G Rechnerkommunikation (TCP/IP)

G.1 Überblick

- Ende der 70er Jahre in den USA Entwicklung neuer Netzwerkprotokolle, gefördert durch ***DARPA*** *(Defense Advanced Research Project Agency)*
- daraus entstandene Protokollfamilie: **TCP/IP**
  (nach den Namen der zentralen Protokolle -
  T*ransmission* ***C****ontrol* ***P****rotocol* und ***I****nternet* ***P****rotocol*)
- ab 1983 Standardprotokoll im ARPANet
- in Berkeley Implementierung der Protokolle für UNIX (4.1c bsd)
- ab 1985/86 Einsatz von TCP/IP in den Netzen der **NSF** *(National Science Foundation)*, **NASA** und anderer Institutionen
- in den 90er Jahren Übergang zu kommerziellen Netzprovidern (ISPs)
  (z. B. USA: MCI, Sprint, AT&T, D: EUnet, XLink, MAZ)

G.1 Überblick (2)

- Implementierungen der Protokolle sind heute für alle in für Netzbetrieb relevanten Betriebssysteme verfügbar
- die TCP/IP-Protokolle sind unabhängig von der verwendeten Netzwerktechnologie - verwendbar sind z. B.
  - ◆ Ethernet, Token Ring, FDDI, ATM
  - ◆ X.25-Netze (IP-Pakete werden in X.25-Pakete verpackt)
  - ◆ serielle Leitungen, Modemverbindungen (PPP), ISDN
- dieTCP/IP-Protokolle sind keine Standards im Sinn internationaler Standardisierungsgremien (ISO, CCITT), aber ***de facto Standards***
- Arbeitsberichte, Protokollvorschläge und -standards werden in einer Serie von technischen Berichten veröffentlicht, den ***Requests for Comments - RFC****s*

G.1 Überblick (2)

- IP ermöglicht den Verbund von Netzen,
  das entstandene Gesamtnetz wird als **Internet** bezeichnet
- Größe: 1987: ca. 20.000 Rechner, mehrere hundert Netze
  1994: über 3 Mio. Rechner in 61 Ländern
  1999: ca. 80 Mio. Rechner
  exponentieller Anstieg
- seit Ende der 80er Jahre starke Ausbreitung des **Internet** in Europa (NORDUnet, XLINK, SURFnet, ...)
- in Deutschland zuerst regionale Cluster (Bayern und Baden Württemberg),
  - ➤ ab Ende 1989 bundesweites Internet mit dem WIN (Wissenschaftsnetz) als Transportnetz
  - ➤ seit ca. 1995 kommerzielle Provider

G.2 Einordnung in das ISO/OSI Referenzmodell

7	Anwendungsschicht	*ftp, telnet, rlogin, rsh, ...*
6	Darstellungsschicht	-
5	Sitzungsschicht	
4	Transportschicht	*TCP, UDP*
3	Netzwerkschicht	*IP*
2	Verbindungsschicht	*Ethernet*
1	physikalische Schicht	

G.3 Anwendungen

- Electronic Mail
  - TCP/IP kann - neben anderen Protokollen - zum Transport von E-Mail verwendet werden (eine der häufigsten Anwendungen im Internet)
  - Anwendungsprotokoll: ***smtp*** - ***s**imple **m**ail **t**ransfer **p**rotocol*
  - Benutzerschnittstelle: normale Mailschnittstelle, das Mail-Transportsystem entscheidet selbständig ob TCP/IP benutzt wird
- Dateitransfer
  - ermöglicht Transfer von Dateien zwischen Rechnern
  - Anwendungsprotokoll: ***ftp*** - ***f**ile **t**ransfer **p**rotocol*
  - Benutzerschnittstelle: Kommando ***ftp(1)***
  - unter UNIX außerdem auch *remote copy* (***rcp(1)***)
  - Verschlüsselter Datentransfer: *secure copy* (***scp(1)***)

G.4 Internet Protocol - IP

- Netzwerkprotokoll zur Bildung eines virtuellen Netzwerkes auf der Basis mehrerer physischer Netze
- definiert Format der Datengrundeinheit - IP-Datagramm
- enthält Regeln, wie Pakete verarbeitet und Fehler behandelt werden (Fragmentierung, maximale Lebenszeit, ...)
- unzuverlässige Datenübertragung
- Routing-Konzepte

G.3 Anwendungen (2)

- Dialog
  - ermöglicht interaktiven Dialogbetrieb mit einem anderen Rechner - wie von einem direkt angeschlossenen Bildschirm
  - Anwendungsprotokoll: ***telnet***
  - Benutzerschnittstelle: Kommando ***telnet(1)***
  - unter UNIX außerdem auch
    - *remote login* (***rlogin(1)***)
    - *remote shell* (***rsh(1)*** / ***remsh(1)***)– Ausführung einzelner Kommandos
  - Verschlüsselter Datentransfer: ***ssh(1)***, *slogin(1)*
- weitere Anwendungen
  - World Wide Web (WWW) (***netscape(1)***)
  - Network News (***nn(1)***)
  - Multimedia-Datenübertragung
  - ... ...

1 Adressierung

Grundprinzipien

- einheitliche Adressierung, unabhängig von Netzwerk-Technologie (Ethernet, Token-Ring)
  - ➥ virtuelles Netz: ***Internet***
- Netzwerkeinheitlich (über Router) - transparent für Benutzer
- Aufbau einer IP-Adresse:
  - 4 Byte: **a.b.c.d**, unterteilt in Netzwerk- und Rechneradresse

a < 128:	**a** = Netzwerk, **b.c.d** = Rechner *(Class A Net)* große Organisationen
127 < **a** < 192:	**a.b** = Netzwerk, **c.d** = Rechner *(Class B Net)* Organisationen mittlerer Größe
a > 191:	**a.b.c** = Netzwerk, **d** = Rechner *(Class C Net)* kleinere Organisationen

1 Adressierung (2)

Probleme des Adressierungsschemas

- Mehrere physische Netze in einem Class A oder B Netz
  - ***Subnetting***: Der Rechneranteil der Adresse kann durch sog. ***Subnetmask*** in einen Unternetz- und den eigentlichen Rechneranteil unterteilt werden (Bit-Maske)
  - Nach "außen" wird nur das Class A/B-Netz bekanntgegeben, intern hat man mehrere Netze
- Nur 127 Class A- und ca. 17.000 Class B-Netze, aber über 2 Mio Class-C-Netze ➞ *ROADS*-Problem *(Run Out Of Address Space)*
  - Mehrere Class-C-Netze werden zusammengefaßt (***Supernetting***) — statt einem Class-B-Netz z. B. 256 Class-C-Netze
  - Beim Routing wird der Netzblock gemeinsam behandelt
  - Durch Abschottung von Unternehmen durch Firewalls (nur noch wenige Rechner im Internet sichtbar) ROADS-Problem in den letzten Jahren wieder gebessert

2 Routing

Motivation

- **Routing** bezeichnet die Aufgabe, einen Weg zu finden, über den IP-Pakete geschickt werden
- Prinzipiell gibt es zwei Arten des Routings:
  - **Direktes Routing:** Ein Paket wird direkt von einem Rechner zu einem anderen geschickt (**auf einem physischen Netz**, Grundlage für alle Internet-Kommunikation)
  - **Indirektes Routing:** Der Zielrechner ist **nicht auf demselben physischen Netz**, so daß das Paket über (einen) Router geschickt werden muß

1 Adressierung (2)

IP-Adressen an der Uni Erlangen:

- **REVUE-Netz**
  Class B Netz 131.188
- Subnetmask: 0xFFFFFF00
  (teilweise auch 0xFFFFFFC0)
- Beispiele:
  Informatik Hauptnetz: 131.188.1
  Informatik FFDI-Netz: 131.188.2
  Inf.-CIP Subnetz 1: 131.188.61
  Inf.-CIP Subnetz 2: 131.188.62
  Inf.-CIP Subnetz 3: 131.188.63
  faui01: 131.188.2.1, 131.188.61.1, 131.188.62.1, 131.188.63.1
  faui04a: 131.188.63.10
  faui00d: 131.188.61.13

LRS-Netz 1: 131.188.137.0
LRS-Netz 2: 131.188.137.64
LRS-Netz 3: 131.188.137.128
LRS-Netz 4: 131.188.137.192
halt.e-technik: 131.188.137.6, 131.188.137.65

- zusätzlich Medizin-internes Netz
  UNI_ERL_MED
  Class B Netz 141.69

➥ Rechner mit mehreren IP-Adressen unerwünscht — besser spezielle Geräte (Router) für die Verbindung zwischen IP-Netzen!!!

2 Routing (2)

Direktes Routing

- Der Absender verpackt das IP-Datagramm in ein Paket des physischen Netzes (Encapsulation)
- Die Ziel-IP-Adresse wird in eine Adresse des physischen Netzes abgebildet
- Das Paket wird mittels der Netzwerk-Hardware übertragen
- häufigstes Verfahren: **ARP** *(address resulution protocol)* (auf Broadcast-Basis)

2 Routing (3)

Indirektes Routing

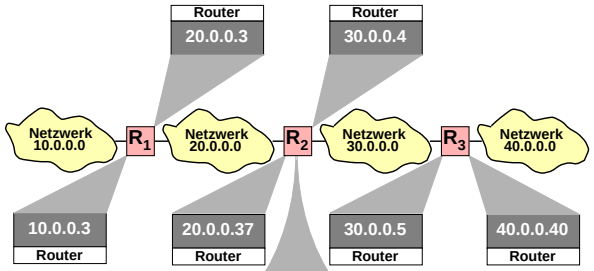
- Beim indirekten Routing muß der Absender das verpackte Datagramm **an einen Router schicken**, der selbst wiederum das Datagramm zum Ziel-Netzwerk weiterleiten muß
- **Router** im Internet bilden eine **kooperative, verbundene Struktur**
- **Datagramme werden von Router zu Router weitergereicht**, bis sie das Ziel-Netzwerk erreicht haben.
- Da sowohl der Absender-Rechner als auch ein Router wissen müssen, welchen Router sie für ein bestimmtes Ziel-Netzwerk benutzen müssen, brauchen sie **Routing-Informationen.**
- Routing-Informationen werden in Rouing-Tabellen gehalten

2 Routing (4)

Routing-Informationen

- Um die **Routing-Tabellen so kompakt** wie möglich zu halten, beziehen die Routing-Informationen sich (im allgemeinen) **nicht auf Ziel-Adressen, sondern auf Zielnetzwerke**
- IP-Software schickt das jeweilige Datagramm **entsprechend dem Netzwerk-Teil der IP-Adresse** an einen bestimmten Router
- Routing-Tabelleneinträge sind normalerweise ein **Tupel (Netw,Router)**
- Um den Unfang der Routing-Tabellen zu begrenzen, kann ein ***Default Router*** angegeben werden, der für alle nicht explizit aufgeführten Netze benutzt wird.

2 Routing (5)



Router und Routing-Tabellen

Routing-Tabelle Router R_2

Ziel-Netzwerk	Route
20.0.0.0	Deliver Direct
30.0.0.0	Deliver Direct
40.0.0.0	30.0.0.5
10.0.0.0	20.0.0.3

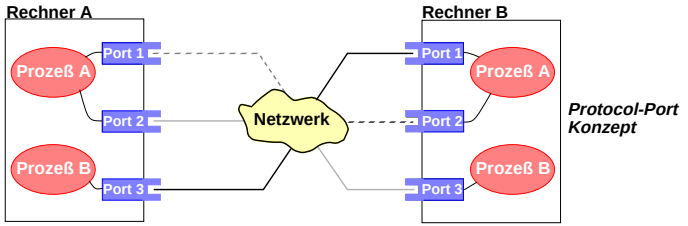
G.5 User Datagram Protocol - UDP

1 Motivation

- bei IP definiert eine Adresse einen Rechner
- keine Möglichkeit, einen bestimmten Benutzer oder Prozeß (Dienst) anzusprechen
- die intuitive Lösung, als Ziel einen Prozeß zu nehmen hat Nachteile
  - ➤ Prozesse werden dynamisch erzeugt und vernichtet
  - ➤ Prozesse können ersetzt werden - die *PID* ändert sich dadurch
  - ➤ Ziele sollten aufgrund ihrer Funktion (Dienst) ansprechbar sein
  - ➤ Prozesse könnten mehrere Dienste anbieten (vgl. *inetd*)

2 *Protocol-Port* Konzept

- *Protocol-Ports* sind abstrakte Kommunikations-Endpunkte
- Pakete, die einem Port zugestellt werden, werden in einer Warteschlange gehalten, bis sie abgeholt werden
- Empfangsanforderungen an einen Port werden blockiert, bis ein Paket eintrifft
- zur Kommunikation mit einem anderen Port müssen **Port-Nummer** und IP-Adresse bekannt sein



G.6 Transmission Control Protocol - TCP

- IP stellt nur unzuverlässige Datenübertragung zur Verfügung
- TCP sichert die Übertragung gegen Fehler (Datenverlust, Paketverdopplung, falsche Reihenfolge)
- TCP versucht die zur Verfügung stehende Netzkapazität möglichst "*freundlich" und optimal* auszunutzen
- das Port-Konzept von UDP ist auch in TCP realisiert
- TCP-Kommunikation hat folgende Eigenschaften
  - ***Stream*-Orientierung** - die Daten werden als Byte-Strom entgegengenommen / abgeliefert
    - unstrukturiert (keine Paketgrenzen!)
    - gepuffert (IP-Pakete werden - wenn möglich - gefüllt)
  - **virtuelle Verbindung** - vor Beginn der Datenübertragung muß eine Verbindung zum Kommunikationspartner aufgebaut werden
  - voll-duplex Verbindung

2 … *Protocol-Port* Konzept (2)

- für eine Reihe von Standard-Diensten (route-Protokoll, rwho, ...) sind Portnummern festgelegt (UNIX → Datei **`/etc/services`**)
- UDP agiert als Multiplexer/Demultiplexer auf IP
- UDP bietet sonst keine zusätzliche Funktionalität gegenüber IP - UDP-Pakete können
  - verloren gehen
  - in falscher Reihenfolge beim Empfänger ankommen

G.7 Systemschnittstelle

1 Sockets — Überblick

- Endpunkte einer Kommunikationsverbindung
- Arbeitsweise: FIFO, bidirektional
- Attribute:
  - **Name** (durch *Binding*)
  - Communication Domain
  - Typ
  - Protokoll

2 Communication Domain und Protokoll

- **Communication Domain** legt die **Protokoll-Familie**, in der die Kommunikation stattfindet, fest
- durch die Protokoll-Familie wird gleichzeitig auch die Adressierungsstruktur **(Adreß-Familie)** festgelegt (war unabhängig geplant, wurde aber nie getrennt)
- das **Protokoll**-Attribut wählt das Protokoll innerhalb der Familie aus
- **Internet Domain**: Communiaction Domain **AF_INET**
  - für Kommunikation über Rechnernetz
  - Protokolle: **TCP/IP** oder **UDP/IP**
  - Namen: **IP-Adressen** und **Port-Nummern**

3 Socket Typen

- **Stream-Sockets**
  - unterstützen bidirektionalen, zuverlässigen Datenfluß
  - gesicherte Kommunikation (gegen Verlust und Duplizierung von Daten)
  - die Ordnung der gesendeten Daten bleibt erhalten
- **Datagramm-Sockets**
  - unterstützen bidirektionalen Datenfluß
  - Datentransfer unsicher (Verlust und Duplizierung möglich)
  - die Reihenfolge der ankommenden Datenpakete stimmt nicht sicher mit der der abgehenden Datenpakete überein
  - Grenzen von Datenpaketen bleiben im Gegensatz zu **Stream-Socket** - Verbindungen erhalten (Internet-Domain Sockets mit UDP/IP)

2 Communication Domain und Protokoll (2)

- **UNIX Domain**
  - für lokale Prozeßkommunikation
  - bidirektionale Pipes
  - Namen aus dem UNIX Dateisystem
    - Eintrag im Dateisystem (Dateityp Socket)
    - verwendbar wie Named Pipes
    - Anlegen und Öffnen nicht mit *mknod( )* und *open( )*, sondern über die normale Socket-Schnittstelle
    - Schließen und Löschen normal über *close( )* und *unlink( )*

4 Client-Server Modell

★ Ein ***Server*** ist ein Programm, das einen Dienst ***(Service)*** anbietet, der über einen Kommunikationsmechanismus erreichbar ist

- Server
  - **akzeptieren Anforderungen**, die von der Kommunikationsschnittstelle kommen
  - **führen** ihren angebotenen **Dienst aus**
  - **schicken** das **Ergebnis zurück** zum Sender der Anforderung
  - *Server* sind normalerweise als normale Benutzerprozesse realisiert
- Client
  - ein Programm wird ein ***Client***, sobald es
    - ➤ eine **Anforderung an einen Server** schickt und
    - ➤ auf eine Antwort wartet

5 Generieren eines Sockets

- Sockets werden mit dem Systemaufruf *socket(2)* angelegt

```
s = socket(Domain, Typ, Protokoll);
```

6 Namensgebung

- Sockets werden ohne Namen generiert
- durch den Systemaufruf *bind(2)* wird einem Socket ein Name zugeordnet

```
bind (s, Name, Namenslänge);
```


7 Verbindungsanforderungen annehmen

- ein Socket wird durch ***listen(2)*** auf Verbindungsanforderungen vorbereitet, die in einer Warteschlange zwischengespeichert werden
- die angeforderten Verbindungen werden einzeln aufgebaut/ zurückgewiesen

```
listen (s, queuelength);
```

8 Verbindungsaufbau und Kommunikation (Stream-Sockets)

- Über einen Socket ist eine Kommunikation zwischen Prozessen möglich:
  - **UNIX-Domain**-Sockets: Kommunikation lokal auf einem Rechner
  - **Internet**-Sockets: Kommunikation über Netzwerk(e)
- Der Kommunikationsaufbau ist asymmetrisch - ein Prozeß agiert als ***Client***, der andere als ***Server***.

8 Verbindungsaufbau und Kommunikation (Stream-Sockets) (2)

- Ablauf:

Client

```
...
connect(s, &server,
        sizeof(server))

...
write(s, buf, 100);
...
read(s, buf1, 20);
```

Server

```
...
listen(s, 5);
s_new = accept(s, from,
         sizeof(from));

...
read(s_new, buf, 100);
...
write(s_new, bufx, 20);
```


9 Stream-Sockets — Systemschnittstelle

- Verbindungsaufbau
  - ***listen(2)*** teilt dem System mit, daß man bereit ist, Verbindungen auf dem Socket entgegenzunehmen
  - ***accept(2)*** nimmt eine Verbindung an und generiert einen neuen Socket ***(s_new***), der für die Kommunikation verwendet werden kann - steht kein Verbindungwunsch an, blockiert *accept*
  - ***connect(2)*** baut die Verbindung vom ***client*** zum ***server*** auf
- Kommunikation
  - ***read(2), write(2)*** verhalten sich wie gewohnt
  - ***send(2), recv(2)*** wie *write( )* bzw. *read(2)* + zusätzliche Funktionalität (Daten ansehen ohne zu lesen, *out-of-band-data, ...).*
- Verbindungsabbau
  - ***close(2)*** Socket schließen
  - ***shutdown(2)*** Elimieren von bereitliegenden Daten vor einem *close( )*

10 Verbindungslose Sockets

- bei einer Kommunikation über Datagramm-Sockets ist kein Verbindungsaufbau notwendig
- Systemaufrufe

  sendto(2) Datagramm senden

  recvfrom(2) Datagramm empfangen
- **Besonderheit:** Im **Internet** Domain können über Datagramm-Sockets ***Broadcasts*** versendet werden.

11 weitere Socket-Systemaufrufe

getpeername(2) Namen der mit dem Socket verbundenen Gegenstelle abfragen

getsockname(2) Namen eines Sockets abfragen

getsockopt(2), setsockopt(2)
Parametrierung eines Sockets abfragen / setzen

12 Adressierung

- Rechner-Adresse (32-bit, ***IP-Address***) und Portnummer
- Include-Datei: `<netinet/in.h>`,
  Struktur *sockaddr_in* :

```
struct sockaddr_in {
    short  sin_family;                  /* =AF_INET */
    short  sin_port;                    /* Port */
    struct in_addr sin_addr;            /* IP-Addr */
    char   sin_zero[8];
};
```

sin_port: Portnummer (1 bis 1023 reserviert!)
ab 1024 frei wählbar
0 = Port wird vom System gewählt

sin_addr: IP-Adresse, mit ***gethostbyname(3)*** zu finden

13 Rechnernamen / IP-Adressen

- IP-Adressen sind für den Benutzer unhandlich
  → Vergabe von Rechnernamen
- Form der Rechnernamen bei größerer Rechnerzahl wichtig
  - Abbildung Rechnername ↔ Adresse muß möglich sein
  - es dürfen keine Namenskonflikte auftreten
- flacher Namensraum
  - Namen sind Sequenz von Zeichen, ohne Strukturierung
  - zentrale Vergabe notwendig
  - bei großem Namensraum unmöglich zu verwalten
- hierarchischer Namensraum
  - vergleichbar mit UNIX-Dateinamen
  - Namensvergabe dezentral möglich
  - Abbildung Name ↔ Adresse ebenfalls dezentral möglich

13 Rechnernamen / IP-Adressen (2)

- dezentrale Abbildung über ***Nameserver***
  - Beispiel: **`faui04.informatik.uni-erlangen.de`**
- Abbildung Rechnernamen ↔ IP-Adressen (Schnittstelle)
- Funktionen: ***gethostent(3)***
  gethostbyname(3)
  gethostbyaddr(3)
- Include-Datei: `<netdb.h>`
  Struktur *hostent* :

```
struct hostent {
    char *h_name;               /* official name of host */
    char  **h_aliases;          /* alias list */
    int   h_addrtype;           /* host address type (AF_INET) */
    int   h_length;             /* length of address */
    char  **h_addr_list         /* list of addresses from
                                   name server */
};
```