

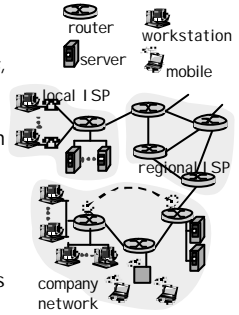
# Internet Praktikum

## 2. Übungsblatt: Socketprogrammierung

Prof. Anja Feldmann, Ph.D.  
 Nils Kammenhuber  
 Arne Wichmann  
 Olaf Maennel

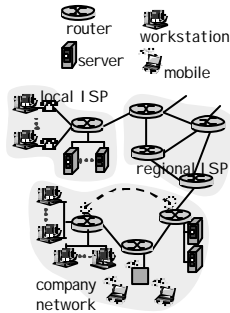
# Was ist das Internet: "nuts and bolts" Blick

- Millionen von verbundenen Geräten: *hosts, end-systems*,
  - PCs, Workstations, Servers
  - PDA's, Telefone, Toaster
 die *Netzanwendungen* laufen lassen
- *Kommunikationslink*
  - Glasfaser, Kupfer, Radio, Sattelliten
- *Routers*: senden von Paketen (chunks) von Daten durch das Netz



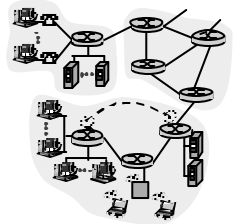
# Was ist das Internet: "nuts and bolts" Blick

- *Protokolle*: kontrollieren Senden und Empfangen von Nachrichten, Meldungen (msgs)
  - z.B., TCP, IP, HTTP, FTP, PPP
- *Internet*: "Netz von Netzen"
  - Mehr oder weniger hierarchisch
  - Öffentliche Internet gegenüber privaten Intranets
- Internet Standards
  - RFC: Request for comments
  - IETF: Internet Engineering Task Force



# Was ist das Internet: Dienst Blick

- *Kommunikationsinfrastruktur* ermöglicht verteilte Anwendungen:
  - WWW, email, Spiele, e-commerce, Datenbanken, Wahlen, ...
  - mehr?
- *Kommunikationsdienste* können sein:
  - Verbindungslos (connectionless)
  - Verbindungs-orientiert (connection-oriented)
- *Cyberspace* [Gibson]:
  - "a consensual hallucination experienced daily by billions of operators, in every nation, ...."



# Was ist ein Protokoll?

## Menschliches Protokoll:

- "wie spät ist es?"
- "ich habe eine Frage"
- Einleitung

... spezifische Nachricht gesandt  
 ... spezifische Aktionen ausgeführt, wenn eine Nachricht erhalten wird oder andere Ereignisse eintreten

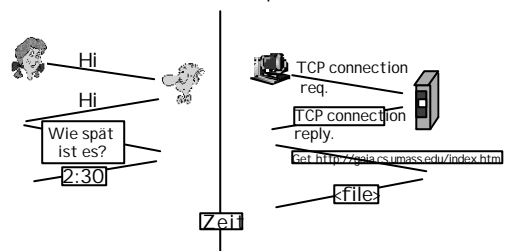
## Netzprotokoll:

- Maschine nicht Mensch
- Alle Kommunikation im Internet wird durch Protokolle festgelegt

*Protokolle definieren Format und Ordnung der Nachrichten, die von Netzelementen gesandt und erhalten werden, und welche Aktionen bei Erhalt einer Nachricht ausgeführt werden.*

# Was ist ein Protokoll?

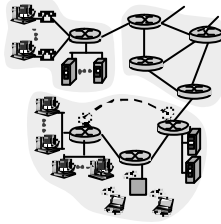
Ein menschliches und ein Computer Protokoll:



E: Andere menschliche Protokolle?

## Genauerer Blick auf die Netzstruktur

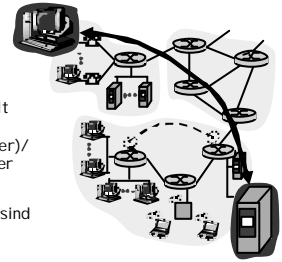
- Netzkannte: Anwendungen und Hosts
- Netzinneres:
  - Router
  - Netzwerk von Netzen
- Zugangnetz, physikalisches Medium: Kommunikationslink



7

## Die Netzkannte:

- Endsysteme (hosts):
  - Anwendungsprogramme
  - e.g., WWW, email
  - at "edge of network"
- Client/Server Model
  - Clienthost verlangt, erhält Dienst vom Server
  - z.B., WWW client (browser)/server; email Client/Server
- Peer-peer Model:
  - Interaktionen der Hosts sind symmetrisch
  - Z.B.: teleconferencing



8

## Netzkannte: Connection-oriented Dienst

Ziel: Datentransfer zwischen Endsystemen

- *Händeschütteln (handshaking)*: für den Datentransfer bereit machen, vor dem eigentlichen Transfer
  - Hallo, hallo im menschlichen Protokoll
  - *Aufsetzen von "Zustand" (state)* in beiden der Kommunikation beteiligten Hosts
- TCP - Transmission Control Protocol
  - Internet's Connection-orientierter Dienst

9

## Netzkannte: Connection-oriented Dienst

TCP Dienst [RFC 793]

- *Verlässlicher (reliable), geordneter (in-order), Bytestrom (byte-stream) Datentransfer*
  - Verluste (loss): Bestätigungen (acknowledgements) und wiederholte Übertragungen (retransmissions)
- *Flusskontrolle (flow control)*:
  - Der Sender wird den Empfänger nicht überfordern
- *Überlastkontrolle (congestion control)*:
  - Der Sender wird die „Senderate reduzieren“ wenn das Netz überlastet ist

10

## Netzkannte: Connectionless Dienst

Ziel: Datentransport zwischen Endsystemen

- Wie zuvor!
- UDP - User Datagram Protocol [RFC 768]: Internet's connectionless service
  - Nicht verlässlicher (unreliable) Datentransfer
  - Nicht geordneter (in-order) Datentransfer
  - Keine Flusskontrolle (flow control)
  - Keine Überlastkontrolle (congestion control)

11

## Dienste der Netzkannten

Anwendungen die TCP benutzen:

- HTTP (WWW)
- FTP (File transfer)
- Telnet (remote login)
- SMTP (email)

Anwendungen die UDP benutzen:

- Streaming media
- Teleconferencing
- Internet telephony

12

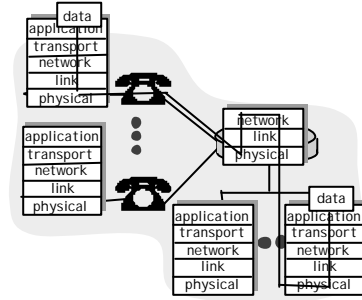
## Protokollschichten "Layers"

### Netzwerke sind kompliziert!

- viele "Teile":
  - Hosts
  - Router
  - Verschieden Medien
  - Anwendungen
  - Protokolle
  - Hardware, Software

13

## Schichtung: (siehe VL)



14

## Anwendungsebene

### Ziele:

- Konzeptionelle und implementationsabhängige Aspekte von Netzwerk Anwendungsprotokolle
  - Client Server Paradigma
  - Dienstmodelle
- Lernen über Protokolle durch betrachten von populären Anwendungsprotokolle

### Weitere Ziele:

- Spezifische Protokolle:
  - http
  - dns
- Programmierung von Netzanwendungen
  - Socket Programmierung

15

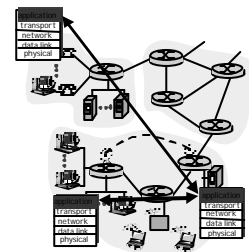
## Anwendungen und Protokolle der Anwendungsebene

### Anwendung: kommunizierende, verteilte Prozesse

- Laufen auf Hosts im Netz im "user space"
- Tauschen Nachrichten aus um die Anwendung zu implementieren
- Z.B. .. Email, File Transfer, das Web

### Protokolle der Anwendungsebene

- Ein "Teil" der Anwendung
- Definiert Nachrichten ausgetauscht durch Anw. und ausgeführte Aktionen
- Benutzerdienste gestellt von den unteren Protokollebenen



16

## Protokolle der Anwendungsebene (2)

### API: Application Programming Interface

- definiert Interface zwischen Anwendung und Transportebene
- Socket: Internet API
  - Zwei Prozesse kommunizieren indem sie Daten in einen Socket senden und Daten aus einem Socket lesen

### Q: wie kann ein Prozess einen anderen mit dem er kommunizieren will "identifizieren"?

- IP Adresse des Host der den anderen Prozess ausführt
- "Portnummer" - erlaubt es dem Host der die Daten erhält festzustellen welcher lokale Prozess die Daten erhalten soll

... in Prof. Feldmann's "Internet Protocols" -VL mehr darüber.

17

## Dienste der Internet Transport Protokolle

### TCP Dienst:

- *connection-oriented*: Aufbau notwendig zwischen Client, Server
- *reliable transport* zwischen sendenden und empfangenden Prozess
- *flow control*: Sender wird den Empfänger nicht überwältigen
- *congestion control*: drosseln des Senders wenn das Netzwerk überlastet ist
- *Stellt nicht zur Verfügung*: Takt (timing), minimale Bandbreitengarantien

### UDP Dienst:

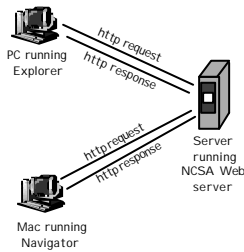
- unreliable Daten Transfer zwischen sendenden und empfangenden Prozess
- *Stellt nicht zur Verfügung*: Verbindungsaufbau, Verlässlichkeit, Flusskontrolle, Überlastkontrolle, oder Bandbreitengarantien

18

## WWW: das http Protokoll

http: hypertext transfer protocol

- WWW's Anwendungsebenenprotokoll
- Client/Server Model
  - Client: Browser das anfragt, erhält, "anzeigt" WWW Objekte
  - Server: WWW Server sendet Objekte als Antwort auf Anfragen
- http1.0: RFC 1945
- http1.1: RFC 2068



19

## Das http Protokoll: weiter

http: TCP Transport Dienst:

- Client initiiert TCP Verbindung (generiert Socket) zum Server, Port 80

http ist "zustandslos"

- Server speichert keine Daten über vorherige Client Anfragen
- Server akzeptiert TCP Verbindung vom Client
- http Nachricht (Anwendungsebeneprotokollnachricht) wird ausgetauscht zwischen dem Browser (http Client) und dem WWW Server (http Server)
- TCP Verbindung wird geschlossen

20

## http Beispiel

Annahme der Benutzer gibt folgende URL ein (enthält Text, Referenzen zu 10 jpeg Bildern)

- 1a. http Client initiiert TCP Verbindung zum http Server (Prozess) auf www.someSchool.edu. Port 80 ist Standardport für http server.
- 1b. http Server auf Host www.someSchool.edu wartet auf TCP Verbindungen auf Port 80. "akzeptiert" Verbindung, benachrichtigt Client
2. http Client sendet http Request message (mit URL) in den TCP Verbindung's Socket
3. http Server erhält Request message, bildet Response message mit angefragtem Objekt (someDepartment/home.index), sendet Message in den Socket

Zeit

21

## http Beispiel (2)

4. http Server schließt TCP Verbindung
  5. http Client erhält Response message mit der html Datei, zeigt html an. Parsen der html Datei, ergibt 10 referenzierte jpeg Objekte
  6. Wiederholung von Schritten 1-5 für jedes der 10 jpeg Objekte
- Zeit
- Nicht-persistente Verbindungen: ein Objekt in jeder TCP Verbindung
    - Einige Browser benutzen mehrere TCP Verbindungen gleichzeitig - eine pro Objekt
  - Persistente Verbindungen: mehrere Objekte können über eine TCP Verbindung transferiert werden

22

## http Nachrichtenformat: Request

- Zwei Typen von http Nachrichten: Request, Response
- http Request Message:
  - ASCII (Menschen-lesbares Format)

request line (GET, POST, HEAD commands)

header lines

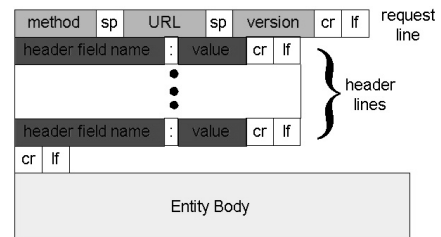
Carriage return, line feed (extra carriage return, line feed) indicates end of message

```

GET /somedir/page.html HTTP/1.1
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: fr
    
```

23

## http request Nachricht: generelles Format



24

## http Nachrichtenformat: Reply

```

status line
(protocol
status code
status phrase)
header
lines
data, e.g.,
requested
html file
    
```

```

HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 .....
Content-Length: 6821
Content-Type: text/html
data data data data data ...
    
```

25

## http Reply Statuscode

In der ersten Linie der server->client Response Nachricht Ein paar Beispielcode:

### 200 OK

- Request erfolgreich, angefordertes Objekt in der Nachricht

### 301 Moved Permanently

- Angefordertes Objekt bewegt, neuer Ort ist in der Nachricht spezifiziert (Location:)

### 400 Bad Request

- Request Nachricht wurde vom Server nicht verstanden

### 404 Not Found

- Angefordertes Dokument wurde auf dem Server nicht gefunden

### 505 HTTP Version Not Supported

26

## Selber ausprobieren von http (client side)

1. Telnet zu Ihrem favorisierten WWW Server:

```
telnet www.eurecom.fr 80
```

Öffnet TCP Verbindung zum port 80 (Default http Serverport) bei www.eurecom.fr. Alles was getippt wird wird an port 80 bei www.eurecom.fr gesandt.

2. Eintippen eines GET http Request

```
GET /~ross/index.html HTTP/1.0
```

Indem man diese eintippt (drucke enter zweimal), sendet man diesen minimalen (aber kompletten) GET Request zum http Server

3. Beachte die Response Nachricht, die vom http Server gesandt wird!

27

## Socket Programmierung in Perl

Ziel: lernen wie man eine Client/Server Anwendung schreibt, die über Sockets kommuniziert

### Socket API

- eingeführt in BSD4.1 UNIX, 1981
- Explizit generiert, benutzt und losgelassen von den Anwendungen
- Client/Server Paradigma
- Zwei Typen von Transportdiensten stehen über die Socket API zur Verfügung:
  - Unverlässliche Datengramme
  - Verlässliche, byte strom-orientierte

### socket

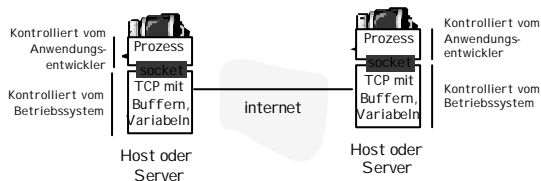
eine host-lokale, Anwendungsgenerierte/ besitzende, OS-kontrolliertes Interface (eine "Tür") in die Anwendungsprozesse von oder an einem anderen (entfernten oder lokalen) Anwendungsprozess sowohl Nachrichten senden als auch empfangen können

28

## Socket Programmierung mit TCP

Socket: eine Tür zwischen Prozess und End-End-Transport Protokoll (UCP or TCP)

TCP Dienst: verlässlichen Transfer von Bytes von einem Prozess zum Anderen



29

## Socket Programmierung mit TCP

Client muss den Server kontaktieren

- Server Prozess muss zuerst laufen
- Server muss zunächst den Socket (Tür) generiert haben, so dass der Kontakt vom Client willkommen ist

Client kontaktiert den Server über:

- Generierung eines lokalen Client TCP socket
- Spezifizierung der IP Adresse und Portnummer des Serverprozesses

- Wenn Client den Socket generiert: Client TCP etabliert die Verbindung zum Server TCP
- Wenn vom Clienten kontaktiert, Server TCP generiert neuen Socket damit der Serverprozess mit dem Client kommunizieren kann
  - Erlaubt dem Server das Sprechen mit mehreren Clienten

### Anwendungsblickwinkel

TCP stellt verlässlichen, in-order Transfer von Bytes ("pipe") zwischen Client und Server

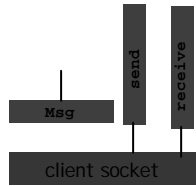
30

## Socket Programmierung mit TCP

Beispiel Client-Server Anwendung:

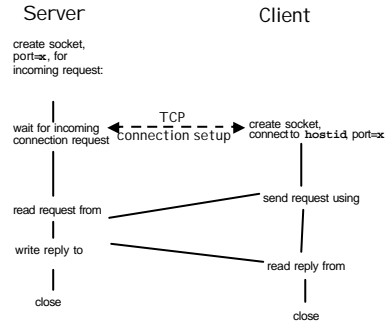
- ❑ Client liest Nachricht von Kommandozeile, sendet an Server über Sockets
- ❑ Server liest Daten vom Socket
- ❑ Server bearbeitet Nachricht und sendet sie zurück an den Client
- ❑ Client liest die modifiziert Nachricht vom Socket und gibt sie aus

Input Stream: Sequenz von Bytes an einen Prozess  
Output Stream: Sequenz von Bytes von einem Prozess



31

## Client/Server Socket Interaktion: TCP



32

## Grundelemente in Perl

- ❑ Packen von Host und Port in C—ähnliche Struktur
 

```
use Socket;
$packed_ip = inet_aton("208.146.240.1");
$socket_name = sockaddr_in($port, $packed_ip);
```
- ❑ Auspacken von Host und Port aus Struktur
 

```
($port, $packed_ip) = sockaddr_in($socket_name);
```
- ❑ Manipulation von IP Adressen
 

```
$ip_address = inet_ntoa($packed_ip);
$packed_ip = inet_aton("204.148.40.9");
$packed_ip = inet_aton("www.oreilly.com");
```

33

## TCP Client (per Hand)

```
use Socket;
# create a socket
socket(TO_SERVER, PF_INET, SOCK_STREAM, getprotobyname('tcp'));
# build the address of the remote machine
$internet_addr = inet_aton($remote_host)
or die "Couldn't convert $remote_host into an Internet address: $!\n";
$padding = sockaddr_in($remote_port, $internet_addr);
# connect
connect(TO_SERVER, $padding)
or die "Couldn't connect to $remote_host:$remote_port: $!\n";
# ... do something with the socket
print TO_SERVER "Why don't you call me anymore?\n";
# and terminate the connection when we're done
close(TO_SERVER);
```

34

## TCP Client (Alternative)

```
use IO::Socket;
$socket = IO::Socket::INET -> new(PeerAddr => $remote_host,
    PeerPort => $remote_port,
    Proto => "tcp",
    Type => SOCK_STREAM)
or die "Couldn't connect to $remote_host:$remote_port : $@\n";
# ... do something with the socket
print $socket "Why don't you call me anymore?\n";
$answer = <$socket>;
# and terminate the connection when we're done
close($socket);
```

35

## TCP Sockets SOCK\_STREAM

```
# Es ist möglich den Port und die Adresse zu kombinieren
$client = IO::Socket::INET -> new("www.yahoo.com:80")
or die $@;
# Aufpassen: Return Werte nach Fehler: undef and $@
$s = IO::Socket::INET -> new(PeerAddr => "Does not Exist",
    Peerport => 80,
    Type => SOCK_STREAM)
or die $@;
# Verringern des TCP_WAIT Timeouts
$s = IO::Socket::INET -> new(PeerAddr => "bad.host.com",
    PeerPort => 80,
    Type => SOCK_STREAM,
    Timeout => 5) or die $@;
```

36

## TCP Server (per Hand)

```
use Socket;
# make the socket
socket(SERVER, PF_INET, SOCK_STREAM, getprotobyname('tcp'));
# so we can restart our server quickly
setsockopt(SERVER, SOL_SOCKET, SO_REUSEADDR, 1);
# build up my socket address
$my_addr = sockaddr_in($server_port, INADDR_ANY);
bind(SERVER, $my_addr) or die "Couldn't bind to port $server_port: $!\n";
# establish a queue for incoming connections
listen(SERVER, SOMAXCONN) or die "Couldn't listen on port $server_port: $!\n";
# accept and process connections
while (accept(CLIENT, SERVER)) {
    # do something with CLIENT
}
close(SERVER);
```

37

## TCP Server (Alternative)

```
use IO::Socket;

$server = IO::Socket::INET->new(LocalPort=> $server_port,
    Type => SOCK_STREAM,
    Reuse => 1,
    Listen => 10 ) # or SOMAXCONN
    or die "Couldn't be a tcp server on port $server_port: $@\n";

while ($client = $server->accept()) {
    # $client is the new connection
}

close($server);
```

38

## Accept

```
# accept nimmt 2 Filehandles als Argumente: Remote Client, Server
# Gibt Port und IP Adresse des Clients zurück
use Socket;
while ($client_address = accept(CLIENT, SERVER)) {
    ($port, $packed_ip) = sockaddr_in($client_address);
    $dotted_quad = inet_ntoa($packed_ip);
    # do as thou wilt
}
# Accept ist eine Methode auf der Server Filehandle
while ($client = $server->accept()) {
    # ...
}
# Listen Umgebung!
while (($client, $client_address) = $server->accept()) {
    # ...
}
```

39

## Nicht blockierende Sockets

```
use Fcntl qw(F_GETFLF_SETFL O_NONBLOCK);

$flags = fcntl($SERVER, F_GETFL, 0)
    or die "Can't get flags for the socket: $!\n";

$flags = fcntl($SERVER, F_SETFL, $flags | O_NONBLOCK)
    or die "Can't set flags for the socket: $!\n";
# Accept wird undef und setzt $! Auf EWOULDBLOCK wenn
# keine Verbindung bereit ist
```

40

## Senden von Daten

```
# Print oder <>
print SERVER "What is your name?\n";
chomp ($response = <SERVER>);
# Benutze send und recv
defined (send(SERVER, $data_to_send, $flags))
    or die "Can't send: $!\n";
recv(SERVER, $data_read, $maxlen, $flags)
    or die "Can't receive: $!\n";
# oder die Methoden der IO::Socket Objekte
use IO::Socket;
$server->send($data_to_send, $flags)
    or die "Can't send: $!\n";
$server->recv($data_read, $flags)
    or die "Can't recv: $!\n";
```

41

## Senden von Daten (2.)

```
# Herausfinden ob Daten gesandt oder geschrieben werden
# können benutze select
use IO::Select;

$select = IO::Select->new();
$select->add(*FROM_SERVER);
$select->add($to_client);

@read_from = $select->can_read($timeout);
foreach $socket (@read_from) {
    # read the pending data from $socket
}
```

42

## Select

```
# Select ermöglicht es zu bestimmen, welche Filehandle ungelesene
# Daten haben, in welche geschrieben werden kann oder
# wo Exceptions aufgetreten sind
# Argumente für select:
#   Bitmask read, Bitmask write, Bitmask exceptions, Timeout

$rin = ""; # initialize bitmask
vec($rin, fileno(SOCKET), 1) = 1; # mark SOCKET in $rin
# repeat calls to vec() for each socket to check

$timeout = 10; # wait ten seconds
$nfound = select($rout = $rin, undef, undef, $timeout);
if (vec($rout, fileno(socket), 1)){
    # data to be read on SOCKET
}
```

43

## Socket Programmierung mit UDP

- UDP: keine "Verbindung" zwischen Client und Server
- Kein Händeschütteln
  - Sender führt explizit die IP Adresse und Portnummer der Destination auf
  - Server muss die IP Adresse, Portnummer des Senders aus dem empfangenen Datengramm extrahieren

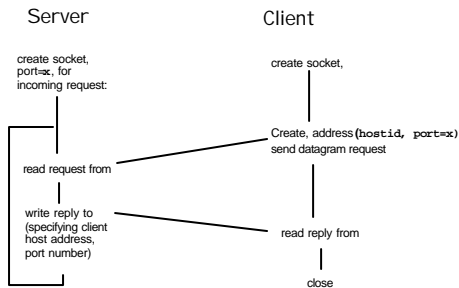
### Anwendungsblickwinkel

UDP stellt unverlässlichen Transfer von Gruppen von Bytes ("Datagramme") zwischen Client und Server

UDP: übertragene Daten können in anderer Reihenfolge oder niemals empfangen werden

44

## Client/server socket interaction: UDP



45

## UDP Client

```
# Öffnen eines UDP Sockets
socket(SockHandle, PF_INET, SOCK_DGRAM, getprotobyname("udp"))
or die "socket $!";

# Senden einer Nachricht zur Maschine $HOSTNAME auf Port $PORTNO
$ipaddr = inet_aton($HOSTNAME);
$portaddr = sockaddr_in($PORTNO, $ipaddr);
send(SockHandle, $MSG, 0, $portaddr) == length($MSG)
or die "cannot send to $HOSTNAME($PORTNO): $!";

# Empfangen einer Nachricht nicht länger als $MAXLEN
$portno = recv(SockHandle, $MSG, $MAXLEN, 0) or die "recv: $!";
($portno, $ipaddr) = sockaddr_in($portno);
$host = gethostbyaddr($ipaddr, AF_INET);
print "$host($portno) said $MSG\n";
```

46

## UDP Server

```
# Öffnen eines UDP Socket (anonym)
use IO::Socket;
$server = IO::Socket::INET->new(LocalPort => $server_port,
    Proto => "udp")
or die "Couldn't be a udp server on port $server_port : $@\n";
# Empfangen von Daten vom Socket
while ($shim = $server->recv($datagram, $MAX_TO_READ, $flags)) {
    # do something
}
```

47

## UDP Server (Beispiel)

```
# Wartet auf Nachrichten und sendet eine Nachricht basierend
# auf der letzten Nachricht
#!/usr/bin/perl -w use strict use IO::Socket;
my($sock, $oldmsg, $newmsg, $hisaddr, $hisport, $MAXLEN, $PORTNO);
$MAXLEN = 1024; $PORTNO = 5151;
$sock = IO::Socket::INET->new(LocalPort => $PORTNO, Proto => 'udp')
or die "socket: $@";
print "Awaiting UDP messages ($PORTNO)\n"; $oldmsg = „Starting message.“;
while ($sock->recv($newmsg, $MAXLEN)) {
    my($sport, $sipaddr) = sockaddr_in($sock->peername);
    $hisport = gethostbyaddr($sipaddr, AF_INET);
    print "Client $hisport said \"$newmsg\"\n";
    $sock->send($oldmsg);
    $oldmsg = "$hisport $newmsg";
}
die "recv: $!";
```

48

## UDP Server (Beispiel)

```
# Wartet auf Nachrichten und sendet eine Nachricht basierend
# auf der letzten Nachricht
#!/usr/bin/perl -w use strict; use IO::Socket;
my($sock, $oldmsg, $newmsg, $hisaddr, $hishost, $MAXLEN, $SPORTNO);
$MAXLEN = 1024; $SPORTNO = 5151;
$sock = IO::Socket::INET->new(LocalPort => $SPORTNO, Proto => 'udp')
    or die "socket: $@";
print "Awaiting UDP messages ($SPORTNO)\n"; $oldmsg = „Starting message.“;
while ($sock->recv($newmsg, $MAXLEN)) {
    my($port, $ipaddr) = sockaddr_in($sock->peername);
    $hishost = gethostbyaddr($ipaddr, AF_INET);
    print "Client $hishost said ``$newmsg``\n";
    $sock->send($oldmsg);
    $oldmsg = "[$hishost] $newmsg";
}
die "recv: $!";
```

49

## Beispiel: Laden einer URL

□ Das richtige Modul macht das Leben einfach!

```
use LWP::Simple;
```

```
$content = get($URL);
```

# Aufpassen mit Fehlermeldungen:

```
# LWP::Simple gibt undef zurück
```

```
use LWP::Simple;
```

```
unless (defined ($content= get $URL)) {
```

```
    die "could not get $URL\n";
```

```
}
```

50

## Beispiel: Laden einer URL (2.)

□ Virtueller Browser

- LWP::UserAgent

□ Generierung eines HTTP Requests

- HTTP::Request

□ Objekte für HTTP Response

- HTTP::Response

□ URL Verarbeitung

- URI::Heuristic  
(Expandiere partielle URLs)

51