

Grundlagen

In diesem Abschnitt werden die Grundlagen erarbeitet, die für das Verständnis verteilter Systeme wichtig sind:

- Systemaufrufe von Betriebssystemen,
- Netzwerke und
- Internet Protokolle.

Die Themen werden nicht in der Tiefe vermittelt, es soll ein für alle Beteiligten gleiches Verständnis der Themen geschaffen werden. Die behandelten Themen sind eigentlich Inhalt von eigenen Spezialveranstaltungen.

Inhalt

| | |
|-----------------------------|----|
| 1. Systemaufrufe..... | 4 |
| 1.1. Prozessverwaltung..... | 7 |
| 1.2. Signale..... | 13 |
| 1.3. Dateiverwaltung..... | 20 |

| | |
|---|----|
| 1.4.Katalogverwaltung..... | 28 |
| 1.5.Schutzmechanismen..... | 30 |
| 1.6.Zeitverwaltung..... | 33 |
| 2.Netzwerkgrundlagen..... | 34 |
| 2.1.Begriffe..... | 35 |
| 2.2.Klassifizierung..... | 39 |
| 2.3.Topologien und Vermittlungsverfahren..... | 43 |
| 2.4.Netzwerktechnologien..... | 45 |
| 2.4.1.Technologien lokaler Netze..... | 45 |
| 2.4.2.MAN-Technologien..... | 52 |
| 2.4.3.WAN Technologien..... | 55 |
| 2.5.OSI Referenzmodell..... | 59 |
| 2.5.1.Übertragungsschicht..... | 64 |
| 2.5.2.Sicherungsschicht..... | 64 |
| 2.5.3.Vermittlungsschicht..... | 65 |
| 2.5.4.Transportschicht..... | 65 |

| | |
|---|----|
| 2.5.5.Sitzungsschicht..... | 66 |
| 2.5.6.Darstellungsschicht..... | 66 |
| 2.5.7.Anwendungsschicht..... | 68 |
| 2.5.8.Nachrichtenaufbau nach OSI..... | 69 |
| 3.Internet Protokollfamilie..... | 71 |
| 3.1.Internetprotokoll..... | 73 |
| 3.1.1.IP-Adressierung..... | 73 |
| 3.1.2.IP-Funktionalität..... | 76 |
| 3.1.3.Versionen..... | 79 |
| 3.2.UDP und TCP..... | 80 |
| 3.2.1.User Data Protocol, UDP..... | 80 |
| 3.2.2.Transmission Control Protocol, TCP..... | 82 |

1. Systemaufrufe

Die vorgestellten Systemaufrufe sind an der **Implementierung in Linux** orientiert. In anderen Betriebssystemen sind die Umsetzungen zwar teilweise anders, das Prinzip ist aber das gleiche.

Systemaufrufe bilden die **Schnittstelle zur Hardware**, auf dem das Betriebssystem abläuft. Deshalb sind große Teile eines Systemaufrufs in Assembler programmiert. Um sie für Programmierer nutzbar zu machen, wird oft eine **C-Bibliothek** bereitgestellt.

Beispiel für Systemaufruf in C/C++: Lesen einer Datei

```
count = read(file, buffer, nbytes);
```

Durch den Systemaufruf werden wird die durch „file“ angegebene Datei gelesen und „count“ Bytes in die Variable „buffer“ gespeichert.

Normalerweise ist „count“=„nbytes“, aber wenn das Dateiende erreicht wird, sind u.U. weniger als „nbytes“ Bytes zum Lesen da. Wenn der Systemaufruf nicht ausgeführt werden kann (Plattenfehler oder Datei nicht lesbar), wird „count“ auf -1 gesetzt und die Fehlernummer wird in einer globalen Variablen „errno“ abgelegt.

Ein **vollständiges C++ Programm** (simpleCat.cpp), das eine Datei liest und auf dem Bildschirm ausgibt, ist nachfolgend gezeigt:

```
#define BUFSIZE 512
#include <unistd.h>
#include <iostream>
using namespace std;
int main(int argc, char** argv)
{
    char buf[BUFSIZE];
    int n;
    while ((n = read(0, buf, BUFSIZE)) > 0)
        write(1, buf, n);
}
```

Das Programm wird von Shellebene aus mit dem Kommando

```
$ cc simpleCat.c -o simpleCat
$
```

übersetzt. Die resultierende ausführbare Datei „simpleCat“ kann nun verwendet werden, um Dateien zu lesen, etwa durch das Kommando:

```
$ simpleCat < simpleCat.c
#define BUFSIZE 512
. .
$
```

Im Folgenden werden nun die wichtigsten Systemaufrufe kurz vorgestellt. Dabei werden sie gemäß folgender Gruppen diskutiert:

- Prozessverwaltung
- Dateiverwaltung
- Katalog- und Dateisystemverwaltung
- Schutzmechanismen
- Zeitverwaltung

1.1. Prozessverwaltung

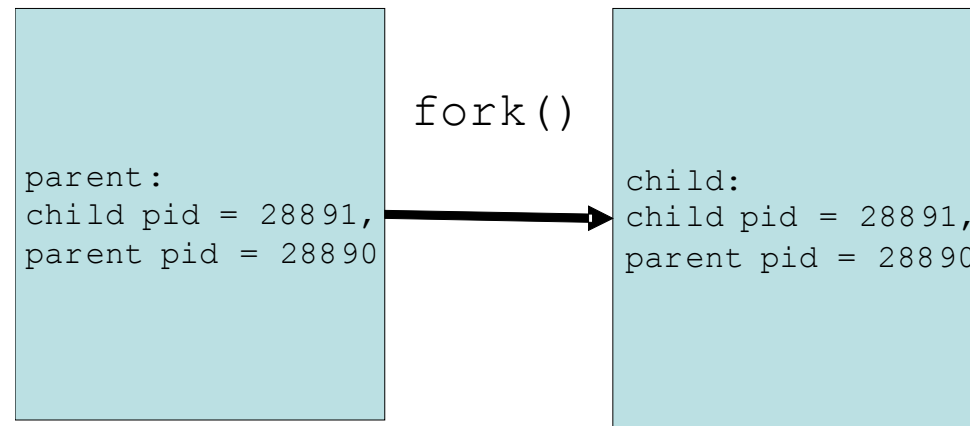
Ein Prozess wird erzeugt, indem ein Eltern Prozess durch den Systemaufruf „fork“ ein Kind Prozess erzeugt. Der Aufruf erzeugt eine exakte Kopie des Originalprozesses (Kind=Clone des Vaters), einschließlich aller Dateideskriptoren, Register, Stand der Befehlszähler usw. Nach dem fork werden der Vater und das Kind unterschiedliche Aktivitäten übernehmen. Zum Zeitpunkt des fork haben alle Variable die gleichen Werte, nach dem fork wirken sich Änderungen der Variablen nur noch im jeweiligen Prozess aus.

Der **fork** Aufruf gibt einen Wert zurück, durch den im Programm unterschieden werden kann, ob der Kode des Kindes oder des Vaters gemeint ist: NULL ist der Kindprozess, Wert größer 0 ist die Prozessidentifikation (pid) des Kindprozesses. Ein Rückgabewert von fork, der kleiner als 0 ist, zeigt an, dass kein neuer Prozess erzeugt werden konnte.

fork.cpp

```
#include <iostream>
#include <unistd.h>
using namespace std;
int main(int argc, char** argv) {
    int childPid;
    if ((childPid = fork()) == -1) {
        cerr << "can't fork" << endl;
        exit(1);
    } else if (childPid == 0) { /* child process */
        cout << "child process: " << " child pid = " << getpid()
            << " parent pid = " << getppid() << endl;
        return 0;
    } else { /* parent process */
        cout << "parent process: " << "child pid = " << childPid
            << " parent pid = " << getpid() << endl;
        return 0;
    }
}
```

Das C-Programm produziert folgende Ausgabe.

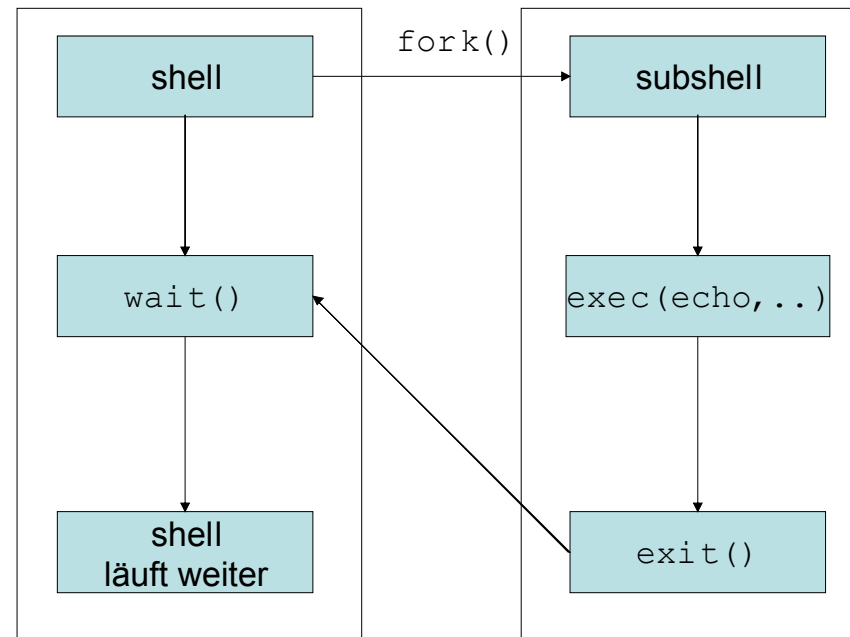


Ausgabe der beiden Prozesse,
die durch Aufruf des C-Programms erzeugt werden

Der „exit“ Systemaufruf beendet die Prozessauführung und gibt den Exitstatus zurück. Die Systemaufrufe „getpid“ und „getppid“ liefern die PID von einem Prozess resp. die PID des eigenen Erzeugers.

Ein reales Beispiel, bei dem ein Prozess erzeugt wird, ist die Shell. Für jedes Kommando, das aus der Shell heraus ausgeführt wird, wird von der Shell ein eigener Prozess erzeugt. Dabei dupliziert sich die Shell, überlagert den eigenen Code mit dem Code des auszuführenden Kommandos und wartet bis der so erzeugte Prozess terminiert.

Dazu sind die Systemaufrufe „exec“ und „wait“ wie im nachfolgenden Schaubild verdeutlicht, verwendet. Dabei wird von der Shell das Kommando echo aufgerufen.



Ausführung „echo“-Kommando durch Shell

Die Shell erzeugt durch `fork` einen Prozess und führt den Systemaufruf „`wait`“ aus. Dadurch wartet sie, bis sie ein Signal erhält. Dieses Signal wird vom „`exit`“ Aufruf des Kindprozesses erzeugt. Der Kindprozess ruft den Systemaufruf „`exec`“ auf. Er nimmt den Code des ersten Parameters (hier das `echo` Kommando) und überlagert den eigenen Code damit. Somit kann in der Umgebung des Kindprozesses, das `echo` Kommando ausgeführt werden. Es existieren mehrere Varianten des `exec` Systemaufrufes. Wir werden sie später genauer kennen lernen.

Das folgende Programm stellt eine Minishell dar, die nach dem o.a. Prinzip funktioniert.

```

#include <iostream>
using namespace std;
void read_command(char *com, char **par){
    cout <<"$ ";
    .....
    return;
}

int main(int argc, char** argv) {
    int childPid, status;
    char command[20];
    char *parameters[60];
    while (1) {
        read_command(command, parameters);
        if ((childPid = fork()) == -1) {
            cerr << "can't fork" << endl;
            exit(1);
        } else if (childPid == 0) {
            /* child */
            execv(command, parameters);
            exit(0);
        } else {
            /* parent process */
            wait(&status);
        }
    }
}

```

„exit“ hat einen Parameter, den so genannten Exitstatus; dies ist ein Integer zwischen 0 und 255. Konvention in Unix ist, dass ein Exitstatus von Null bedeutet, dass die Aktion erfolgreich ausgeführt werden konnte. Jeder andere Wert wird als Fehler angesehen. Dieser Status wird dem Elternprozess in der Variablen „status“ des wait Aufrufs mitgegeben.

Das niederwertige Byte von „status“ enthält 0 für normale Terminierung des Kindes, einen Fehlercode, wenn Fehler aufgetaucht sind. Im höherwertigen Byte ist der Exitstatus des Kindes gespeichert. Der Returncode von „wait“ ist die PID des Kindes.

Soll z.B. eine Kommunikation zwischen Kind und Eltern stattfinden, so kann das Kind z.B. durch

```
exit(4);
```

dem Elternprozess die Nachricht „4“ übergeben. Der Elternprozess wird durch

```
n = wait(status);
```

dann die Information in der Variablen „status“ sehen:

| niederwertige Byte | höherwertige Byte |
|--------------------|----------------------------|
| 0000000000000000 | 0000000000000000 10 |

1.2.Signale

Signale sind das Äquivalent im Bereich Software zu Interrupts im Bereich Hardware. Programme, die als Prozess innerhalb des Betriebssystems ablaufen, müssen unterbrechbar sein, damit z.B. ungewollte Aktionen nicht ausgeführt werden. So kann die Ausgabe (durch das Kommando `cat`) einer großen Datei mit „CTR-C“ abgebrochen werden. Durch „CTR-C“ wird dem Ausgabeprozess ein Signal gesendet. Als Reaktion beendet der Prozess die Ausgabe.

Wenn ein **Signal** zu einem Prozess gesendet wird und der Prozess das Signal nicht annimmt, dann wird der Prozess vom Betriebssystem **automatisch entfernt**. Um sich vor diesem Automatismus schützen zu können, kann sich ein Prozess durch den Systemaufruf „`signal`“ auf das Eintreffen von Signalen vorbereiten. Dazu muss er eine **Signalbehandlungs-Routine** bereitstellen.

Ein Prozess, der eine Signalbehandlungs-Routine bereitgestellt hat, wird bei **Eintreffen eines Signals angehalten**, der **Prozesszustand** auf den **Stack** geschrieben und die **Signalroutine** wird **aufgerufen**. Diese Routine darf selbst beliebige Systemaufrufe veranlassen. Ist sie beendet, wird der Prozess da weiter ablaufen, wo er vorher unterbrochen wurde (der **Zustand** wird vom **Stack restauriert**).

In einem Betriebssystem gibt es mehrere Signalarten. Die meisten Signale werden durch Ereignisse, die von der Hardware ausgelöst werden erzeugt. Die nachfolgende Tabelle listet die wichtigsten Signalarten:

| Nummer | Bezeichnung | Bedeutung |
|--------|-------------|--------------------------------------|
| 1 | SIGHUP | Hang up, Modemunterbrechung |
| 2 | SIGINT | DEL Taste |
| 3 | SIGQUIT | Quit Signal von Tastatur |
| 4 | SIGILL | Nicht erlaubte Instruktion |
| 5 | SIGTRAP | Unterbrechung für Testzwecke |
| 8 | SIGFPE | Gleitkommaüberlauf |
| 9 | SIGKILL | Abbruch |
| 10 | SIBBUS | Busfehler |
| 11 | SIGSEGV | Segmentfehler |
| 12 | SIGSYS | Ungültige Argumente bei Systemaufruf |
| 13 | SIGPIPE | Ausgabe auf Pipe ohne Leser |
| 14 | SIGALARM | Alarm |
| 15 | SIGTERM | Softwareerzeugtes Endesignal |
| 16 | frei | |

Die Signalbehandlungs-Routine sollte als erstes selbst wieder den Mechanismus zur Signalbehandlung aufrufen, damit ein erneut eintretendes Signal ebenfalls abgefangen wird.

Beispiel (signal.cpp): Unendlichschleife, wobei auf „SIGINT“ (CTR-C) mit Ausgabe eines Textes reagiert wird

```
#include <unistd.h>
#include <signal.h>
#include <iostream>
using namespace std;

void handler(int sig) {
    signal(SIGINT, handler);
    cout <<"handler" << endl;
    return;
}

int main(int argc, char** argv) {
    signal(SIGINT, handler); /* CTR-C handled */
    while (1) {
        cout << "main" << endl;
        sleep(2);
    }
}
```

Anstelle eines selbst programmierten Handlers, kann man vordefinierte Handler verwenden. Sie werden durch die Konstante `SIG_IGN` (ignoriere Signal) und `SIG_DFL` (reagiere per Default Aktion) beim Systemaufruf „signal“ verwendet. So ist in der Implementierung der Shell vor dem „fork“ ein Systemaufruf zum ignorieren des Signals `SIGINT`, wenn der Prozess im Hintergrund gestartet werden soll (`signal(SIGINT, SIG_IGN);`).

In Unix gibt es das Kommando „**kill**“ zum Beenden von Prozessen, die im Hintergrund laufen. Wenn ein Programm so geschrieben ist, dass es alle Signale ignoriert, könnte es nie abgebrochen werden. Deshalb gibt es das Signal „`SIGKILL`“. Dieses Signal kann **nicht** per Signalhandler abgefangen werden.

```
$ ps
2864 pts/1      00:00:18  bash
4423 pts/1      00:00:01  signal
4525 pts/1      00:00:00  ps
$
$ kill -9 4423
killed signal
$
```

Im Bereich Echtzeitanwendungen muss ein Betriebssystem in der Lage sein, Prozesse nach einer gewissen Zeit zu informieren, dass bestimmte Dinge zu erledigen sind. In Kernkraftwerken muss die Temperatur des Reaktors regelmäßig überprüft werden. Deshalb wird dem Prozess, der die Überwachung bewerkstelligt regelmäßig ein Signal gesendet, wodurch er die Temperatur prüft. Der Systemaufruf „**alarm**“ hat einen Parameter, der die Anzahl Sekunden angibt, nach denen das Signal „SIGALARM“ erzeugt werden soll. Im Umfeld der **Netzprogrammierung** wird der Systemaufruf z.B. verwendet, um das **ping** Kommando zu realisieren. Dabei wird jede Sekunde ein Datenpaket zu einem Rechner gesendet und gewartet, ob es zurück gesendet wird.

Wenn Programme ablaufen, bei denen eine gewisse Zeit gewartet werden soll, kann dies programmtechnisch realisiert werden, indem man eine Schleife verwendet, die stets nichts tut, als testet, ob die Zeit schon um ist. Diese Lösung ist CPU intensiv.

Besser wäre es, man könnte per Systemaufruf sagen, dass eine gewisse Zeit pausiert werden soll. Dazu ist der Systemaufruf „pause“ verfügbar. Der Systemaufruf „**pause**“ veranlasst den aufrufenden Prozess zu warten, bis er das entsprechende Signal zum Weitermachen erhält.

Das folgende Programm "timer.cpp" verwendet den Systemaufruf „alarm“, um einen Timer zu setzen. Wenn innerhalb von 5 Sekunden keine Benutzereingabe erfolgt, wird das Programm beendet.

```
#include <iostream>
#include <signal.h>
using namespace std;
void handler() {
    cout << "By" << endl;
    exit(0);
}

main() {
    char str[80];
    signal(SIGALRM, handler);

    while (1) {
        alarm(5);          /* start timer */
        cout << "> ";
        gets(str);
        cout << str << endl;
    }
}
```

Ein Aufruf von „timer“ bewirkt etwa:

```
$ timer  
> 12345  
12345  
5 Sekunden warten  
> By  
$
```

1.3.Dateiverwaltung

Das nachfolgende Beispielprogramm „simpeTouch.cpp“ demonstriert den Systemaufruf „creat“. Das Programm legt die als Aufrufparameter anzugebende Datei mit den Zugriffsrechten „0640“ an.

```
#include <fcntl.h>
#include <iostream>
using namespace std;
int main(int argc, char **argv){
    int fd;
    if (argc != 2) {
        cerr << "usage " << argv[0] << " file" << endl; exit(1);
    }
    if ((fd=creat(argv[1],0640)) < 0) {
        cerr << "create error" << endl; exit(2);
    }
}
```

Bevor eine Datei bearbeitet werden kann, muss sie geöffnet werden. Dazu existiert der Systemaufruf „open“, der neben dem Namen noch die Art des Zugriffs (Lesen, Schreiben oder beides) benötigt.

Das folgende Programm („cat.c“) liest die als Parameter anzugebende Datei und schreibt den Inhalt auf die Standardausgabe.

```
#include <fcntl.h>
#include <iostream>
using namespace std;
#define BUFSIZE 512
int main(int argc, char **argv){
    int fd;
    int n;
    char buf[BUFSIZE];

    if (argc != 2) { /* check usage */
        cerr << "usage " << argv[0] << " file" << endl; exit(1);
    }
    if ((fd=open(argv[1], O_RDWR)) < 0) { /* open file */
        cerr << "open error" << endl; exit(2);
    }
    while ((n = read(fd, buf, BUFSIZE)) > 0) /* read and write file */
        write(1, buf, n);
    return 0;
}
```

Der **wahlfreie Zugriff** auf Dateien wird durch den Systemaufruf „lseek“ realisiert. lseek hat drei Parameter:

1. einen Filedescriptor, der die zu bearbeitende Datei definiert,
2. den Offset , der die Position des Lese/Schreibkopfes in Byte relativ zum Ausgangspunkt definiert und
3. den Ausgangspunkt (SEEK_SET=Dateianfang, SEEK_END=Dateiende, SEEK_CUR=aktuelle Position) für die Positionierung des Lese/Schreibkopfes

Damit kann man ein Programm („revCat.cpp“), das eine Datei von hinten nach vorne liest einfach schreiben:

```
#include <fcntl.h>
#include <iostream>
using namespace std;

int main(int argc, char **argv)
{
    int fd;
    int pos;
    char buf[1];
```

```

/* check usage */
if (argc != 2) {
    cerr << "usage " << argv[0] << " file" << endl;
    exit(1);
}
/* open file */
if ((fd=open(argv[1], O_RDWR)) < 0) {
    cerr << "open error" << endl;
    exit(2);
}
/* set position for reading at end of file */
if ((pos=lseek(fd, -1, SEEK_END)) == -1) {
    cerr << "lseek error" << endl;
    exit(1);
}
/* read and write file */
while (pos>=0) {
    read(fd, buf, 1);
    write(1, buf, 1);
    lseek(fd, --pos, SEEK_SET);
}
cout << endl;
return 0;
}

```

Pipes sind ein Kommunikationsmedium, das es erlaubt, dass Prozesse in FIFO Manier kommunizieren. Eine Pipe ist dabei eine Pseudodatei, die die Kommunikationsdaten temporär beinhaltet.

In Unix könne zwei Dateien sortiert in einer zusammengeführt werden durch folgende Kommando-
folge:

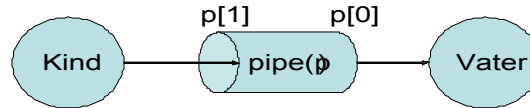
```
$ cat file1 file2 | sort
```

Hierbei werden zwei Prozesse erzeugt: der erste (cat) schreibt seine Ausgabe in die Pipe, der zweite (sort) liest die Standardeingabe aus der Pipe.

Der Systemaufruf „pipe“ erzeugt eine Pipe und gibt zwei Dateideskriptoren zurück, einen zum Lesen und einen zum Schreiben.

```
int p[2];  
pipe(p); /* p[1]=Schreibende, p[0]=Leseende */
```

Der Mechanismus wird nun am Beispiel eines Programms gezeigt, bei dem ein Vaterprozess einem Kind Informationen über eine Pipe sendet, d.h. es wird folgende Kommunikation realisiert:



1. Sohn schließt
Leseende
2. Sohn schreibt

1. Vater erzeugt pipe
2. Vater erzeugt Sohn
3. Vater schließt
Schreibende
4. Vater liest

pipe.cpp:

```

/* Example: Child | Father */
#define BUFSIZE 20
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <iostream>
#include <sys/wait.h>
using namespace std;
int main(int argc, char** argv) {
    int pid, status;
    int p[2];
    char buf[BUFSIZE];

```

```

if (pipe(p) != 0) {
    cerr << "pipe error" << endl;;
    exit(1);
}
switch (pid = fork()) {
    case -1: /* fork failed */
        cerr << "cannot fork" << endl;
        exit(2);
    case 0: /* child: write into pipe */
        { int i;
            close(p[0]); /* close read end of pipe */
            /* create example data */
            for (i=getpid(); i>0; i--) {
                sprintf(buf, "%d\n", i);
                write(p[1], buf, BUFSIZE); /* write into pipe */
                sleep(2); /* just to have more time to see it with "ps -el" */
            }
            close(p[1]);
            exit(0);
        }
    default: /* father: read from pipe */
        { int length;
            close(p[1]); /* close write end of pipe */
            do {
                length=read(p[0], buf, BUFSIZE); /* read data from pipe */
                cout << buf;
            } while (length>0);
        }
}

```

```
    close(p[0]);  
    while (wait(&status) !=pid);  
    exit(0);  
}  
}  
}
```

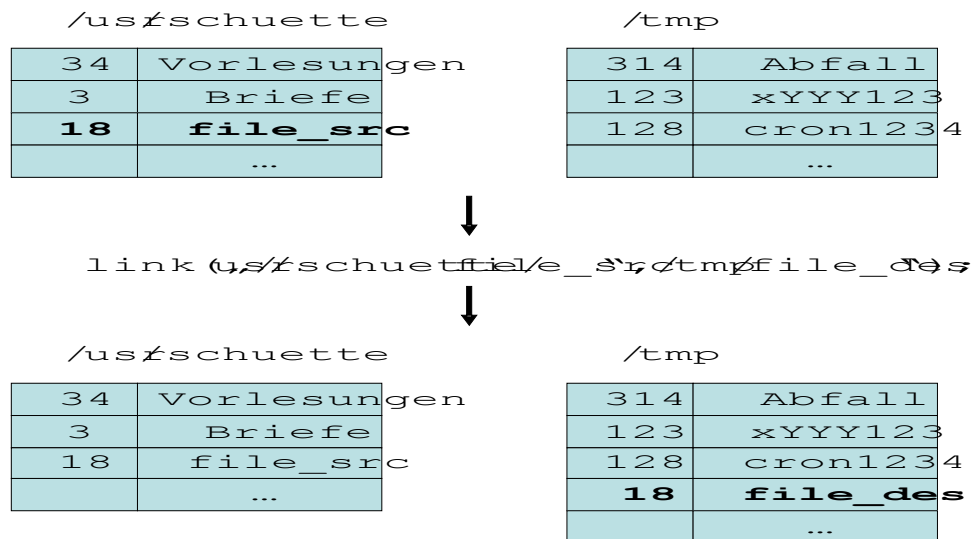
1.4.Katalogverwaltung

Durch den Systemaufruf „**link**“ kann eine physische Datei unter mehreren Namen, auch in unterschiedlichen Verzeichnissen erscheinen. Eine Änderung über den Zugriff mit einem Namen wirkt sich immer auf das reale Dateiojekt aus. Um dies zu realisieren, wird eine Datei in Unix eindeutig identifiziert durch ihre **i-Zahl** (*inumber*). Ein Verzeichnis ist eine Datei, die eine Anzahl von Paaren (*inumber*, Name im ASCII Code) enthält. Durch einen Link wird dann einfach ein neuer Eintrag im Verzeichnis erzeugt, der zur selben *inumber* einen zusätzlichen Namen einführt.

Der Systemaufruf

```
link(„/usr/schuette/file_src“, „/tmp/file_des“);
```

bewirkt folgende Veränderung im tmp-Verzeichnis:



Durch den Systemaufruf „**unlink**“ kann man Links wieder Löschen. Dabei wird nur der Eintrag aus dem Verzeichnis entfernt; die Datei bleibt erhalten. Wenn es keinen weiteren Link auf die Datei gibt, so wird die Datei physisch gelöscht.

Das Ein- und Aushängen von Verzeichnissen wird durch die Systemaufrufe „**mount**“ und „**umount**“ bewerkstelligt.

Um vom aktuellen Verzeichnis in ein anderes Verzeichnis zu wechseln, existiert der Systemaufruf „**chdir**“. Nachdem das aktuelle Verzeichnis mit dem Systemaufruf

```
chdir („/usr/schuetzte/tmp“)
```

geändert wurde, werden die folgenden Aufrufe (create, open), bei denen kein voller Pfadname angegeben ist, immer im tmp-Verzeichnis Dateien anlegen, bzw. öffnen.

1.5.Schutzmechanismen

In Linux besitzt eine Datei 12 Schutzbits in 4 Gruppen:

| Ausführungsmodi | Benutzer | Gruppe | alle anderen |
|-------------------------------------|--------------------------------|--------|--------------|
| Sgd | rwX | rwX | rwX |
| s=setuid g=setgid d=directory | r=read w=write x=execute | | |

Durch den Systemaufruf „chmod“ kann die Zugriffsmaske für eine Datei gesetzt werden. Die Maske wird dabei oktal angegeben. So bedeutet 0644, dass der Benutzer den Zugriff „6=110=rw-“ hat, die Gruppe „4=100=r-“.

Das folgende Programm (simpleChmod.c) setzt die Zugriffsrechte einer Datei (2.Parameter) so wie es die Maske (1. Parameter) angibt.

```

#include <iostream>
using namespace std;
int main(int argc, char **argv) {
    int mask;
    if (argc != 3) {
        cerr << "usage " << argv[0] << "mask file" << endl;
        exit(1);
    }
    sscanf(argv[1], "%o", &mask);
    if ((chmod(argv[2], mask)) < 0) {
        cerr << "chmod error" << endl;
        exit(2);
    }
}

```

Wenn ein Programm gestartet wird, so hat es die Rechte, die der Aufrufer des Programms hat. Deshalb kann ein Programm dem Superuser gehören (etwa das Programm `rm=remove file`), aber der Aufrufer kann nur die Dateien löschen, für die er die Rechte hat. Mit dem „**setuid**“ („setgid“) Bits kann man erreichen, dass ein Programm mit den Rechten des Besitzers (Gruppe) abläuft und nicht wie normal mit den Rechten des Aufrufers. Dies wird benötigt, um zum Beispiel die Passwort-Datei durch die Ausführung des Kommandos „`passwd`“ abzuändern. Dazu hat das Kommando (=Datei) das `setuid`-Bit gesetzt.

Der Systemaufruf „**umask**“ setzt eine Linux-interne Bitmaske, die die Schutzbits beim Anlegen einer Datei maskiert. Wird nach „umask(022) eine Datei angelegt, etwa durch „create(„date“, 0777) so hat die neue Datei nicht die Schutzbits 0777, sondern 0755:

| | | | |
|-----------------------|-----|-----|-----|
| File | 111 | 111 | 111 |
| umask | 000 | 010 | 010 |
| Resultat (file-umask) | 111 | 101 | 101 |

1.6. Zeitverwaltung

In Betriebssystemen muss die Verwaltung von Datum und Uhrzeit durch Systemaufrufe unterstützt werden. Neben dem Abfragen der Systemzeit muss es Aufrufe für das Setzen der Zeit und für die Umstellung von Sommerzeit zu Winterzeit und umgekehrt geben.

In Unix ist Datum und Uhrzeit als Anzahl Sekunden, die seit dem „Unix Urknall“ (1.1.1970 00:00:00) vergangen sind, abgelegt. Alle Datumsangaben, werden so gespeichert und erst in der Anzeige in lesbare Form gebracht. Dazu existiert der Systemaufruf „time“, der die Sekundenanzahl liefert und Routinen, um diese Intergerzahl in ein lesbares Format zu konvertieren.

Das folgende C-Programm („now.cpp“) liefert das aktuelle Datum und die Uhrzeit.

```
#include <time.h>
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
    time_t now;
    now = time(NULL); /* now as no. secs since ZERO */
    cout << ctime(&now) << endl; /* convert into readable form */
    exit(0);
}
```

```
$ now
Thu Nov 15 16:07:55 2001
$
```

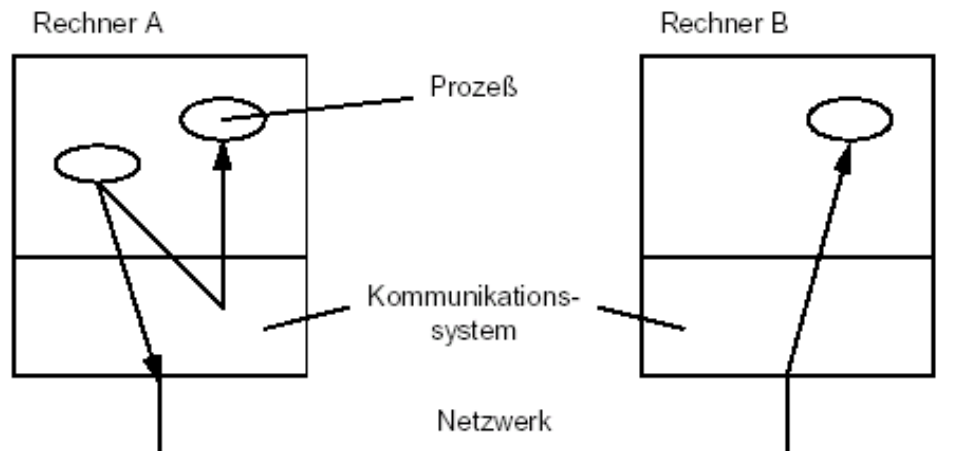
2. Netzwerkgrundlagen

Verteilte Systeme werden auf der Basis von Multicomputern betrachtet. Daraus folgt immer eine Vernetzung von Einzelrechnern.

Die Aufgabe des Netzwerkes ist:

- Übertragung der Informationen von einem Rechner zum anderen.
- Beim Empfänger ist die Information an einen entsprechenden Prozess gerichtet.

Dazu ist ein entsprechendes Kommunikationssystem erforderlich.



2.1. Begriffe

In **Netzen** werden die Daten meist **bit-seriell** übertragen. Innerhalb der **Computer** dagegen **wort-parallel**.

Daraus folgt:

- vor dem Abschicken der Daten ist eine parallel-zu-seriell Konversion,
- nach dem Empfang ist eine seriell-zu-parallel Konversion notwendig.

Da die beteiligten Anwendungsprozesse auf unterschiedlichsten Rechnertypen ablaufen können sind zu beachten:

- Zahlen- und Buchstabenrepräsentation
- Byteordnung
- unterschiedliche Betriebssystemeigenschaften z.B. präemptives Multitasking vs. kooperatives Multitasking.

Für die Prozesse sollen diese Umsetzungs- und Anpassungsverfahren möglichst transparent stattfinden.

Anforderungen bezüglich der **Qualität** der Übertragung spielen bei Verteilte Systeme eine Rolle. Die Qualität, die ein Kommunikationssystem anbieten kann, bezeichnet man als **Dienstgüte** (engl.: Quality of Service, QoS).

Qualitätsparameter sind dabei:

Latenzzeit [s]

Latenzzeit ist die Zeit, die eine leere Nachricht, d.h. eine Nachricht ohne Nutzinformation, vom Sendeprozess durch die Kommunikationsschichten und das Netzwerk bis zum Empfangsprozess braucht.

Datentransferrate [bit/s]

Darunter versteht man die Anzahl der Bits, die maximal zwischen zwei Prozessen pro Sekunde übertragen werden kann.

Nachrichtentransferzeit [s]

Die Nachrichtentransferzeit errechnet sich aus
(Latenzzeit + Nachrichtenlänge / Datentransferrate).

Durchsatz [bit/s]

Anzahl übertragener Bits pro Sekunde über eine gewisse zeitliche Dauer. Hier werden alle Prozesskommunikationen zusammengefasst betrachtet.

Bandbreite [bit/s]

Die physikalische obere Schranke des Durchsatzes, d.h. das Gesamtvolumen, das durch ein Netzwerk potentiell übertragen werden kann.

Verzögerung [bit/s]

Das ist die Zeit, die zwischen dem Abschicken und der Ankunft einer Nachricht verstreicht.

Eine Verbindung über ein Netzwerk ist fehleranfällig. Das Netz muss mit solchen Fehlern umgehen können:

Fehlerkontrolle

Das Netz benötigt eine Instanz, die zum einen Fehler erkennen kann, die so genannte **Fehlererkennung**, und die Fehler durch redundante Datenübertragung möglicherweise beheben kann, die **Fehlerkorrektur**.

Flusskontrolle

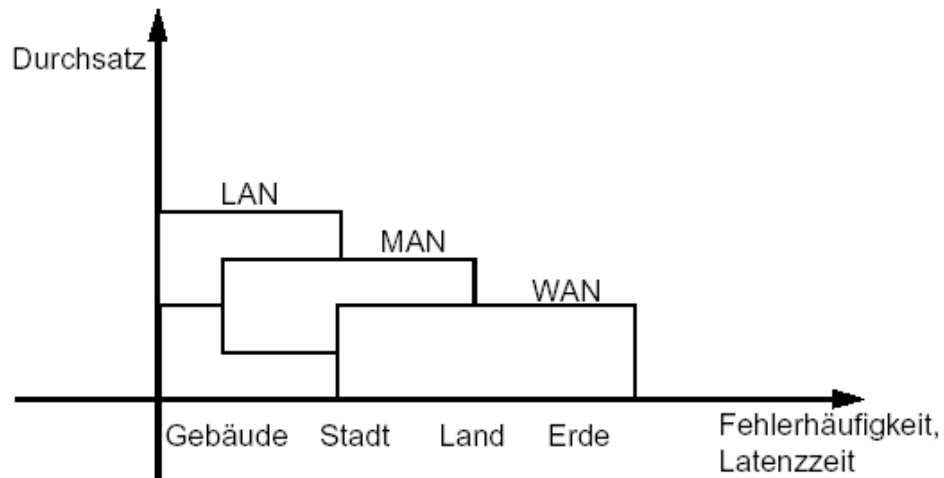
Die Geschwindigkeit, mit welcher der Sender mit seinem Empfänger kommuniziert, muss eventuell angeglichen werden. Die Flusskontrolle kann dies mit Puffern oder durch Ratenkontrolle bewerkstelligen.

Bei verschiedenen möglichen Wegen von Prozess A zu Prozess B entscheidet die **Wegeführung** (engl.: Routing) über den Zustellweg.

2.2.Klassifizierung

Netze kann man nach ihrer Ausdehnung in Kategorien einteilen. Dabei gilt: je weiter ein Netz reicht, desto geringer wird in der Regel dessen Datentransferrate; dagegen nimmt die Fehlerhäufigkeit und Latenzzeit zu.

Klassifikation nach Ausdehnung:



LAN

Ein lokales Netzwerk (engl.: local area network, **LAN**) verbindet **Rechner innerhalb eines Gebäudes**. Einzelne LANs lassen sich untereinander verknüpfen. Man spricht dann immer noch von LANs, nun jedoch mit einzelnen LAN-Segmenten.

MAN

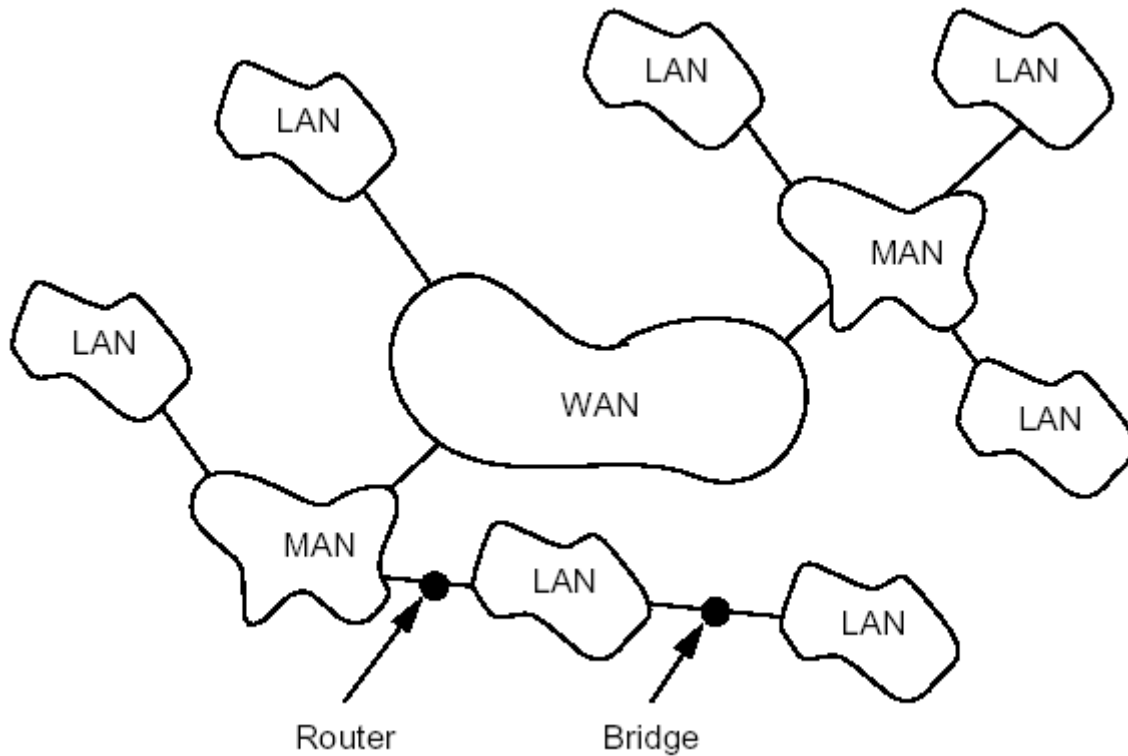
Ein Metropolitan Area Network, **MAN**, wird als **Netz in Stadtgebieten**, Ballungsräumen oder größeren Firmenansiedlungen eingesetzt und verbindet dabei verschiedenste LANs.

WAN

Ein Weitverkehrsnetz (engl.: Wide Area Network, **WAN**) **verbindet** verschiedene **MANs** und **LANs** miteinander.

Internet

Ein Internet ist ein weltweit umspannendes Netz als Verbund verschiedener WANs.



Als Schaltstellen zwischen und in diesen Netzen gibt es verschiedene Geräte, die die Kopplung bewerkstelligen.

Router

Router können **unterschiedliche** Netzwerktechnologien miteinander verbinden. Sie haben Informationen, um Wegelenkung zu realisieren, z.B. Routingtabellen.

Bridges

Bridges verbinden Netze **gleicher** Technologie miteinander und trennen den internen Verkehr in den einzelnen verbundenen Segmenten voneinander ab.

Repeater

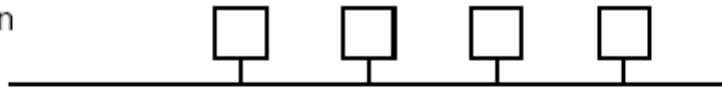
Repeater sind lediglich Verstärker und dienen der Verlängerung einer Netzstrecke gleicher Technologie.

Der Rest dieses Kapitles ist lediglich Wiederholung. Es sollte in den Netzveranstaltungen abgehandelt werden!

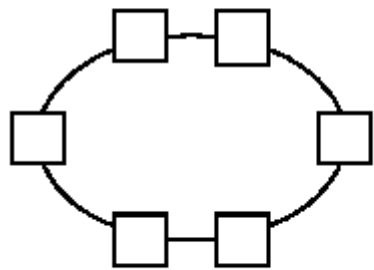
2.3.Topologien und Vermittlungsverfahren

Typische LAN-Topologien sind Busstrukturen(z.B.: Ethernet) oder Topologien mit dedizierten Verbindungen, wie Ringe (z.B.: Tokenring), Sterne oder Gitter.

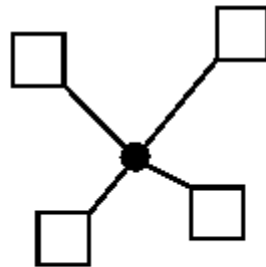
Busstrukturen



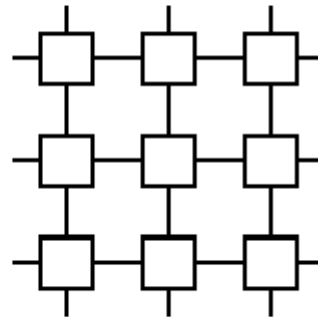
Dedizierte Verbindungen



Ring

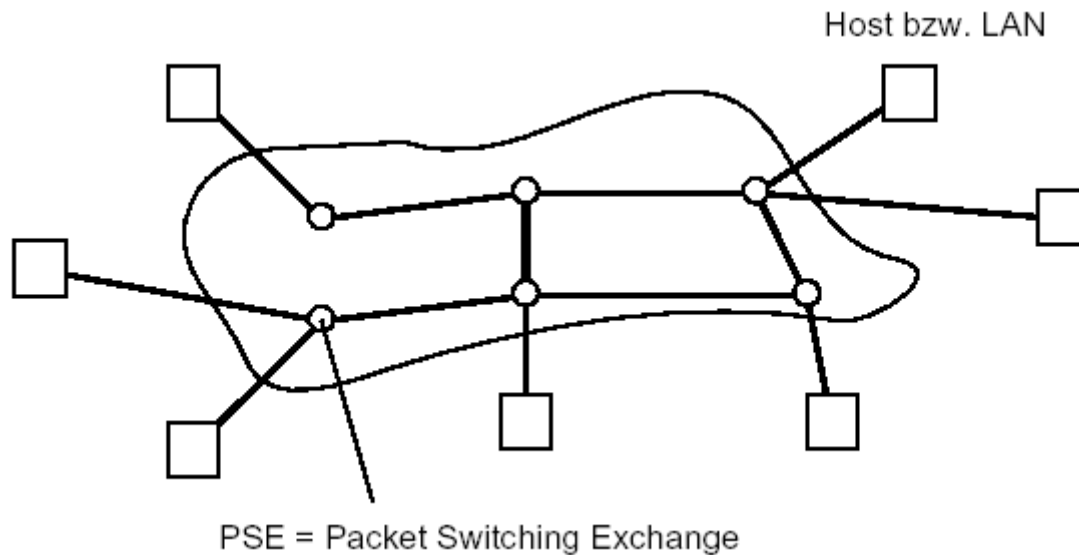


Stern



Gitter

WAN-Topologien haben eine unregelmäßige Vermaschung über Vermittlungsknoten (engl.: packet switching exchange, PSE).



Unabhängig von der Topologie eines Netzes unterscheidet man folgende Vermittlungsverfahren:

Leitungsvermittelt (engl.: circuit switched)

Vor der Übertragung muss eine **Verbindung aufgebaut** werden. Alle übermittelten Daten nehmen dann immer den gleichen Weg. Nach Beendigung der Übertragung wird die Verbindung wieder gelöst.

Paketvermittelt (engl.: packet switched)

Die **Nachricht** wird in **Pakete zerhackt**, die gesondert verschickt werden. Dabei können die Pakete unterschiedliche Wege nehmen. Der Empfänger muss die Pakete wieder zur vollständigen Nachricht zusammensetzen.

Nachrichtenvermittelt (engl.: message switched)

Alle **Pakete**, die zu **einer Nachricht** gehören, nehmen den **gleichen Weg**. Verschiedene Nachrichten können jedoch auch verschiedene Wege nehmen.

2.4. Netzwerktechnologien

Die Technologien, die in den einzelnen Netzwerkkategorien Verwendung finden, werden erläutert.

2.4.1. Technologien lokaler Netze

Ein LAN hat folgende Eigenschaften:

- Verbindungen zwischen Räumen in einem Gebäude oder zwischen einzelnen Gebäuden.
- Die maximale Ausdehnung reicht von ca. 100m bis ca. 10 km (inkl. Repeatern).
- Es sind private Netze, d.h. der Nutzer ist auch der Besitzer des Netzes.
- Die Datentransferrate ist hoch (heute 10 bis 1Gb/s).
- Die Fehlerrate ist sehr niedrig.
- Nachrichten werden bei Bustopologie als Broadcast verschickt.

- Es gibt keine Zwischenspeicherung der Nachrichten oder Pakete im Netz.

Typische LAN-Dienste sind:

- Das gemeinsame Nutzen von Druckern (Druckersharing),
- Das gemeinsame Nutzen von Speicherplatten (Plattensharing).

Durch **Kontrollprotokolle** werden typischerweise folgende Fragestellungen behandelt:

- Was ist die Adresse und damit Identifikation von Sender und Empfänger?
- Wann kann der Sender Daten senden?
- Wie weiß der Empfänger, dass etwas angekommen ist?
- Was passiert, wenn eine fehlerhafte Übertragung stattfand?
- Wie kann der Empfänger die Nachricht interpretieren?

Typische LAN Technologien sind:

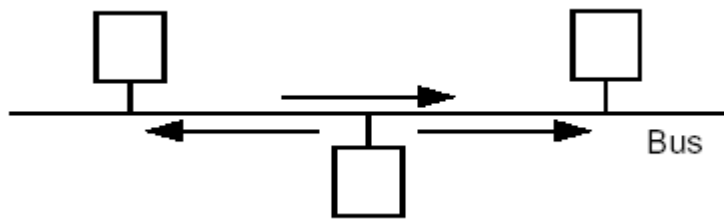
- Ethernet

- Tokenring

Ethernet (IEEE 802.3)

Ethernet wurde bei Xerox PARC 1973 entwickelt und 1985 als IEEE 802.3 standardisiert. Das Kontrollprotokoll ist CSMA/CD (Carrier Sense Multiple Access/Collision Detection).

Ethernet stellt eine Bus Topologie dar.



Das **Kontrollprotokoll CSMA/CD** ist charakterisiert durch:

- Ein sendebereiter Rechner „hört“, ob gerade eine andere Übertragung auf dem Bus stattfindet, d.h. er hört auf eine Frequenz, ähnlich wie das Freizeichen beim Telefonieren (carrier sense).
- Falls die Leitung frei ist, schickt er seine Nachricht als Broadcast eines Paketes ins Netz.

- Empfänger entdecken „ihr“ Paket, indem sie die im Paket eingetragene Zieladresse interpretieren. Diese Interpretation nehmen alle am Bus befindlichen Stationen vor.
- Falls zwei Rechner zur gleichen Zeit gesendet haben (multiple access), entsteht eine Kollision und beide Pakete werden zerstört.
- Dazu überprüft jeder Sender, ob eine Kollision stattgefunden hat, indem er sein Ausgabesignal mit dem Signal auf dem Bus vergleicht (collision detection).
- Falls eine Kollision entdeckt wurde, sendet der Sender ein Jammingsignal, um allen die Kollision mitzuteilen. Alle Sender beenden dann sofort ihre Sendung und warten daraufhin jeweils eine zufällige Zeit ab, um dann erneut mit dem Versenden ihres Paketes zu beginnen.

Ein Ethernet-Rahmen (engl.: frame) hat folgende Gestalt:

| | | | | | |
|----------|-------------|--------------|--------|----------------|-------------|
| 8 Byte | 6 Byte | 6 Byte | 2 Byte | 46 - 1500 Byte | 4 Byte |
| Präambel | Zieladresse | Quelladresse | Typ | Daten | Prüfsequenz |

- Die Präambel dient der Synchronisierung der Empfänger auf den Sender.
- Ziel- und Quelladresse sind weltweit eindeutig vergebene Adressen von Ethernetkarten (MAC Adressen).

- Der Typ gibt an, welches Protokoll zur Übertragung der Nutzdaten verwendet wird.
- Die Nutzdaten werden in variabler Anzahl übertragen.
- Die Prüfsequenz dient der Fehlererkennung.
- Da keine Rahmenlänge übertragen wird und diese variabel ist, müssen die Empfänger das Ende eines Rahmens erkennen können. Dies ist möglich, da zwischen zwei Rahmen mindestens 9,6 Mikrosekunden Intervall bleiben muss.

Ethernet erzielt Auslastungen von 80 bis 95%. Bei über 50% sind Verzögerungen aufgrund von Kollisionen merkbar.

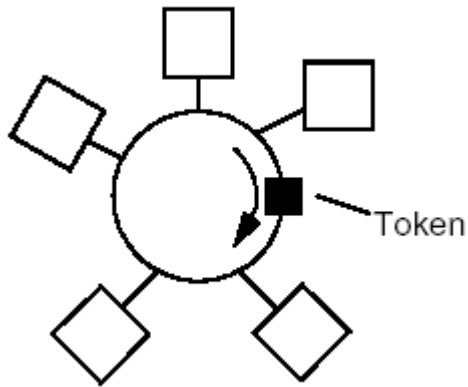
Ein Ethernet kann 500 Meter (ThickWire, 10BASE5) oder 200 Meter (ThinWire, 10BASE2) lang sein.

Ethernets können über Repeater oder Bridges gekoppelt werden, da weltweit alle Ethernetkarten eindeutige Adressen haben.

In letzter Zeit werden Wireless LAN (WLAN) zur Gebäudeverkabelung eingesetzt (IEEE 802.11b mit max. 11 Mbit/s).

Tokenring (IEEE 802.5)

Tokenringe sind in Ringtopologie aufgebaut und haben Datentransferraten von 4 oder 16 Mb/s.

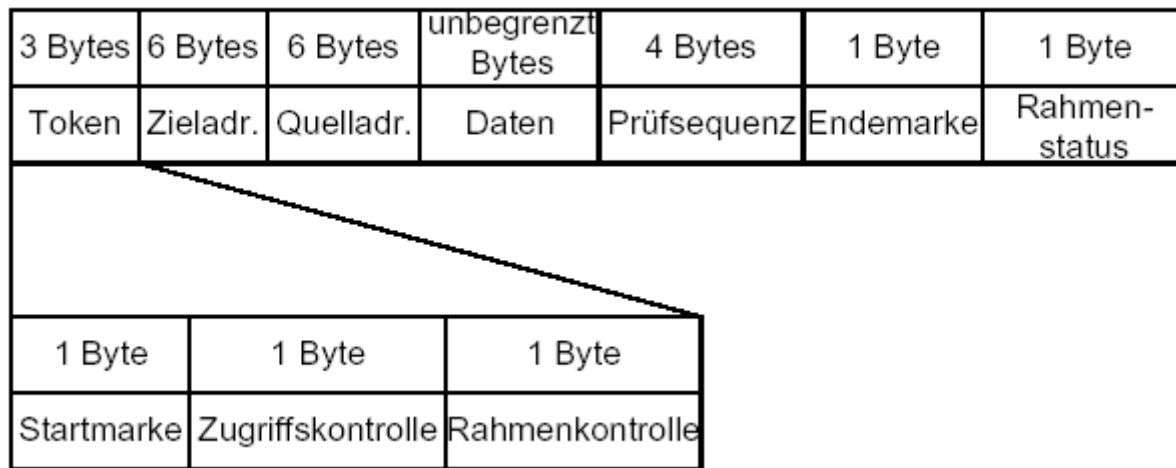


Das Kontrollprotokoll kann wie folgt beschrieben werden:

- Ein Token bewegt sich in einer Richtung im Ring.
- Das leere Token wird vom Sender beim „Vorüberkommen“ erkannt.
- Der Sender hängt die Ziel- und Quelladresse, sowie die Nachricht, an das Token, markiert es als „besetzt“ und schickt es wieder auf die Reise.
- Jede Station prüft das Token, ob die Nachricht für sie bestimmt ist.

- Die Zielstation liest die Nachricht, markiert den Lesestatus des Tokens und schickt es im Ring weiter.
- Wenn die Sendestation nicht mehr senden will, setzt sie das Token auf „frei“, wenn es wieder bei ihr vorbeikommt und schickt es weiter.

Ein Tokenring Frame ist wie folgt aufgebaut:



- Die Startmarke (engl.: starting delimiter) dient zum Erkennen des Rahmenanfangs.
- Die Zugriffskontrolle (engl.: access control) beinhaltet unter anderem die Information, ob das Token „besetzt“ oder „frei“ markiert ist.

- Prinzipiell können beliebig lange Nachrichten im Tokenring übertragen werden. In der Praxis wird jedoch eine Beschränkung auf ca. 2 Kilobyte bei Ringen mit 4 Mb/s bzw. auf ca. 8 Kilobyte bei Ringen mit 16 Mb/s vorgenommen.
- Eine Monitorstation prüft, ob der Ring richtig funktioniert und ersetzt verloren gegangene Token bzw. nimmt doppelte Token vom Ring.

2.4.2.MAN-Technologien

Ein Metropolitan Area Network umfasst ein Gebiet von bis zu 50 km Durchmesser. MANs sind oft Verbindungsstrang (engl.: backbone) zwischen LANs und Vermittlungsstellen.

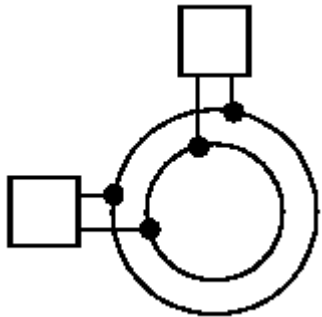
Die Bandbreiten liegen z.Z. in der Größenordnung von 34 – 155 Mb/s.

Wichtige Technologien sind FDDI und DQDB, beide basierend auf Glasfasertechnologien. Glasfaser hat gegenüber Metalleitungen den Vorteil, dass es billig, leicht und zudem nur schwer anzapfbar ist.

Fiber Distributed Data Interface, FDDI

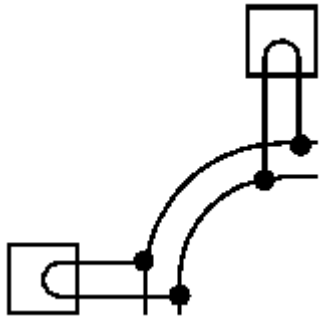
- LEDs dienen als Lichtquelle.
- Die Bandbreite beträgt 100 Mb/s über Entfernungen von bis zu 200 km.
- FDDI wird oft in Ringtopologie als Doppelring verlegt.
- Auf einem Ring wird im Uhrzeigersinn, auf dem anderen gegen
- den Uhrzeigersinn übertragen.

FDDI als Doppelring:



Im Falle der Zerstörung können die offenen Enden (innerhalb zweier angeschlossener Geräte) verbunden und so mit einem Ring weitergearbeitet werden.

Dies führt zu einem FDDI Einzelring:



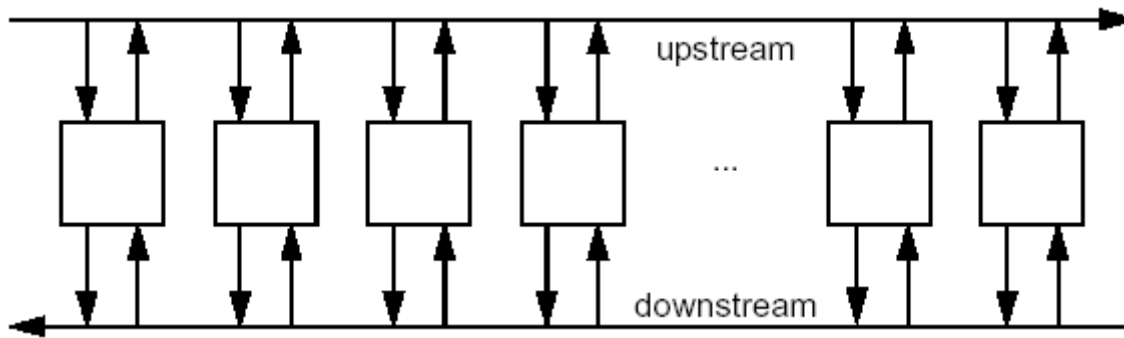
Das Kontrollprotokoll kann wie folgt beschrieben werden:

- Das Verfahren ist ähnlich dem Tokenring-Protokoll, jedoch können mehrere Tokens gleichzeitig im Ring sein.

FDDI wird auch als lokale Netzwerktechnologie und für Backbone-Netze von LANs eingesetzt.

Distributed Queue Dual Bus (IEEE 802.6)

Hier werden zwei Busse eingesetzt: ein Upstream- und einen Downstreambus, typischerweise mit 34 Mb/s in Europa, bzw. 45 Mb/s in USA, über bis zu 160 km Distanz.



Das Kontrollprotokoll kann wie folgt beschrieben werden:

- Will ein Knoten zu einem anderen senden, so muss dies unter Verwendung des entsprechenden Busses geschehen.
- Knoten reserviert sich ein leeres Paket, welche von den äußeren Kopfstationen generiert werden, auf dem anderen der beiden Busse.
- Alle Stationen, die die Reservierung am Bus mitbekommen, merken sich die Reservierung und lassen ein leeres Paket auf dem anderen Bus vorbei. Damit kann die anfragende Station bedient werden.
- Durch dieses Verfahren ist eine verteilte, dezentrale Warteschlangerealisiert.

2.4.3.WAN Technologien

Diese Netze verbinden Städte, Länder und Kontinente miteinander.

Häufig arbeiten sie auf öffentlichen oder gemieteten Leitungen (engl.: leased lines) des öffentlichen Telefonnetzes. Typisch sind heute ISDN-basierte Verbindungen mit $2 * 64$ oder $6 * 64$ Kb/s oder ISDN im Primärmultiplexverfahren mit 2 Mb/s.

Eine spezielle Technologie, die sich insbesondere im Weitverkehrsbereich zu etablieren beginnt, ist ATM.

Asynchronous Transfer Mode, ATM

- ATM ist ein Übertragungsverfahren, das sich mit den entsprechenden Geschwindigkeiten über jedes physikalische Medium übertragen lässt.
- Es verbindet die Vorteile der verbindungsbasierten Übertragung mit denen der Paketvermittlung.
- Standardisierte Geschwindigkeiten des ATM gibt es für Glasfaserleitungen mit 155 Mb/s, bzw. mit 622 Mb/s.

Ablauf

- Der Sender baut eine Verbindung zum Empfänger über mehrere ATM-Vermittlungsstellen (engl.: ATM-Switch) auf. Unterwegs merken sich die Vermittlungsstellen diesen Weg.

- Die Nachrichten werden vom Sender in Zellen zerhackt und entlang dieses Weges geschickt. Der Empfänger setzt die Zellen wieder zur kompletten Nachricht zusammen.
- Nach Beendigung der Übertragung wird die Verbindung wieder abgebaut und die Wegeinformation gelöscht.

Die **ATM-Zelle** umfasst 53 Byte bei 48 Byte Nutzdaten.

| 4 Bit | 8 Bit | 16 Bit | 3 Bit | 1 Bit | 8 Bit | 48 Byte |
|--------------------------|---|--|--------------|-----------------|-------|---------|
| Genetische Flußkontrolle | VPI (Virtueller Pfadidentifikator) | VCI (Virtueller Kanalidentifikator) | Nutzdatentyp | Zellverlustrang | CRC | Daten |

virtueller Kanal:

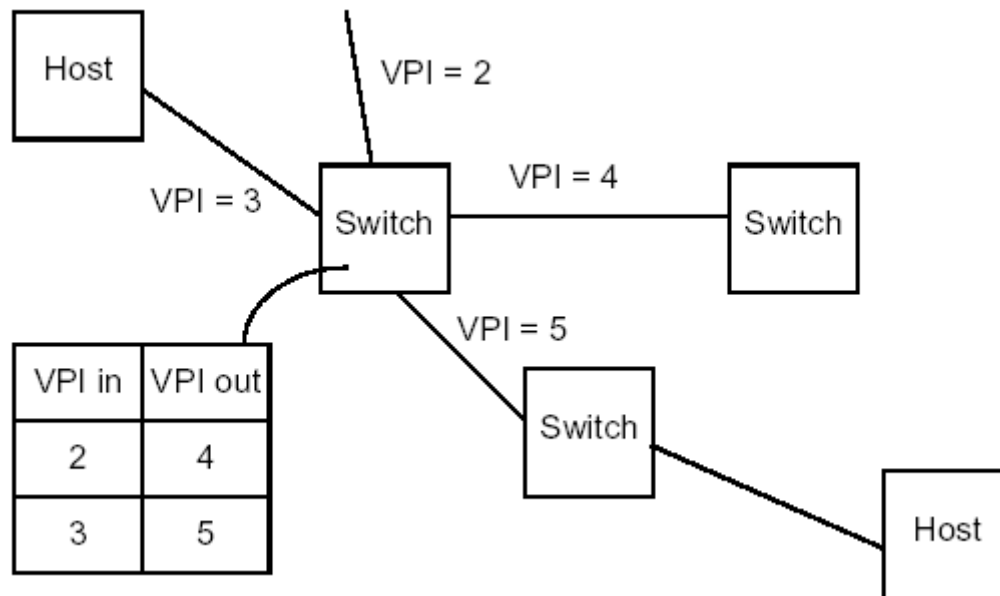
unidirektionale Verbindung zwischen zwei Endpunkten.

virtueller Pfad:

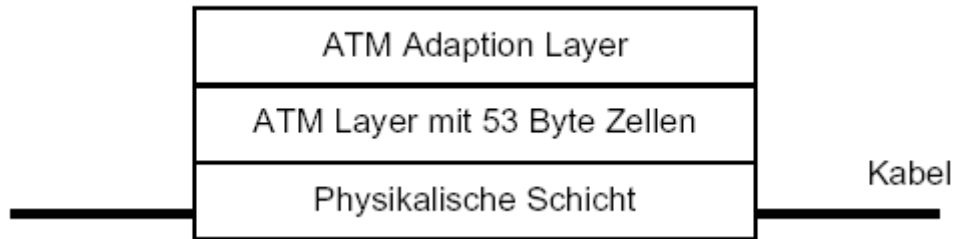
Bündel virtueller Kanäle assoziiert mit einem physikalischen Pfad zwischen zwei Vermittlungsstellen.

Das zweistufige Schema hat den Vorteil, dass in Vermittlungsstellen die Pfadinformation genügt, um die Wegelenkung vorzunehmen.

Damit kann das Switching nach folgendem Schema verlaufen:



Bei 155 Mb/s Übertragungsleistung kann theoretisch ca. alle 3 μ s eine Zelle an einem Switch ankommen, d.h. es gibt ca. 300 000 interrupts/s. Deshalb gibt es den ATM Adaption Layer, AAL. Er ist in das ATM Schichtenmodell wie folgt eingebettet:



Für verschiedene Arten von Datenverkehr sind unterschiedliche AAL-Konzepte standardisiert:

AAL 1: Konstante Bitraten für Audio- und Videoübertragungen.

AAL 2: Übertragung mit variabler Bitrate und beschränkter Verzögerung.

AAL 3/4: Verbindungsloser und verbindungsorientierter Datenverkehr.

AAL 5: Computer-zu-Computer Datenverkehr.

2.5.OSI Referenzmodell

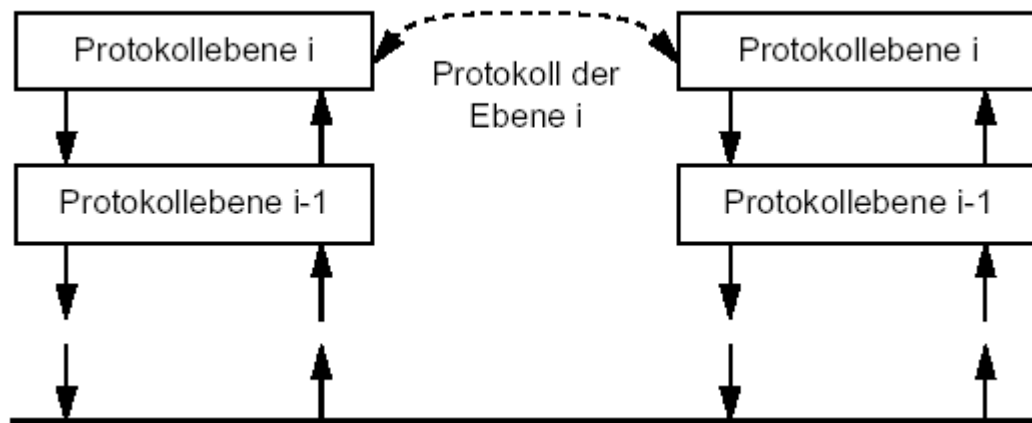
Die **Programmierung** der „nackten“ **Kommunikations-Hardware** ist **kompliziert**. Deshalb nutzt man über der Hardware als **Abstraktionsschicht** ein **Kommunikationssystem**.

Das Kommunikationssystem bietet alle benötigten Kommunikationsmöglichkeiten als Programmierschnittstelle an und bildet diese auf die vorhandenen Netzwerke und deren Technologie ab.

Ein **monolithisch** aufgebautes Kommunikationssystem hat **Nachteile**:

- sehr komplex
- schlecht wartbar
- schwere Anpassung an neue Netzarten.

Deshalb wird ein **Kommunikationssystem** nach einem **Ebenenmodell** mit wachsendem Abstraktionsgrad der einzelnen Protokollebenen und höherer Flexibilität gebraucht.



Jede Protokollebene nutzt die Abstraktion der darunter liegenden Ebene und bietet selbst eine erneute Abstraktionsebene an.

Folgende Begriffe tauchen immer wieder im Zusammenhang mit dem OSI Referenzmodell auf:

Dienst (engl.: service)

Ein Dienst beinhaltet Funktionen, die die Protokollebene $i-1$ für die Ebene i anbietet.

Protokoll (engl.: protocol)

Ein Protokoll umfasst Regeln für den Nachrichtenaustausch zwischen Komponenten gleicher Protokollebenen (diese Komponenten heißen engl.: peers).

Protokollstapel (engl.: protocol stack)

Ein Protokollstapel ist eine Schichtung von Protokollen über mehrere Ebenen hinweg.

Protokollsuite (engl.: protocol suite)

Eine Protokollsuite ist eine Sammlung von geschichteten Protokollen, wobei auf einer Ebene durchaus verschiedene Protokolle verwendet werden können.

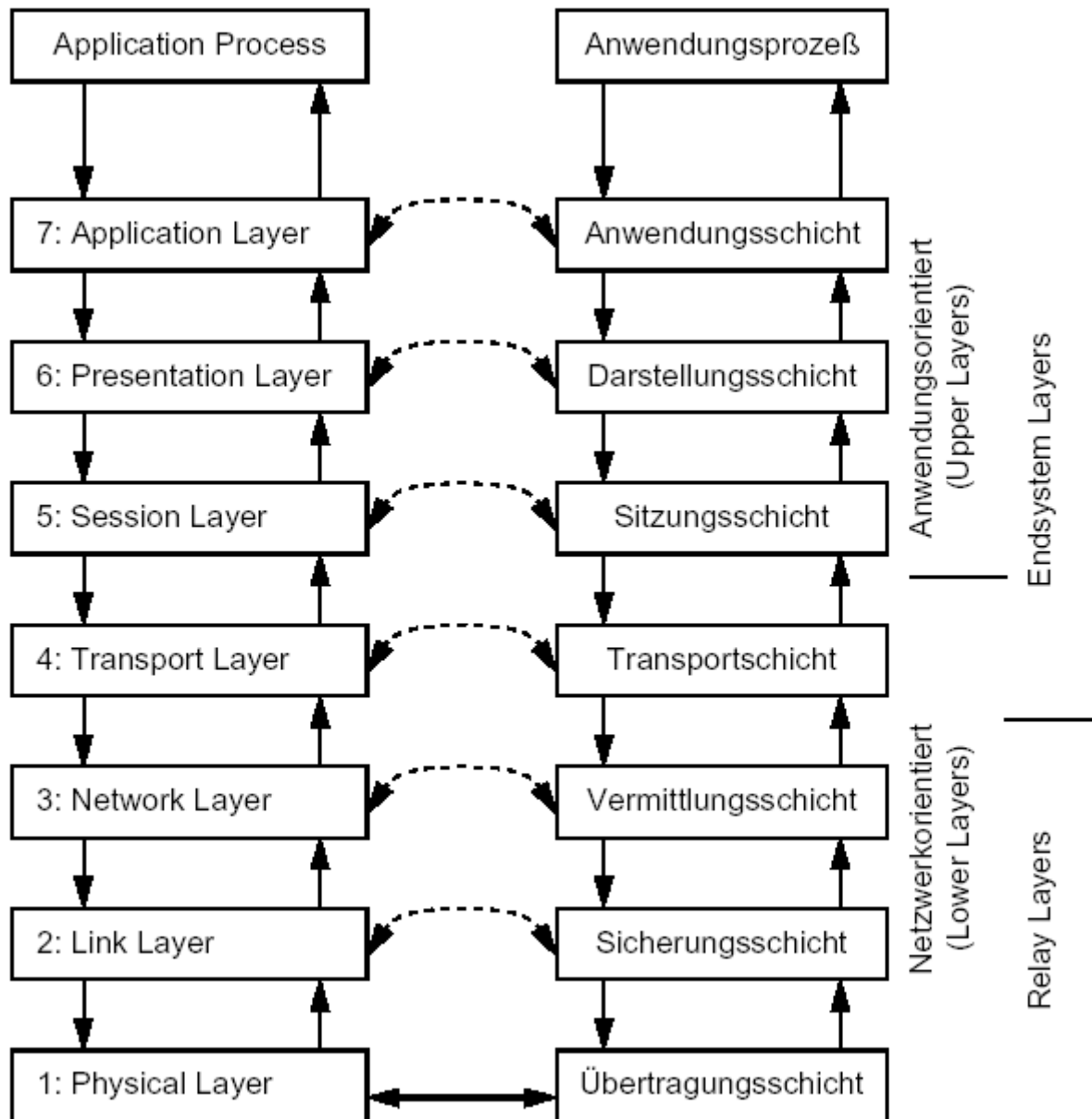
Standard

Ein Standard ist eine Übereinkunft über eine formale Spezifikation von Protokollen und Diensten. Standards sind die Basis für interoperable verteilte Systeme.

Das **International Standards Organization Open Systems Interconnection Referenzmodell** (kurz: OSI-Referenzmodell, 1977) hat sieben Protokollebenen.

Es gibt zwar OSI-konforme Protokolle, das OSI-Referenzmodell ist jedoch meist „nur“ ein Modell, um Struktur und Funktionalitäten eines Kommunikationssystems zu betrachten und einzuordnen.

ISO Referenzmodell für Open Systems Interconnection:



2.5.1.Übertragungsschicht

Hier handelt es sich um ungesicherte Übertragung eines **Bitstroms über das physikalische Medium**.

Die Übertragung erfolgt in Form von

- analogen elektrischen Signalen unter Verwendung von Amplituden- oder Frequenzmodulation über Kabel, oder
- in Form von Lichtsignalen über Glasfaser, oder
- in Form von elektromagnetischen Signalen bei Radio bzw. Mikrowellenübertragung.

Die Charakteristika der Signalformen sind Gegenstand der Standardisierung.

2.5.2.Sicherungsschicht

In dieser Schicht werden **Datenblöcken (Rahmen oder Zellen) über die physikalische Verbindung** zwischen zwei direkt verbundenen Einheiten übertragen.

Aufgaben sind:

- Synchronisation,

- Fehler- und Flusskontrolle, sowie
- eventuelle Nachrichtenwiederholung.
- Zugangsregelung zum physikalischen Medium, falls mehrere Nutzer aus einer höheren Schicht gleichzeitig zugreifen wollen.

2.5.3.Vermittlungsschicht

Sie stellt eine netzwerkweite „Verbindung“ zwischen zwei Protokolleinheitender Transportebene dar. D.h. **Rechner** werden untereinander „**verbunden**“.

Weitere Aufgabe ist die Wegelenkung von Paketen oder Nachrichten.

2.5.4.Transportschicht

Hier handelt es sich um die Abstraktion von spezifischen Eigenschaften der darunter liegenden Netzwerktechnologien und -topologien.

Sie stellt einen transparenten Mechanismus zum **Austausch** von **Daten** zwischen den Endsystemen auf Prozessebene dar.

Zum Nachrichtenaustausch wird ein Satz von Nachrichtentransfer-Primitiven zur Verfügung gestellt.

Es gibt zwei Kommunikationsformen:

Verbindungslose Kommunikation

Nach dem Prinzip des „Best-try“ wird jede Nachricht ohne Garantie für Zustellung und Zustellreihenfolge zum Empfänger geschickt.

Verbindungsorientierte Kommunikation

Zur Übermittlung von Nachrichten wird eine **echte Verbindung aufgebaut**, während der Übertragung bereitgehalten und anschließend wieder abgebaut. Dabei erfolgt Fehlererkennung und Fehlerbehebung auf der gesamten Strecke.

2.5.5. Sitzungsschicht

Die Sitzungsschicht reguliert den Dialog, z.B. durch Vergabe eines „**Rederechts**“.

Manchmal werden Synchronisationspunkte zur Verfügung gestellt. Manchmal gibt es Informationen über den Zustand einer Sitzung.

2.5.6. Darstellungsschicht

Sie stellt ein gemeinsames Format der zu übertragenden Daten zur Verfügung.

Aufgaben:

- Daten werden meist in eine Transfersyntax übersetzt und auf der Empfangsseite in die dort lokale Syntax zurückübersetzt. Dadurch können heterogene Systeme einfacher kommunizieren.
- Eventuelle Verschlüsselung und Kompression.

Beispiele:

- ASN.1 (abstract syntax notation)
- XDR (external data representation).

2.5.7. Anwendungsschicht

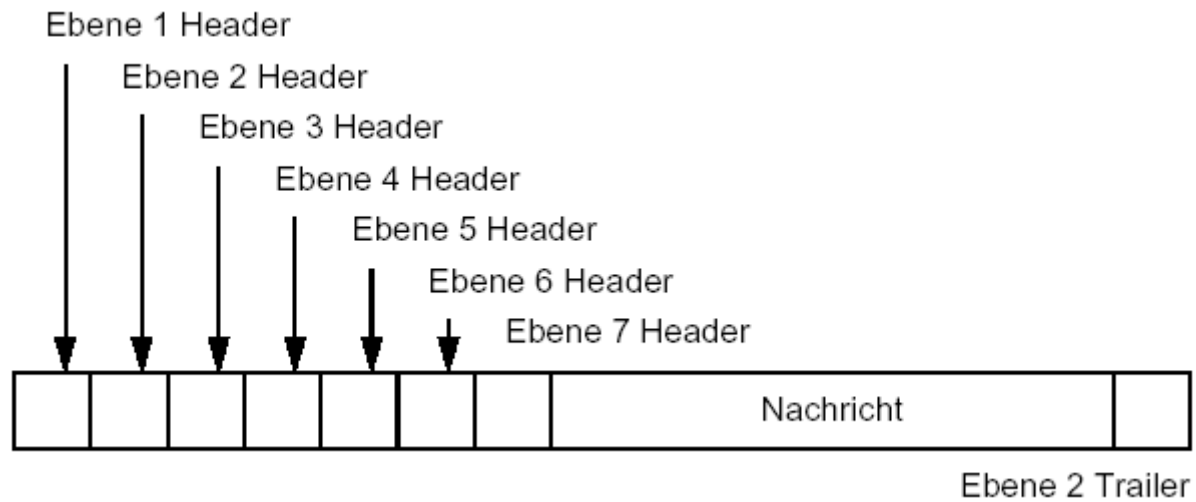
Hier sind spezifische **Anwendungsprotokolle**, die häufig die Schnittstelle zu allgemeinen Diensten sind angesiedelt.

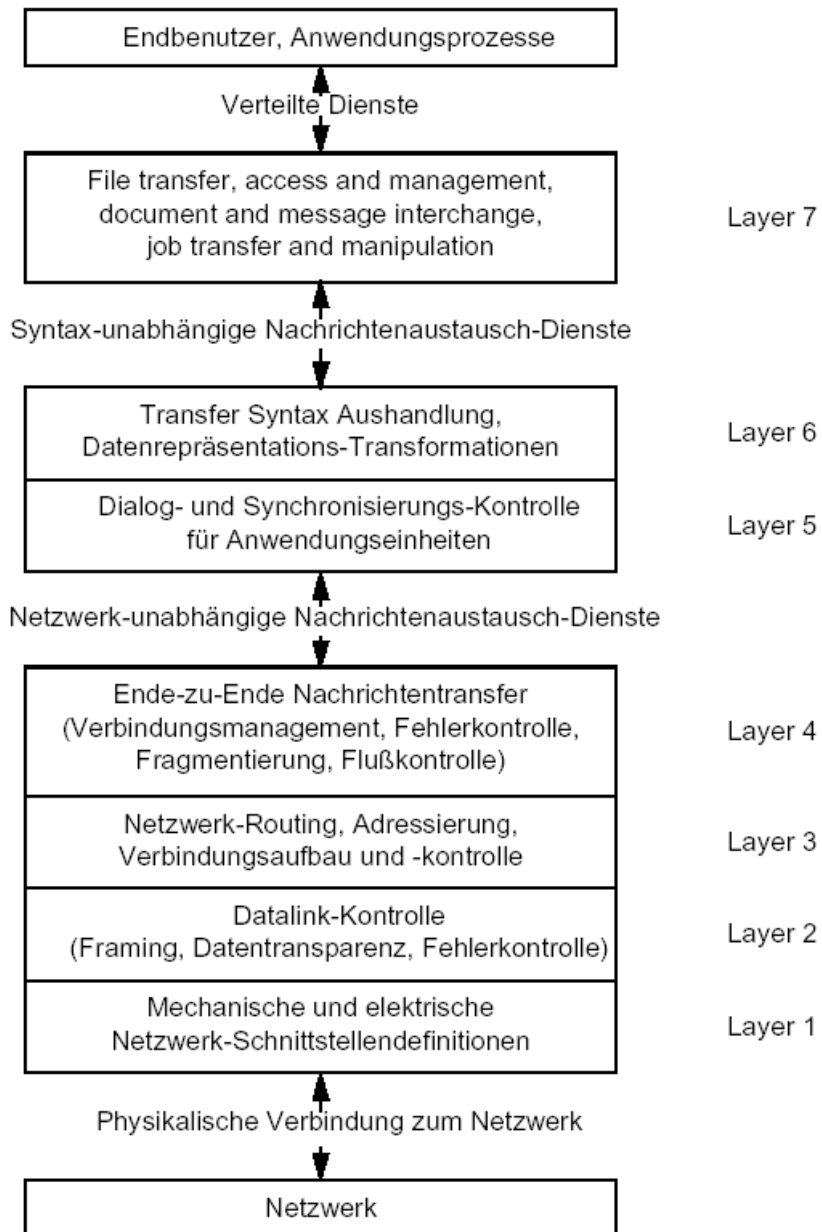
Typische Dienste:

- Dateitransfer (**FTP**),
- Nutzen entfernter Rechner (**Telnet**, rlogin),
- Distributed Transaction Processing (DTP),
- E-Mail (SMTP, X.400),
- Verzeichnis- und Namensdienste (X.500, DNS),
- Datenaustausch, Electronic Data Interchange (EDI),
- Informationsdienste (HTTP des World Wide Web),
- Office Document Architecture (ODA).

2.5.8. Nachrichtenaufbau nach OSI

Den Aufbau einer Nachricht kann man sich wie folgt veranschaulichen:





3. Internet Protokollfamilie

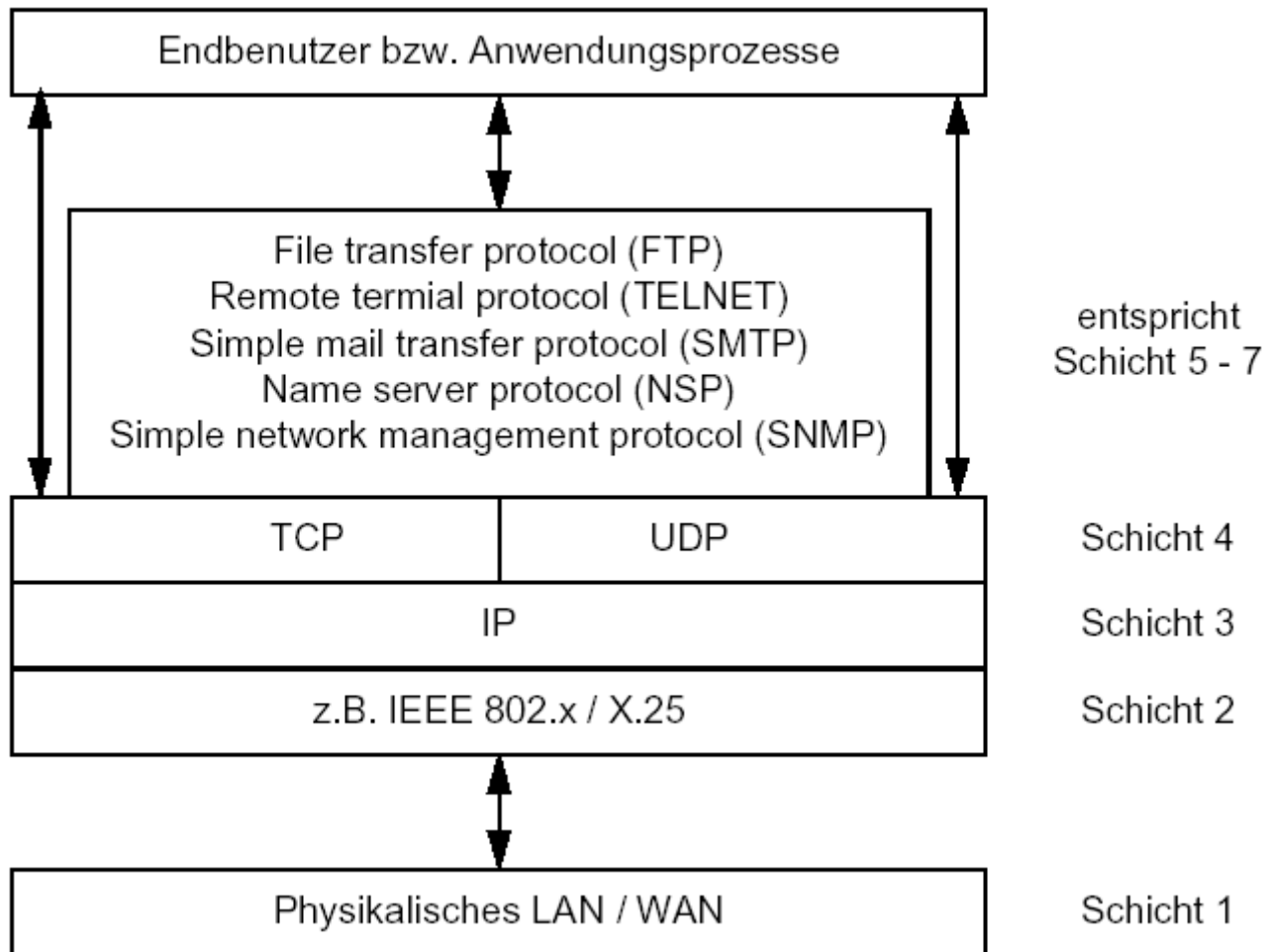
Die Internet Protokollfamilie wird auch **als TCP/IP Protokollsuite** bezeichnet.

Sie entspricht nur in den Schichten 1 - 4 dem OSI-Schichtenmodell.

Die Schichten darüber stellen spezifische Anwendungsprotokolle bzw. -dienste.

Es wird deswegen auch nur ein Upper-Layer-Header an Nachrichten angehängt.

Trotzdem ist die Internet Protokollfamilie Grundlage auch für viele ISO-Standards.



3.1.Internetprotokoll

3.1.1.IP-Adressierung

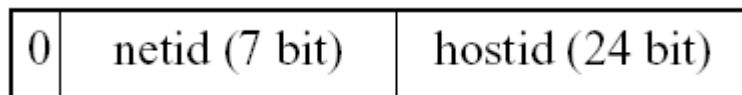
Als Rechneradresse bietet IP drei Adressklassen an.

Jede IP-Adresse besteht aus vier Bytes. Als Darstellung wird häufig die „dotted decimal“-Schreibweise verwendet:

z.B.: | 10000110 | 00111100 | 01000100 | 00001010 | als 134.60.68.10

Die Klassen von IP-Adressen unterscheiden sich in der maximalen Anzahl von Netzsegmenten und den darin maximal befindlichen Rechnern.

Class A



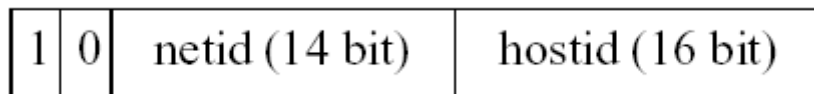
$2^7=128$ Netze mit jeweils bis zu $2^{24}-2$ Hosts

mögliche Wertebereiche:

- netid: 1-127;
- hostid: 0-255.0-255.0-255

Class A Netzadressen sind für sehr große Netzwerke reserviert, wie z.B. für das ARPANET.

Class B



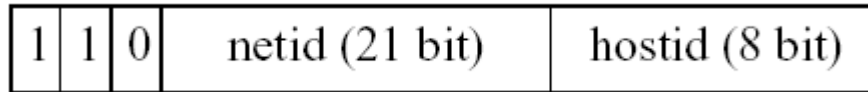
2^{14} Netze mit jeweils bis zu $2^{16}-2$ Hosts

Wertebereiche:

- netid: 128-191.0-255;
- hostid: 0-255.0-255

Class B Netzadressen sind für alle anderen Organisationen reserviert, die Netze mit mehr als 255 Hosts betreiben, wie z.B. Hochschulen.

Class C



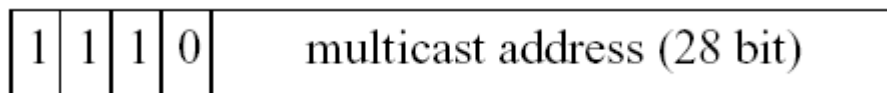
2^{21} Netze mit jeweils bis zu $2^8-2=254$ Hosts

Wertebereiche:

- netid: 192-223.0-255.0-255;
- hostid: 1-254.

Class C Netzadressen sind für alle anderen Organisationen reserviert, die nicht unter Class A oder B fallen.

Durch **Multicastadressen** kann eine Gruppe von Rechnern angesprochen werden, die nicht notwendigerweise in einem Netzsegment liegen müssen:



In einer Multicast-Tabelle wird abgelegt, welche Hosts zur Gruppe gehören.

Zwei festgelegte Multicast-Adressen sind:

- 224.0.0.1: Alle Multicast-fähigen Rechner im Subnetz
- 224.0.0.2: Alle Multicast-fähigen Router im Subnetz

Besondere IP-Adressen

Spezielle Adressen, um Broadcast-Nachrichten zu verschicken, oder um das Netz, d.h. die Router oder Bridges, anzusprechen sind:

- IP-Adresse eines Netzes: hostid = 000...00
- IP-Adresse für Broadcast-Nachrichten: hostid = 111...11

3.1.2.IP-Funktionalität

Die Funktionalität von IP lässt sich wie folgt zusammenfassen:

- IP überträgt Nachrichten (Datagramme) verbindungslos
- Es ist eine Ende-zu Ende Übertragung
- „Best Effort“, d.h. unzuverlässige Zustellung

- Probleme:
 - Paketverluste, Duplikate, vertauschte Reihenfolge
 - Fehler treten auf, wenn in den darunter liegenden Schichten Fehler aufgetreten sind.
- Fragmentierung und Reassemblierung von Datagrammen abhängig von der darunterliegenden Netzhardware.
- Routing der Pakete
 - Address Resolution Protocol, ARP, zur Abbildung von IPAdressen auf Ethernetadressen.
 - Im lokalen Netzsegment reicht diese Information, um Zielrechner direkt zu adressieren.
 - Liegt das Ziel außerhalb, wird die Adresse eines Routers zurückgeliefert.

Das IP Datagrammformat hat folgendes Aussehen:

← 1 Byte →

| | | |
|---------------------------------|---------------|-----------------|
| Version | Header length | Type of service |
| Total length | | |
| Identification | | |
| Fragment offset | | |
| Time-to-live | Protocol | |
| Header checksum | | |
| Source IP address (4 Byte) | | |
| Destination IP address (4 Byte) | | |
| Data (64 kB) | | |

3.1.3. Versionen

Bisher ist im Einsatz die IP Version 4.

Auf Grund des starken Wachstums und neuer Anwendungen im Internet ist **IPv4** an seinen Grenzen gestoßen (Jedes Handy, jeder Kühlschrank braucht demnächst eine IP Adresse)

Deshalb ist **IPv6**, **IPng** verabschiedet und implementiert, aber noch nicht flächendeckend realisiert.

Wesentliche Unterschiede sind:

- 128 Bit Adressen
- Datagrammkopf für effizientere Verarbeitung ausgelegt
- Flow Label zur Unterscheidung spezifischer Nachrichtenströme (bessere Datengüte)

3.2.UDP und TCP

Es existieren zwei **IP-Transportprotokolle** (Schicht 4): **UDP** und **TCP**. Sie dienen der **Adressierung** von **Prozessen** auf Rechnern.

3.2.1.User Data Protocol, UDP

UDP kann wie folgt umschrieben werden:

- es ist ein **verbindungsloser** Datagrammdienst,
- es ist nur **unzuverlässig** Verbindung möglich,
- es gibt keine Garantie auf die Einhaltung der Reihenfolge der Datagramme
- es gibt keine Garantie auf die Vermeidung doppelter Nachrichtenzustellung
- die **Nachrichtengrenzen** werden immer **erhalten** d.h. eine korrekt empfangene und gelesene Nachricht entspricht exakt der gesendeten
- Eine UDP-Protokolleinheit kann mehrere Anwendungen gleichzeitig unterstützen.
- Multicastadressen und Broadcastadressen sind möglich

- Datagrammformat:

| | | | | |
|-------------|------------------|--------|----------|----------|
| 2 Byte | 2 Byte | 2 Byte | 2 Byte | variabel |
| Source port | Destination port | Length | Checksum | Data |

3.2.2. Transmission Control Protocol, TCP

TCP kann wie folgt umschrieben werden:

- es ist ein **verbindungsorientierter** Transportdienst
- **Zuverlässigkeit** ist garantiert
- die **Nachrichten** werden **fehlerfrei** übergeben
- es treten **keine Nachrichtenverluste** oder Duplikate auf
- es existiert eine **selbständige Fehlerkorrektur**
 - verlorengegangene Pakete werden erneut geschickt
 - Duplikate verworfen
 - Sliding-Window-Verfahren mit Sequenz- und Quittungsnummern
- die **Nachrichtengrenzen** werden **nicht erhalten**, da stromorientierte Kommunikation
- eine TCP-Protokolleinheit kann mehrere Anwendungen gleichzeitig unterstützen

- Multicast und Broadcast sind nicht möglich
- TCP-Paketformat

