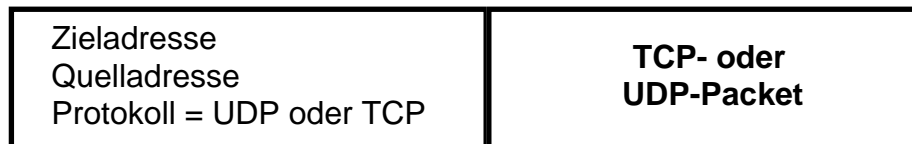


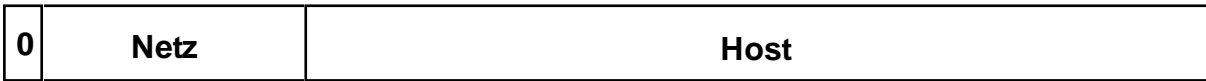
Bild 2.2 IP-Dateneinkapselung

2.1.1 IP-Adressierung

Das Internet benutzt die hierarchische IP-Adressierung. Die Adresse ist 4 Byte lang. Diese 32 Bit teilen sich auf in eine Netzwerkadresse, und eine Hostadresse.

Die Netzwerkadresse kann 8, 16 oder 24 Bit belegen. Man spricht von Klasse A, B oder C Adressen respektiv. IP-Adressen werden in ihrer Oktettform notiert, wobei jeder dezimale Oktettwert (=8 Bit =1 Byte) durch einen Punkt vom nächsten getrennt wird, z.B. 128.12.44.1.

Adresse Klasse A



Adresse Klasse B



Adresse Klasse C

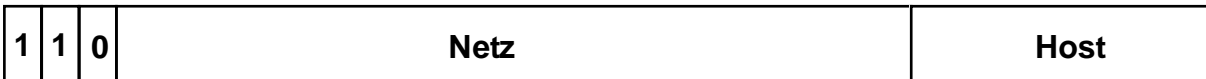


Bild 2.2: Adressen der Klassen A, B und C

2.2 TCP und UDP

Anwenderprozesse tauschen Daten über den TCP/IP-Protokollsatz aus, in dem sie entweder TCP-Daten oder UDP-Daten senden bzw. empfangen.

TCP stellt einem Anwendungsprogramm einen verbindungsorientierten und zuverlässigen byte-orientierten Datenstromdienst mit Vollduplex-Übertragung zur Verfügung. Die IP-Schicht stellt für TCP einen unzuverlässigen, verbindungslosen Übertragungsdienst dar. Deshalb muß das TCP-Modul die erforderliche Logik enthalten, um daraus für den Anwenderprozeß eine zuverlässige, virtuelle Verbindung zu machen. TCP steuert und kontrolliert den Aufbau und die Beendigung der Verbindungen zwischen Prozessen.

Im Gegensatz dazu steht UDP, das einen verbindungslosen, nicht zuverlässigen Datagrammdienst bereitstellt. UDP bietet gegenüber IP nur 2 weitere Eigenschaften: die Portnummern und eine optionale Prüfsumme zur Überwachung des UDP-Datagramminhaltes.

Portnummern

Zu einer Zeit kann mehr als ein Anwenderprozeß TCP und IP verwenden. Um die jeweiligen Daten dem zugehörigen Anwenderprozeß zuzuordnen zu können, verwenden TCP und UDP die 16-Bit-Portnummer, die als eine Erweiterung der Host-Adresse betrachtet werden kann.

TCP-Vorspann



Bild 2.3 TCP Dateneinkapselung

Zu beachten ist, daß die TCP-Ports nicht mit den UDP-Ports zusammenhängen, die TCP-Portnummer 1200 zum Beispiel ist unabhängig von der UDP-Portnummer 1200.

UDP-Vorspann



Bild 2.4 UDP Dateneinkapselung

Typischerweise definiert TCP/IP eine Menge von bekannten Adressen, um die bekannte Standard-Dienste, die ein Unix-Host zur Verfügung stellt, zu kennzeichnen. Diese Zuordnung wird in der Systemdatei /etc/services gemacht, von der ein Auszug folgt:

```
# Network services, Internet style
#
echo          7/tcp
echo          7/udp
discard       9/tcp          sink null
discard       9/udp          sink null
sysstat       11/tcp         users
daytime       13/tcp
daytime       13/udp
netstat       15/tcp
gotd          17/tcp         quote
ftp           21/tcp
telnet        23/tcp
smtp          25/tcp         mail
time          37/tcp         timserver
time          37/udp         timserver
```

3. Client/Server Systeme

Das Standardmodell für Netzwerkanwendungen ist das Client-Server-Modell. An einem einfachen Beispiel mit Datei-Ein-Ausgabe soll das Prinzip verdeutlicht werden. Auf die Unterschiede zur Netzwerkanwendung wird später eingegangen.

3.1 Das Client-Server Modell

Ein Server-Prozeß ist ein Prozeß, der darauf wartet, von einem Client-Prozeß angeregt zu werden, um für den Client-Prozeß einen Dienst auszuführen. Ein typischer Ablauf wäre:

- Auf irgendeinem Computersystem wird ein Server-Prozeß gestartet. Dieser initialisiert sich und geht anschließend in einen Wartezustand, um auf einen Client-Prozeß zu warten, für den er eine Aufgabe ausführen soll. (Server-Prozesse werden oft beim Hochfahren eines Systems gestartet.)
- Erst nach dem Start des Server-Prozesses wird ein Client-Prozeß gestartet (entweder auf dem selben System, oder auf einem anderen, über einem Netz verbundenem System). Client-Prozesse werden von den Anwendern zu einem nicht vorher festlegbaren Zeitpunkt gestartet, um vom Server Aufgaben zu erbeten. Einige Beispiele:
 - Ausdrucken einer Datei für den Client-Prozeß
 - Das Lesen /Schreiben von Dateien von/auf Datenträger
 - Meldung der Systemzeit an den Client-Prozeß
- Hat der Server seine Aufgabe für den Client-Prozeß ausgeführt, so geht er zurück in den Wartezustand. Er wartet nun auf den nächsten Auftrag eines Client-Prozesses.

3.2 Ein Client-Server Beispiel

Der Client liest einen Dateinamen vom Standardeingabegerät und gibt diesen Namen über IPC (Inter Process Communication) an den Server weiter. Der Server liest diesen Namen aus dem IPC-Kanal und versucht, die Datei zum Lesen zu öffnen. Wenn ihm dies gelingt, liest er die Datei und gibt den Inhalt an den IPC-Kanal weiter. Kann der Server die Datei nicht öffnen, so antwortet er mit einer Fehlermeldung (als ASCII-Text), das er die Datei nicht öffnen konnte. Der Client liest diese Fehlermeldung oder der Dateiinhalte aus dem IPC-Kanal aus und gibt alles an das Standardausgabegerät weiter.

4. Das Netzwerk-API: Berkeley Sockets.

4.1 Einführung

API steht für Applikationsprogramm-Interface, d.h. die Aufrufe die dem Programmierer zur Verfügung stehen. Eines der häufigsten verwendeten Netzwerk-APIs für Unix Systeme sind die Berkeley Sockets.

Die original TCP/IP Implementierung der BSD Version von Unix benutzte die sechs Systemaufrufe für Datei-Ein- und Ausgabe auch für die Netzwerk-Ein- und Ausgabe. Erst die Socket-Schnittstelle durch Berkeley erhielt die zusätzlichen, weiter unten im Detail beschriebene Aufrufe.

Damit zwei Anwenderprozesse auf verschiedenen Host miteinander kommunizieren können, müssen folgende Daten bekannt sein:

{Protokoll, lokale-Adresse, lokaler-Prozeß, Partner-Adresse, Partner-Prozeß}

Angewand auf das TCP/IP-Protokoll wird daraus (siehe Dateneinkapselung oben):

Protokoll	TCP oder UDP
lokale Adresse	lokale Internet Adresse des Hosts (32 Bit)
lokaler Prozeß	lokale Portnummer (16 Bit)
Partner-Adresse	Internet Adresse des entfernten Hosts (32 Bit)
Partner-Prozeß	entfernte Portnummer (16 Bit)

4.2 Übersicht

Abbildung 4.1 zeigt ein Ablaufdiagramm einer typischen verbindungsorientierten Übertragung. Zuerst wird der Server gestartet, einige Zeit später wird dann der Client gestartet, der eine Verbindung zum Server aufbauen möchte.

Für Client-Server, die ein verbindungsloses Protokoll benutzen, sind die Systemaufrufe verschieden. Abbildung 4.2 zeigt diese Systemaufrufe. Der Client richtet keine Verbindung zum Server ein. An deren Stelle benutzt der Client lediglich den Systemaufruf `sendto`, der die Adresse des Ziels (des Servers) als Parameter benötigt. Dementsprechend muß der Server keine Verbindung von einem Client akzeptieren. Statt dessen gibt der Server den Systemaufruf `recvfrom` heraus, der so lange wartet, bis Daten von einem Client ankommen. Der `recvfrom` liefert die Netzadresse des Client-Prozesses gemeinsam mit den Daten, so daß der Server seine Rückantwort an den korrekten Prozeß senden kann.

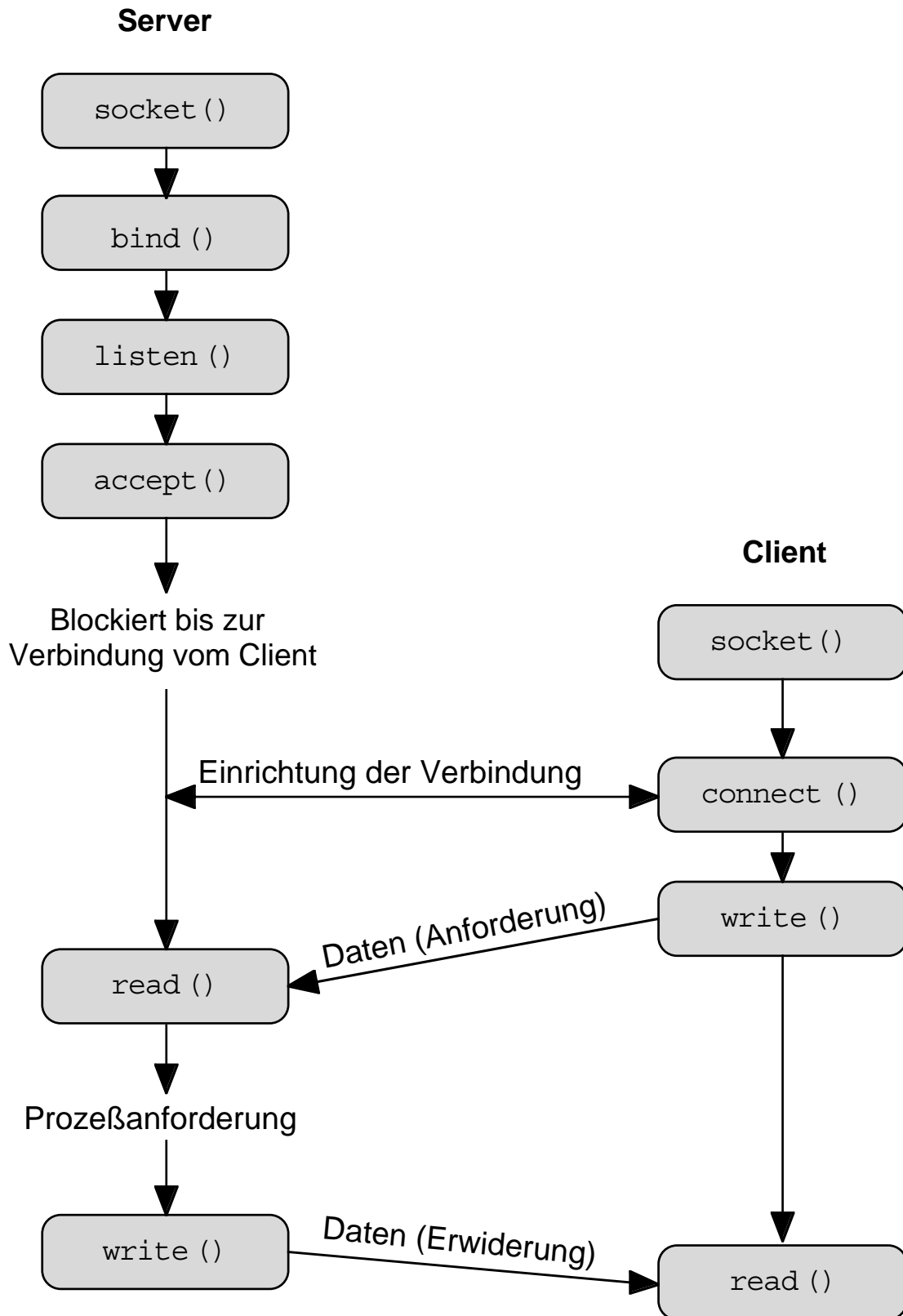


Bild 4.1 Socket-Systemaufrufe bei verbindungsorientiertem Protokoll

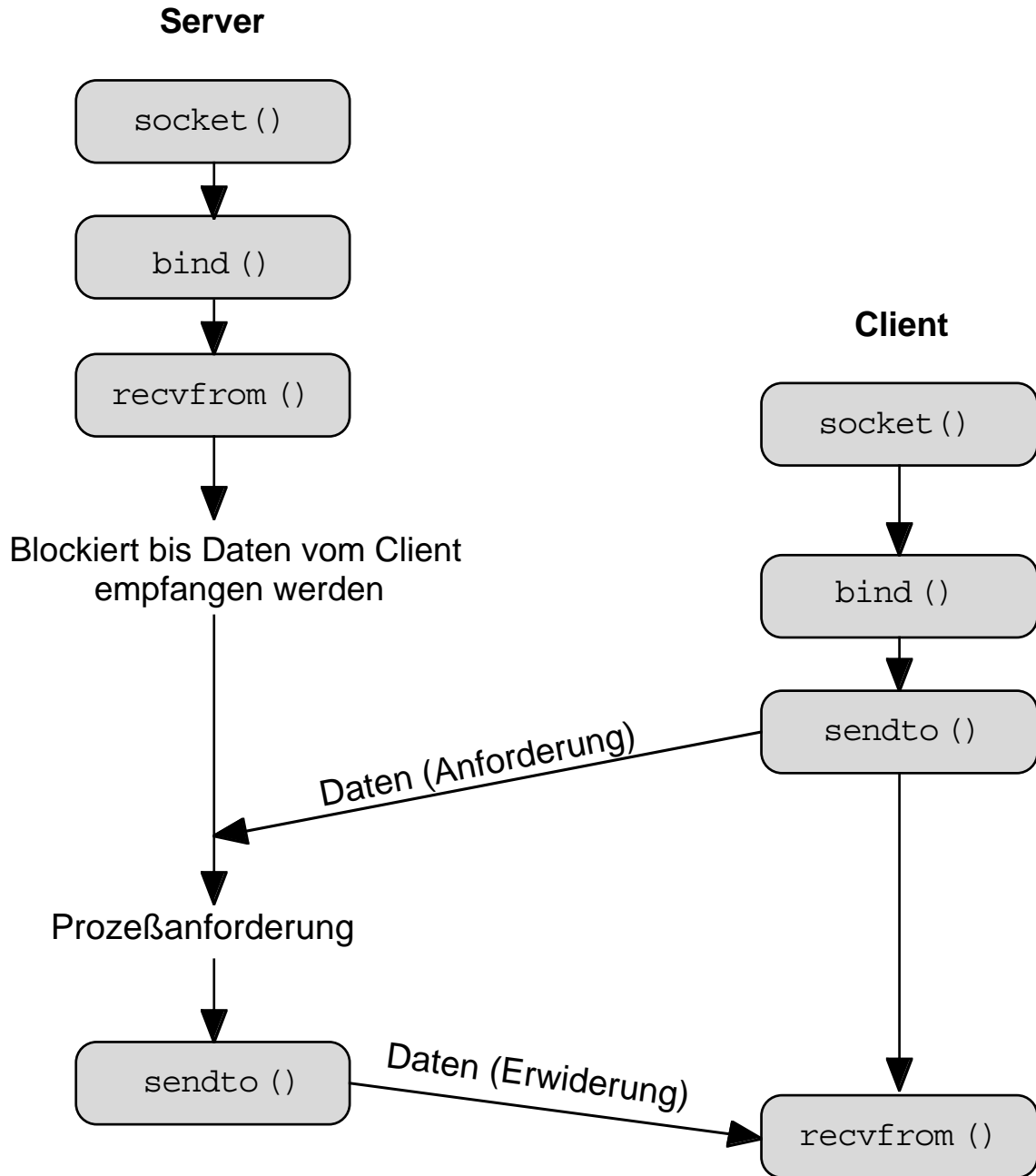


Bild 4.2 Socket-Systemaufrufe bei verbindungslosem Protokoll

4.3 Socket-Adressen

Viele der BSD-Netzsystemaufrufe verlangen einen Zeiger auf eine Socket-Adreßstruktur als Argument. Die Definition dieser Struktur steht im Headerfile `<sys/socket.h>`.

```
struct sockaddr {
    u_short    sa_family; /* Adreßfamilie */
    char      sa_data[14]; /* bis zu 14 Byte für */
} /* protokoll-spez. Adresse */
```

Der Inhalt dieser 14 Bytes der protokollspezifischen Adresse wird bezüglich des Adreß-
typs ausgelegt. Für die Internet-Familie ist folgende Struktur in <netinet/in.h>
definiert:

```
struct sockaddr_in {
    short      sin_family; /* AF_INET */
    u_short    sin_port; /* 16 Bit für Portnummer */
                /* in Netz-Byte Ordnung */
    struct     in_addr sin_addr; /* 32 Bit netid/hostid */
                /* in Netz-Byte Ordnung */
    char      sin_zero[8] /* unbenutzt */
}
```

Eine grafische Darstellung dieser Adreßstruktur sieht demnach so aus:

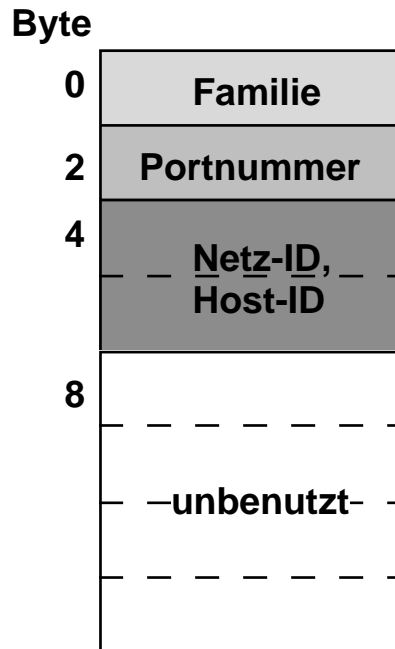


Bild 4.3 Socket-Adreßstruktur der Internet-Adreß-Familie

Die u_short Typendefinition wird in der include Datei <sys/type.h> definiert.

4.4 Grundlegende Systemaufrufe

socket-Systemaufruf

Damit Ein- oder Ausgaben im Netz getätigt werden können, ist der erste Systemaufruf eines Prozesses der `socket`-Systemaufruf. Mit ihm wird der Typ des verwendeten Kommunikationsprotokolls bestimmt (z.B. TCP oder UDP), d.h. der erste der fünf notwendigen Parameter für den Aufbau einer Netzkommunikation.

```
#include <sys/types.h>
#include <sys/socket.h>
int socket (int Familie, int Typ, int Protokoll);
```

Anstelle von Familie setzen wir `AF_INET` ein, da unsere Beispiele nur Internet-Protokolle benutzen.

Die Sockettypen können folgende sein:

<code>SOCK_STREAM</code>	Stream-Socket
<code>SOCK_DGRAM</code>	Datagramm-Socket
<code>SOCK_RAW</code>	Roh-Socket

Das `Protokoll`-Argument im `socket`-Systemaufruf wird für die meisten Benutzeranwendungen auf Null gesetzt.

Der `socket`-Systemaufruf liefert einen Integerwert zurück, einem Dateideskriptor ähnlich. Dieser Wert wird daher als Socketdeskriptor oder `sockfd` (socket file descriptor) bezeichnet.

Anwendungsbeispiel: Ein TCP-Socket soll eingerichtet werden. Als ersten Parameter wird `AF_INET` eingesetzt (für TCP/IP oder Internet-Protokoll-Stapel), als zweiten Parameter wird `SOCK_STREAM` benötigt (da das TCP- und nicht das UDP-Protokoll verlangt werden, und der 3. Parameter ist Null).

```
#include error.c /* eigene Fehlerbearbeitungsroutinen */

int sockfd;
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
err_dump(ERR_NOSOCK);
```

bind-Systemaufruf

Der `bind`-Systemaufruf weist einem noch unbekanntem Socket eine Adresse zu.

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, struct sockaddr *mAdresse, int
AdrLaenge);
```

Das zweite Argument ist ein Zeiger auf eine protokollspezifische Adresse. Das dritte Argument gibt die Größe dieser Adreßstruktur an. Bind wird in drei Fällen angewandt:

1. Server registrieren ihre eigene Adresse innerhalb des Systems. Sie senden an das System: "Dies ist meine Adresse, und jede an diese Adresse gesandte Nachricht ist an mich weiterzuleiten".
2. Ein Client kann eine spezifische Adresse selbst speichern.
3. Ein verbindungsloser Client muß sicherstellen, daß ihm das System individuelle Adressen erstellt, damit das andere Ende (der Server) eine gültige Rückadresse für seine Antwort hat. Diese Korrespondenz ist mit der vergleichbar, mit der ein Briefumschlag mit einer Adresse versehen wird, wenn dabei eine Erwiderung des Adressaten erreicht werden soll.

Anwendungsbeispiel: Ein TCP-Server registriert seine eigene Adressen, und erklärt sich empfangsbereit für beliebige Client-Anfragen. Für die IP-Adresse wird der in `netinet/in.h` definierte Wert `INADDR_ANY` eingesetzt. Dieser 32 Bit-Wert muß mit `htonl()` in Network-Byte-Order gebracht werden. Als Portnummer sollte einen Wert über 1024 angegeben werden, um Kollisionen mit den standardisierten System-Portnummern zu vermeiden. Da die Portnummer eine 16 Bit Größe ist, wird sie zunächst mittels der Routine `htons()` in Network-Byte-Order gebracht. Es ist sinnvoll, vorher den ganzen Datenspeicher zu initialisieren. Dies kann mit der `bzero()` Routine geschehen.

```
#define SERVER_TCP_PORT 2001

int sockfd;
struct sockaddr_in serv_addr;

/* Datenbereich initialisieren */
bzero((char *) &serv_addr, sizeof(serv_addr));
```

```
/* Adressfamilie in Adressstruktur eintragen */
serv_addr.sin_family = AF_INET;
/* IP-Adresse setzen */
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
/* Portnummer setzen */
serv_addr.sin_port = htons(SERVER_TCP_PORT);
/* da für das Setzen der Parameter oben die geeignete
Struktur sockaddr_in definiert wurde, muß für den bind()
Aufruf der zweite Parameter auf die allgemeinere sockaddr
transformiert werden */
bind(sockfd, (struct sockaddr*) &serv_addr,
      sizeof(serv_addr));
```

connect-Systemaufruf

Der Client-Prozeß verbindet (connect) einen Socketdeskriptor mit dem Server. Dies geschieht nach dem socket-Systemaufruf zur Errichtung der Verbindung.

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd, struct sockaddr *ServAdr, int
AdrLaenge);
```

sockfd ist ein Socketdeskriptor, der von einem socket-Systemaufruf zurückgegeben wird. Der zweite und der dritte Parameter sind Zeiger auf eine Socketadresse und deren Größe, wie zuvor schon beschrieben.

Für das verbindungsorientierte Protokoll TCP ergibt der connect-Systemaufruf die aktuelle Einrichtung einer Verbindung von lokalem zum fernen System.

Anwendungsbeispiel: Nachdem ein Client sich ein Socket geöffnet hat, lädt er die Datenstruktur der Serveradresse. Wenn die IP-Adresse in der Dot-Notation vorliegt, kann sie mit der inet_addr() Funktion umgewandelt werden. Die Portnummer des Servers wird zunächst mittels der Routine htons() in Network-Byte-Order gebracht. Es ist wiederum sinnvoll, vorher den ganzen Datenspeicher zu initialisieren.

```
#define SERVER_TCP_PORT 2001
#define SERVER_HOST_ADDR "129.13.35.77"
struct sockaddr_in serv_addr;
```

```
/* ein sockfd bereits wurde eröffnet ... */

/* Datenbereich initialisieren */
bzero((char *) &serv_addr, sizeof(serv_addr));
/* Adressfamilie in Adressstruktur eintragen */
serv_addr.sin_family = AF_INET;
/* IP-Adresse setzen */
serv_addr.sin_addr.s_addr = inet_addr(SERVER_HOST_ADDR);
/* Portnummer setzen */
serv_addr.sin_port = htons(SERVER_TCP_PORT);
/* Verbindung zum Server aufbauen */
connect(sockfd, (struct sockaddr*) &serv_addr,
        sizeof(serv_addr));

/* Datenübertragung ... */
```

listen-Systemaufruf

Dieser Systemaufruf wird von einem verbindungsorientierten Server verwendet, um die Empfangsbereitschaft bezüglich Verbindungen zu signalisieren.

```
int listen(int sockfd, int backlog);
```

Er wird für gewöhnlich nach dem `socket-` und `bind-`Systemaufruf verwendet und unmittelbar vor dem `accept-`Aufruf. Das `backlog`-Argument gibt die Anzahl der möglichen Verbindungsanforderungen wieder, die maximal in die Warteschlange gestellt werden können. Dies geschieht, solange das System auf die Ausführung des `accept-`Aufrufes durch den Server wartet. Dieser Wert wird normalerweise mit 5 angegeben, dem derzeitigen Höchstwert.

Anwendungsbeispiel: Ein Serverprozeß ruft die `listen()` Routine nachdem er mittels `bind()` die von ihm benutzte Portnummer bekanntgegeben hat:

```
listen(sockfd, 5);
```

accept-Systemaufruf

Nachdem ein verbindungsorientierte Server den vorstehend beschriebenen `listen-`Aufruf durchgeführt hat, wartet eine aktuelle Verbindung von einigen Client-Prozessen, bis der Server den `accept-`Aufruf abarbeitet .

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

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *Peer, int
*AdrLaenge);
```

accept nimmt die erste Anforderung in der Warteschlange und *generiert einen weiteren Socket* mit den gleichen Eigenschaften wie sockfd. Wenn keine Verbindungsaufforderungen mehr anstehen, blockiert der Systemaufruf die Warteschlange, bis eine Anforderung ankommt.

Anwendungsbeispiel: Nach einem erfolgreichen accept() startet der Server einen Child-Prozess um die Kommunikation mit dem Client in diesem neuen Prozess zu übernehmen (concurrent Server). Der Parent (initialer Server) kann den neuen Socket schließen und auf weitere Client-Verbindungswünsche warten.

```
int newsockfd, clilen;
struct sockaddr_in cli_addr;

clilen = sizeof(cli_addr);
newsockfd = accept(sockfd, struct sockaddr *) &cli_addr,
                &clilen);
if (newsockfd < 0) err_dump(ERR_NO_NEWSOCK);
if (childpid = fork()) < 0) err_dump(ERR_FORK);
    else if (childpid == 0) { /* Child-Prozess */
        close(sockfd); /* Parent Socket */
        doit(newsockfd); /* bearbeite Client Anfrage */
        exit(0);
    }
close(newsockfd); /* Parent verwirft neuen Socket */
```

send, sendto, recv und recvfrom-Systemaufrufe

Diese Systemaufrufe sind den Standardsystemaufrufen read und write ähnlich, benötigen jedoch zusätzliche Argumente.

```
#include <sys/types.h>
#include <sys/socket.h>
int send(int sockfd, char *Puffer, int nBytes, int Anz);
int sendto(int sockfd, char *Puffer, int nBytes, int Anz,
            struct sockaddr *Ziel, int AdrLaenge);
```

```
int recv(int sockfd, char *Puffer, int nBytes, int Anz);
int recvfrom(int sockfd, char *Puffer, int nBytes, int Anz,
             struct sockaddr *Quelle, int *AdrLaenge);
```

Die ersten drei Argumente `sockfd`, `Puffer` und `nBytes` dieser vier Systemaufrufe sind den ersten drei Argumente von `read` und `write` ähnlich.

Die `Anz` Argumente sind entweder Null oder sind das Resultat einer ODER-Verknüpfung mit einer der folgenden Konstanten.

<code>MSG_OOB</code>	Sende oder empfangen Out-of-Band Daten
<code>MSG_PEEK</code>	Peek auf eine ankommende Nachricht (<code>recv</code> oder <code>recvfrom</code>)
<code>MSG_DONTROUTE</code>	Leite routing um (<code>send</code> oder <code>sendto</code>)

Die `MSG_PEEK` Anzeige erlaubt dem Aufrufenden, die zur Verfügung stehende Daten zu lesen, ohne daß das System nach `recv` oder `recvfrom` Daten abgelegt hat. Auf die anderen Anzeigen soll hier nicht weiter eingegangen werden.

Das Ziel-Argument im `sendto`-Aufruf bezeichnet die protokollspezifische Adresse, an die die Daten gesendet werden. Aufgrund dieser Spezifikation muß die Adreßgröße in `AdrLaenge` angegeben werden. Der `recvfrom`-Aufruf gibt die protokollspezifische Adresse, von der aus die Daten gesendet werden, in `Quelle` an. Die Größe der Adresse wird in `AdrLaenge` angegeben. Es ist zu beachten, daß das letzte Argument im `sendto`-Aufruf ein Integerwert ist, während das letzte Argument in `recvfrom` ein Zeiger auf einen Integerwert ist.

Alle vier Funktionen geben die Länge der Daten zurück, die gesendet oder empfangen wurden.

close-Systemaufruf

Der normale Unix-Systemaufruf `close` wird zum Schließen eines Sockets verwendet.

```
int close(int fd);
```

Wenn der Socket in Verbindung mit einem Protokoll geschlossen wird, muß das System sicherstellen, daß eventuell zu sendende oder noch zu bestätigende Daten im Kernel gesendet werden. Normalerweise kehrt das System sofort nach einem `close` zurück, während der Kernel immer noch versucht, die in der Warteschlange stehende Daten zu senden.

Anwendungsbeispiel: Siehe connect()-Beispiel oben: der Parent Prozeß schließt den neuen, für den Child bestimmten socket.

Byte-Ordnungsroutinen

Die folgende vier Funktionen behandeln die Ordnungsunterschiede von Bytes zwischen unterschiedlichen Computerarchitekturen und verschiedenen Netzprotokollen.

```
#include <sys/types.h>
#include <netinet/in.h>
u_long  htonl(u_long Hostlong);    /* Host -> Netz, lang */
u_short htons(u_short Hostshort); /* Host -> Netz, kurz */
u_long  ntohl(u_long Hostlong);    /* Netz -> Host, lang */
u_short ntohs(u_short Hostshort); /* Netz -> Host, Kurz */
```

In diesen Funktionen wird impliziert, daß short integer 16 Bit und long integer 32 Bit entsprechen.

Byte-Operationen

In den verschiedenen Socket-Adreßstrukturen existieren unterschiedliche Byte-Felder, die behandelt werden müssen. Einige dieser Felder sind, wie auch immer, keine C-Integer-Felder, so daß hier andere Techniken angewand werden müssen, um mit ihnen allen gleich operieren zu können.

Die folgende drei Routinen basieren auf sogenannte "benutzerdefinierten" Byte-Strings (das sind Strings, die im Gegensatz zu Standard-C-Strings, Nullbytes enthalten können).

```
bcopy(char *Quelle, char *Ziel, int nBytes);

bzero(char *Ziel, int nBytes);

int bcmp(char *String1, char *String2, int nBytes);
```

bcopy schiebt die angegebene Anzahl von Bytes von einer Quelle zu einem Ziel. Zu beachten ist, das die reihenfolge beider Zeigerargumente sich von der Standard-Ein-Ausgabefunktion strcpy unterscheidet.

Die bzero-Routine schreibt die angegebene Anzahl von Nullbytes in das Ziel.

bcmp vergleicht zwei beliebige Byte-Strings. Der Rückgabewert ist gleich Null, wenn beide Strings identisch sind, ansonsten ungleich Null. Auch hier unterscheidet sich dies von der Standardfunktion strcmp.

Adreß-Umwandlungen

Eine Internetadresse wird in dem punktierten (dot) Dezimalformat geschrieben , z.B. 129.13.10.1 . Die folgende Funktionen wandeln zwischen diesem Format und der in_addr-Struktur um.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

unsigned long inet_addr(char *Zeiger);

char *inet_ntoa(struct in_addr InAdr);
```

Die erste, inet_addr, wandelt eine Zeichenkette im punktierten Dezimalformat in eine 32-Bit-Internet-Adresse um. Die inet_ntoa-Funktion beschreibt den anderen Weg.

Teil 2: Aufgaben

- 1.a Es soll ein C-Programm erstellt werden, daß die IP-Adresse vom Dot-Format in binär wandelt und hexadezimal ausgibt. Die IP-Adresse wird als Kommandozeilen-Parameter angegeben, z.B.

```
conv_ip 129.13.42.101
```

- 1.b Das nächste C-Programm soll die umgekehrte Konvertierung durchführen. Die IP-Adresse liegt in binärer Form vor (wie in der Struktur `in_addr` aus `netinet/in.h`). Eine C-Funktion soll diesen Wert in die Dot-Darstellung (C-String) umwandeln, und das Rahmenprogramm diese Darstellung ausgeben.

- 1.c Prüfe die Möglichkeiten die C und Unix bieten, um aus einen Hostnamen die IP-Adresse zu ermitteln (optionale Aufgabe).

- 2.a Erstelle eine C-Funktion

```
str_cli(int fdin, int fdout)
```

und das passende Rahmenprogramm um eine Kommandozeile zu kopieren.

- 2.b Zwei Fehlerrountinen sind zu implementieren, gemäß nachstehender Schnittstelle und Aufgabenbeschreibung:

`err_exit(char *str)` gibt den Text `str` als Fehlermeldung aus (auf `stderr`), und beendet den aufrufenden Prozeß.

`err_cont(char *str)` gibt den Text `str` als Fehlermeldung aus (auf `stderr`), und kehrt zum aufrufenden Prozeß zurück.

- 3.a Es ist als Abschluß ein verbindungsorientierten Echo-Client zu erstellen. Der zur Verfügung gestellte Server läuft auf dem Telematik-Host mit Namen Navajo, und erwartet Anfragen von der Portnummer 6666. Der Server liest Zeilenweise aus dem Stream-Socket (max. 512 Zeichen), wandelt jeden Buchstaben in einen Großbuchstaben um (genauer genommen löscht der Server die 2 höherwertigen Bits des Zeichens, d.h Bit 7 und 8 des ASCII Codes), und schreibt diese umgewandelte Zeile zurück in den Socket.

Zur Verfügung gestellt werden:

`inet.h` mit den wichtigsten Deklarationen und Definitionen, bitte unbedingt vorher anschauen!

`readline.c`

`writen.c`

Benutzt werden sollen außerdem die Fehlerrountinen der Aufgabe 2.b und die Routine zum Kopieren der Kommandozeile aus 2.a.

3.b (Optionale Aufgabe)

Erstellen Sie den Server zu 3.a. Zur Verfügung gestellt wird die Routine `str_echo` in der Datei `strecho.c`:

`str_echo(int sockfd)` liest eine Zeile vom Socket und schreibt sie konvertiert zurück.

Literaturverzeichnis

1. Comer, Douglas E. : Internetworking with TCP/IP, Vol. 1: Principles, Protocols, and Architecture. Prentice-Hall Internat. Editions, 1991.
2. Geihs, Kurt : Client/Server-Systeme, Grundlage und Architekturen. Bonn: Internat. Thomson Publ., 1995.
3. Kochan, Stephen G. and Wood, Patrick H. : Unix Networking. Hayden Books, 1989.
4. Krüger, Gerhard : Telematik I. Institut für Telematik, Universität Karlsruhe, Skriptum zum Vorlesung. Studentenwerk.
5. Krüger, Gerhard : Telematik II. Institut für Telematik, Universität Karlsruhe, Skriptum zum Vorlesung. Studentenwerk.
6. Stevens, W. Richard : Programmieren von Unix-Netzen. München Wien : Hanser, 1992.

Anhang: Einige praktische Netzwerk-Tools

Ping

Ping ist ein einfaches aber sehr praktisches Tool um schnell Verbindungen zu testen. Es funktioniert ähnlich einem Sonar: ein Packet wird zu einem entfernten Host geschickt, der es, wie ein Echo, umgehend zurückschickt. Erreicht das Echo dem Sender nicht, dann ist die Verbindung nicht in Ordnung oder der Host hat nicht reagiert (ist nicht erreichbar).

Es handelt sich bei den Packeten um ICMP-Packete (Internet Control Message Protocoll), ein OSI-Schicht 3-Protokoll, das IP Datagramme verwendet.

Syntaxbeispiel:

```
/usr/sbin/ping -c count -s packetsize host
```

count gibt die Anzahl der Pakete, packetsize die Größe in Byte, und host den Zielrechnername oder dessen IP-Adresse. Für eine vollständige Beschreibung wird auf die man-Page verwiesen.

Netstat

Auch Netstat ist ein Tool um mehr über den Zustand einer Verbindung zu erfahren. Für eine vollständige Beschreibung wird wieder auf die man-Page verwiesen.

Syntaxbeispiel:

```
/usr/sbin/netstat [-anr] [-p protocol]
```

- a der Status von IP-Sockets wird mitprotokolliert.
- n Adressen und Portnummern werden numerisch aufgelistet
- r die Routing-Tabellen des Hosts werden aufgelistet
- p protocol nennt das Protokoll, z.B. tcp, udp, icmp

Traceroute

Zeigt den Weg der Pakete zu einem bestimmten Zielrechner. Für eine vollständige Beschreibung wird wieder auf die man-Page verwiesen.

Syntaxbeispiel:

```
/usr/sbin/traceroute host [packetsize]
```

host der Zielrechner

packetsize Packetgröße in Byte (Default Datagrammgröße ist 38 Byte)

Telnet

Auch Telnet, eigentlich als Terminal-Emulation gedacht, kann als Diagnosemittel für Netzverbindungen benutzt werden. Dabei nutzt der Umstand, daß in der Telnet Syntax die Angabe eines Ports (nach der Hostangabe) möglich ist.

```
telnet [host] [port]
```

Wird keine Portnummer angegeben, wird eine Verbindung über die Portnummer 23 aufgebaut. Unter diesem wird bei Unix standardmäßig der Telnet-Deamon angesprochen, und der Login-Prozeß läuft ab (siehe Beschreibung der Portnummer, Seite 8).

So kann z.B. getestet werden, ob der Server der Aufgabe 3a. noch antwortet:

```
apache> telnet navajo 6666
Trying 129.13.35.77...
Connected to navajo.
Escape character is '^]'.
ist das klein
ISTDASKLEIN
wo sind die blanks?
WOSINDDIEBLANKS
^]
telnet> q
Connection closed.
apache>
```

Ist der Server nicht mehr aktiv, sieht eine Kommunikation mit Telnet so aus:

```
apache> telnet navajo 6666
Trying 129.13.35.77...
telnet: Unable to connect to remote host: Connection
refused
apache>
```

Basispraktikum

Kompaktrechner

-A4-

Berkeley Sockets
Literaturverzeichnis

Basispraktikum

Kompaktrechner

Berkeley Sockets

Literaturverzeichnis

-A5-
