Internet-Kommunikation in Python mit Sockets

Dr. Michael Savorić

Hohenstaufen-Gymnasium (HSG)

Kaiserslautern

Version 20140510

Überblick

- Internet-Schichtenmodell
- Grundlagen der Internet-Kommunikation
- Internet-Transportprotokolle: UDP und TCP
- Sockets in Python: socket-Modul
- Kurs-Konventionen
- UDP-Beispiele und Übungen
- TCP-Beispiele und Übungen
- Weitere Übungen
- Spezielle Themen, Ausblick, Anhang
- Literatur

Internet-Schichtenmodell

Grundlagen der Internet-Kommunikation (1/2)

- Jeder Rechner im Internet besitzt mindestens eine IP-Adresse, z.B. 10.221.90.155 (IPv4, 32 Bit)

- Ein Rechner besitzt Portnummern (kurz: Ports) im Bereich 0...65535 (16 Bit)

- Eine Kommunikation zwischen zwei Rechnern A und B (genauer: zwischen Prozessen auf diesen Rechnern) benötigt die IP-Adressen der beteiligten Rechner sowie die Ports:

  A(IP-Adresse, Port) ↔ B(IP-Adresse, Port)

Grundlagen der Internet-Kommunikation (2/2)

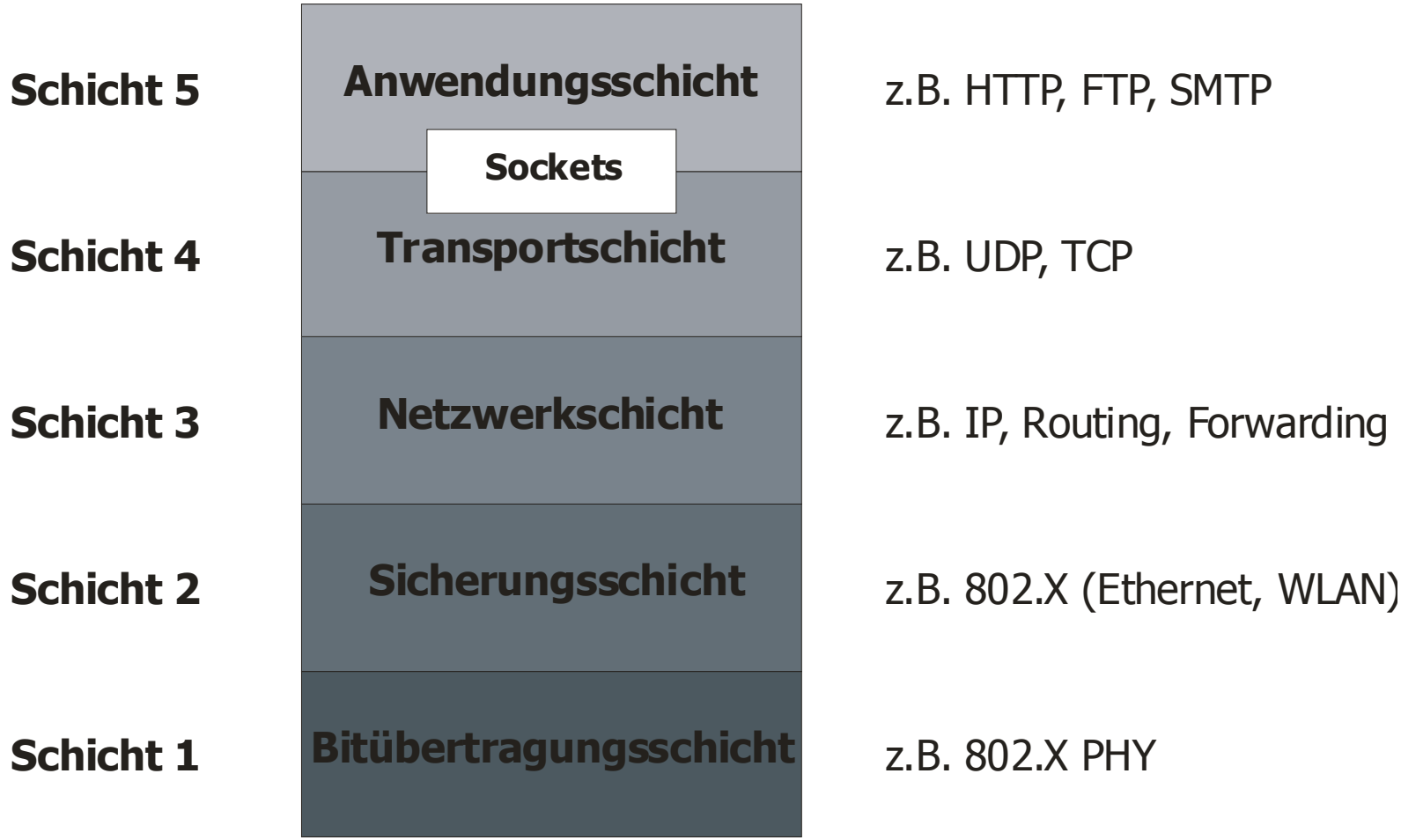
- Bestimmte Ports sind reserviert für besondere standardisierte Dienste, z.B.:

Dienst	Port
Telnet	23
FTP	20, 21
HTTP	80
SMTP	25

Jeder kann eigene Dienste im Netz anbieten, es muss nur der Port für diesen Dienst bekannt sein!

Dienstanbieter (Server) ↔ Dienstbenutzer (Client)

Sockets im Internet-Schichtenmodell



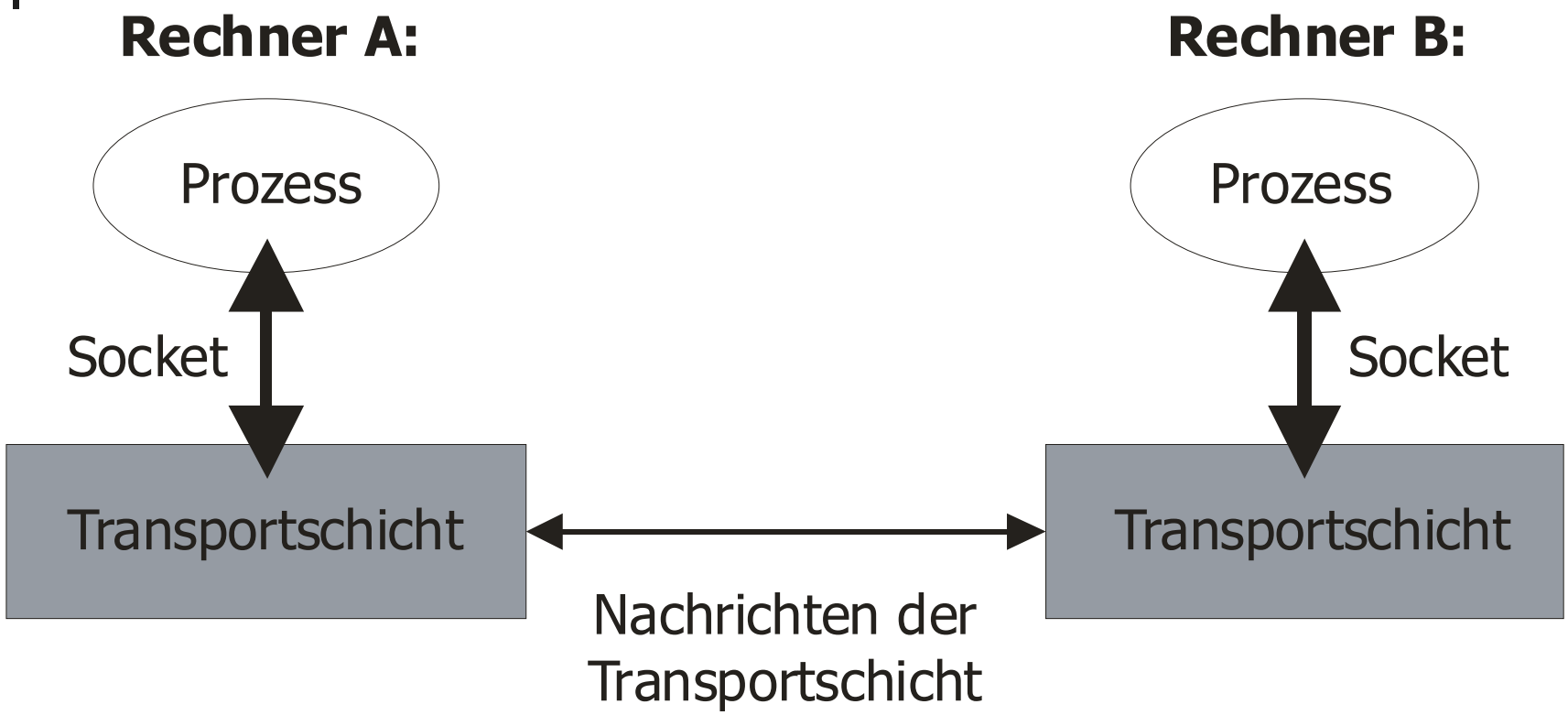
Grundlagen der Internet-Kommunikation mit Sockets (1/3)

- Ein Socket ist eine Schnittstelle zwischen einem Programm bzw. Prozess und den vom Betriebssystem angebotenen Kommunikationsprotokollen der Transportschicht

- Über den Socket läuft die Kommunikation mit dem anderen Rechner (genauer: mit einem Prozess auf dem anderen Rechner)

- Sockets bieten vielfältige Operationen an, verbergen dabei die Komplexität der Kommunikation

Grundlagen der Internet-Kommunikation mit Sockets (2/3)



Nachrichten der Transportschicht sind eingepackt in

Pakete und Rahmen der unterliegenden Schichten

Grundlagen der Internet-Kommunikation mit Sockets (3/3)

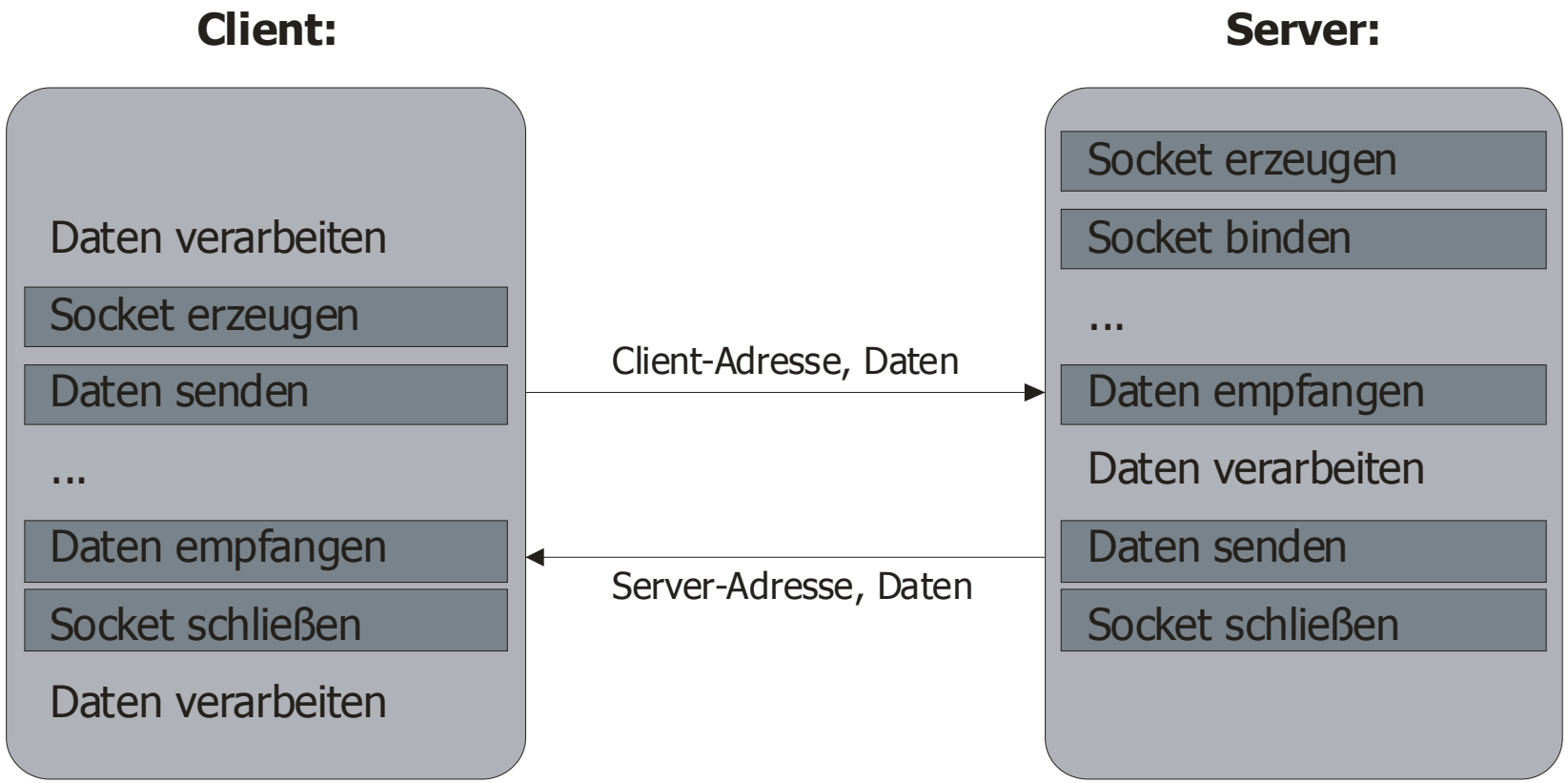
- Für eine Kommunikation über das Internet stehen im wesentlichen zwei Kommunikationsprotokolle der Transportschicht (= Transportprotokolle) zur Verfügung: UDP und TCP

- Abhängig vom gewählten Transportprotokoll UDP bzw. TCP gestaltet sich die Kommunikation mit Sockets unterschiedlich (siehe Beispiele)

UDP (User Datagram Protocol)

- Eigenschaften:
  - verbindungslos
  - keine eingebaute Fehlerkorrektur
- Kommunikationsprinzip:
  - Sofortige Datenübertragung ohne Vor- und Nacharbeiten
- Verwendung:
  - Echtzeit-Video oder Echtzeit-Audio (VoIP)
  - ...

Schematische Kommunikation bei UDP

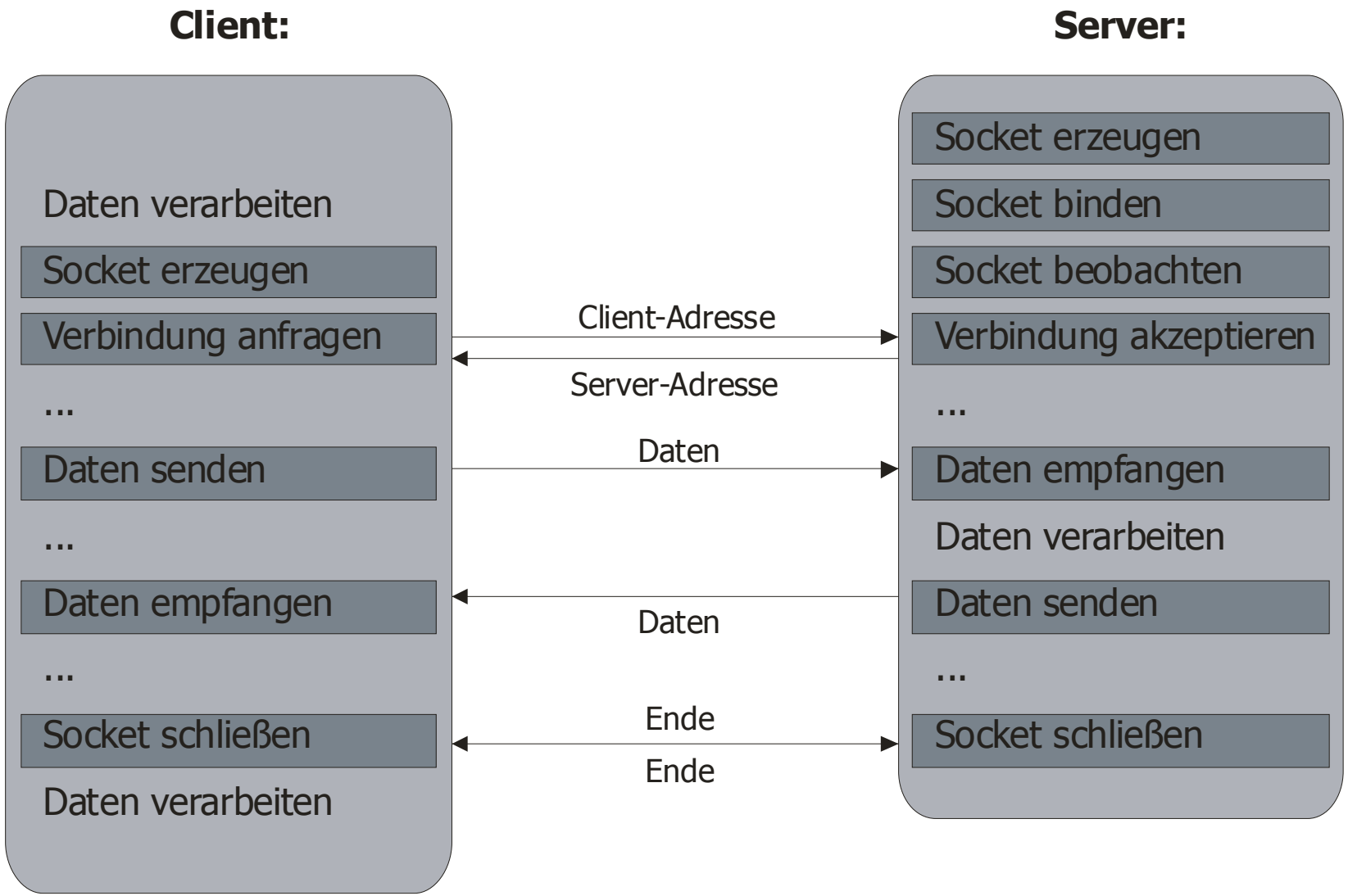


Socket binden = IP-Adresse und Port des Sockets festlegen

TCP (Transmission Control Protocol)

- Eigenschaften:
  - verbindungsorientiert
  - eingebaute Fehlerkorrektur (Quittungsbetrieb)
  - Fluss- und Überlastkontrolle
- Kommunikationsprinzip:
  - Verbindungsaufbau, Datenübertragung, Verbindungsabbau
- Verwendung:
  - HTTP, FTP
  - ...

Schematische Kommunikation bei TCP



Sockets in Python: socket-Modul (1/2)

- Einbinden des socket-Moduls:
  - Variante 1 (am Beispiel eines UDP-Sockets):

```
import socket

client_socket = socket.socket(socket.AF_INET,
                              socket.SOCK_DGRAM)
```

Protokollfamilie

Protokollart

  - Variante 2 (am Beispiel eines UDP-Sockets):

```
from socket import *

client_socket = socket(AF_INET, SOCK_DGRAM)
```

Sockets in Python: socket-Modul (2/2)

- Überblick der im socket-Modul vorhandenen Befehle und Variablen:

```
import socket


dir(socket)
```

- Hilfe zu Befehlen des socket-Moduls, z.B.:

```
import socket


help(socket.socket)
```

Nützliche Funktionen im socket-Modul von Python

- Name des eigenen Rechners:

```
socket.gethostname()
```

- IP-Adresse des eigenen Rechners:

```
socket.gethostbyname(socket.gethostname())
```

- IP-Adresse eines anderen Rechners, z.B.:

```
socket.gethostbyname("www.hsg-kl.de")
```

- Name eines Rechners, z.B.:

```
socket.gethostbyaddr("10.221.90.155")
```

Konventionen für diesen Kurs

- Wir werden Textnachrichten zwischen Prozessen auf den beteiligten Rechnern austauschen
- Über unsere Sockets werden jeweils nur Folgen von Bytes (Byteströme) übertragen
- Wir benötigen daher Funktionen, die aus einem beliebigen Text einen Bytestrom bzw. aus einem Bytestrom einen Text erzeugen

Erklärung für diese Konventionen

- Rechner können für gleiche Datentypen unterschiedliche Speicherformate haben, z.B. das Big-Endian- oder Low-Endian-Format bei Ganzzahl-Datentypen, die länger als ein Byte sind
- Text kann unterschiedlich kodiert sein, z.B. ASCII, CP1252, UTF-8
- Bei der Übertragung von Datentypen- oder Text-Nachrichten zwischen Rechnern müssen diese möglichen Unterschiede immer beachtet werden $\Rightarrow$ erhöhter Programmieraufwand
- Bei der Übertragung von Byteströmen gibt es keine Probleme!

Netzdarstellung und Hostdarstellung von Integer-Zahlen

- Die Netzdarstellung ist Big-Endian, die Hostdarstellung bei z.B. 80x86-PCs Low-Endian, bei vielen Großrechnern Big-Endian

- Hostdarstellung → Netzdarstellung, z.B.:

```
socket.htons(42) → 10752       # zwei Byte
socket.htonl(42) → 704643072   # vier Byte
```

- Netzdarstellung → Hostdarstellung:

```
socket.ntohs(10752) → 42
socket.ntohl(704643072) → 42
```

Konvertierungen String ↔ Bytestrom in Python

- String → Bytestrom:

  Die Funktion

  `string2bytes(text)`

  wandelt einen Text in einen Bytestrom

- Bytestrom → String:

  Die Funktion

  `bytes2string(bytes)`

  wandelt einen Bytestrom in einen Text

- Implementiert im eigenen Modul kodierung (kodierung.py)

kodierung-Modul (Auszug)

```
import sys

def byteorder():
    return sys.byteorder

def standard_encoding():
    return sys.getdefaultencoding()

def standardausgabe_encoding():
    return sys.stdout.encoding

def string2bytes(text):
    return bytes(text, "utf8")

def bytes2string(bytes):
    return str(bytes, "utf8")
```

Hinweise zu den Konvertierungs-Funktionen

- Keine Probleme:
  - Beim Verwenden in eigenen Socket-Anwendungen auf Sender- und Empfängerseite
  - Bei vielen standardisierten Socket-Anwendungen
- Probleme:
  - Bei manchen Socket-Anwendungen, wenn andere Textkodierungen als UTF-8 verlangt bzw. geliefert werden
- Lösung:
  - Im Problemfall entsprechende Textkodierung verwenden

Beispiele: Einbinden der Module socket und kodierung

- socket-Modul:

```
import socket
```

- kodierung-Modul:

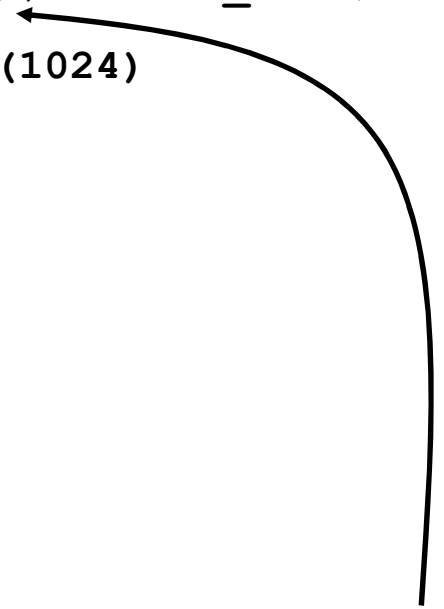
```
from kodierung import string2bytes, bytes2string
```

Gründe:

- Befehle des socket-Moduls im Quelltext kenntlich machen
- Quelltextzeilen nicht zu lang werden lassen

UDP-Beispiel 1: Client für den Daytime-Dienst

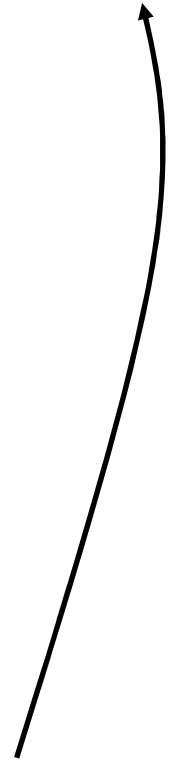
```
server_addr = ("time.fu-berlin.de", 13)


client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client_socket.sendto(string2bytes(""), server_addr)
daten, addr = client_socket.recvfrom(1024)
datenstring = bytes2string(daten)
client_socket.close()
del client_socket


print(datenstring)
```



Daytime-Server benötigen keine Nachricht vom Client

UDP-Beispiel 1: Server für den Daytime-Dienst

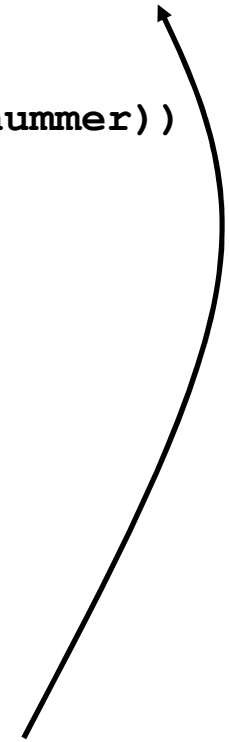
```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind((socket.gethostbyname(socket.gethostname()), 13))
while True:
    daten, addr = server_socket.recvfrom(1024)
    datenstring = ... # String mit der aktuellen Uhrzeit
    server_socket.sendto(string2bytes(datenstring), addr)
server_socket.close()
del server_socket
```



Socket binden an Port 13

UDP-Beispiel 2: Server für einen eigenen Echo-Dienst

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.bind((socket.gethostbyname(socket.gethostname()), 0))
eigene_portnummer = server_socket.getsockname()[1]
print("\nWarte auf Anfragen an Port %5i.\n" % (eigene_portnummer))
while True:
    daten, addr = server_socket.recvfrom(1024)
    datenstring = bytes2string(daten)
    server_socket.sendto(string2bytes(datenstring), addr)
server_socket.close()
del server_socket
```



Socket binden an einen vom System gewählten Port

Übungen mit UDP

- Übung 1:

  Schreiben Sie einen Client für den Echo-Dienst und testen Sie ihn mit dem vorbereiteten Echo-Server.

- Übung 2:

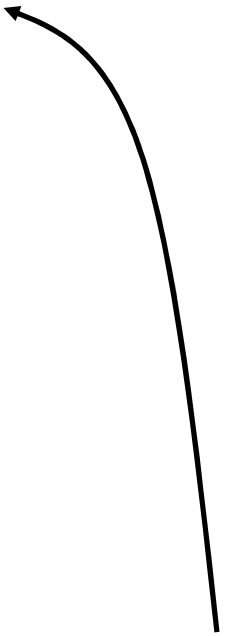
  Schreiben Sie einen Client und einen Server für einen Zeichenzähl-Dienst.

- Übung 3:

  Schreiben Sie einen Client und einen Server für einen Dienst Ihrer Wahl.

TCP-Beispiel 1: Client für den Daytime-Dienst

```
server_addr = ("time.fu-berlin.de", 13)


client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(server_addr)
daten = client_socket.recv(1024)
datenstring = bytes2string(daten)
client_socket.close()
del client_socket


print(datenstring)
```



Daytime-Server benötigen keine Nachricht vom Client

TCP-Beispiel 1: Server für den Daytime-Dienst

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((socket.gethostbyname(socket.gethostname()), 13))
while True:
    server_socket.listen(5)
    client_serving_socket, addr = server_socket.accept()
    datenstring = ... # String mit der aktuellen Uhrzeit
    client_serving_socket.send(string2bytes(datenstring))
    client_serving_socket.close()
    del client_serving_socket
server_socket.close()
del server_socket
```

Socket beobachten

Socket für die Client-Bedienung erzeugen

TCP-Beispiel 2: Server für einen eigenen Echo-Dienst

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((socket.gethostbyname(socket.gethostname()), 0))
eigene_portnummer = server_socket.getsockname()[1]
print("\nWarte auf Anfragen an Port %5i.\n" % (eigene_portnummer))
while True:
    server_socket.listen(5)
    client_serving_socket, addr = server_socket.accept()
    daten = client_serving_socket.recv(1024)
    datenstring = bytes2string(daten)
    client_serving_socket.send(string2bytes(datenstring))
    client_serving_socket.close()
    del client_serving_socket
server_socket.close()
del server_socket
```

Übungen mit TCP

- Übung 1:

  Schreiben Sie einen Client für den Echo-Dienst und testen Sie ihn mit dem vorbereiteten Echo-Server.

- Übung 2:

  Schreiben Sie einen Client und einen Server für einen Quersummen-Dienst.

Weitere Übungen (1/3)

- Übung 1:

  Schreiben Sie einen Client und einen Server für eine Bankanwendung, die für ein Bankkonto Einzahlungen, Auszahlungen (falls möglich!) und Ermittlungen des aktuellen Kontostands erlaubt. Überlegen Sie sich dafür ein geeignetes Protokoll und passende Protokollnachrichten (siehe Anhang) sowie sinnvolle Ein- bzw. Ausgaben beim Client und beim Server.

Weitere Übungen (2/3)

- Übung 2:

  Schreiben Sie einen Client und einen Server für das bekannte Zahlenratespiel, so dass Sie dieses Spiel ab sofort auch über das Internet spielen können. Auch hier müssen Sie sich ein geeignetes Protokoll und passende Protokollnachrichten (siehe Anhang) überlegen.

  Mögliche Erweiterungen:

  - Rateversuche mitzählen und mitteilen
  - Schwierigkeitsgrad vor dem Raten auswählen
  - Spiel abbrechen
  - ...

Weitere Übungen (3/3)

- Übung 3:

  Schreiben Sie einen Client und einen Server für die Übertragung einer (textbasierten, binären oder beliebigen) Datei.

- Übung 4:

  Schreiben Sie einen Client und einen Server für die Übertragung einer Liste oder eines anderen Python-Objekts (Hinweis: das pickle-Modul in Python bietet passende Funktionen an).

Spezielles: Fehlerverarbeitung bei Sockets (1/2)

- Bei der Kommunikation mit Sockets sind viele Fehlerquellen möglich, z.B. ein nicht antwortender Server oder eine nicht vorhandene Verbindung mit dem Internet.

- Verwendung des Try-Except-Mechanismus, z.B.:

```
try:
    client_socket = socket.socket(socket.AF_INET,
                                  socket.SOCK_DGRAM)
    ...
    del client_socket
except socket.error as err:
    print("Fehlernummer: %i, Fehlerbeschreibung: %s"
          % (err.errno, err.strerror))
    sys.exit(1)
```

Spezielles: Fehlerverarbeitung bei Sockets (2/2)

- Der try-except-Mechanismus kann um jeden einzelnen Socket-Befehl verwendet werden, um geeignet auf bestimmte mögliche Fehler bei der Kommunikation reagieren zu können

- Reine Fehlerausgaben reichen in der Praxis nicht aus

Ausblick (1/3)

- Fehler bei Socket-Anwendungen absichtlich herbeiführen und geeignet darauf reagieren

- Clients für standardisierte Dienste schreiben, z.B.:
  - SMTP (Beispiel-Protokollablauf und Client siehe Anhang)
  - HTTP (Beispiel-Protokollablauf und Client siehe Anhang)

  Nachhaltiger Lerneffekt für die ausgewählten Protokolle und die Programmierung von Socket-Anwendungen
  - Hinweis: Python bietet bereits Methoden zur einfachen Verwendung von Standard-Internet-Diensten an, z.B. das Modul urllib für HTTP

Ausblick (2/3)

- Implementierung von Protokollmechanismen anderer Schichten mit UDP, z.B. ein einfacher Quittungsbetrieb aus der Schicht 2 (Alternating-Bit-Protocol):
  - Datenpakete mit alternierender Sequenznummer (0 oder 1)
  - Datenpaket mit aktueller Sequenznummer senden und Timer für den Sender starten
  - Auf Timeout beim Sender reagieren (Wiederholungsübertragung, Anzahl der Wiederholungsübertragungen zählen, eventuell Senden abbrechen, ...)
  - Auf Quittung beim Sender reagieren (Sequenznummer anpassen, neues Datenpaket senden, Senden beenden, ...)

Ausblick (3/3)

- Chatprogramm (LK-Miniprojekt):
  - Protokoll und zusätzliche Dinge (gültige Login-Namen, gültige Nachrichten, ...) für die Kommunikation entwickeln und Alternativen diskutieren (alle gemeinsam an der Tafel, Ergebnis siehe Anhang)
  - Chat-Server schreiben (ein oder zwei Schüler)
  - Chat-Client schreiben (jeder andere Schüler jeweils selbst)
  - Tests

Anhang

Protokollnachrichten des vorbereiteten Bank-Servers

Kommunikation: Client → Server bzw. Client ← Server

- Kontostand ermitteln:

  → "=" ← "OK"+str(kontostand)

- Einzahlung:

  → "+"+str(betrag) ← "OK"

- Auszahlung:

  → "-"+str(betrag) ← "OK" oder "ER"

Protokollnachrichten des vorbereiteten Zahlenratespiel-Servers

Kommunikation: Client $\rightarrow$ Server bzw. Client $\leftarrow$ Server

- Raten:

  $\rightarrow$ str(versuch) $\leftarrow$ "-" oder "+" oder "="

  Bedeutung der Server-Nachrichten:

  "-" : die gesuchte Zahl ist kleiner

  "+" : die gesuchte Zahl ist größer

  "=" : die gesuchte Zahl ist gleich

Einige standardisierte Dienste bzw. Protokolle im Internet

- Simple Mail Transfer Protocol (SMTP):
  - Verschicken von E-Mails
  - TCP an Port 25
- Post Office Protocol (POP):
  - Empfangen und Verwalten von E-Mails
  - TCP an Port 110 bzw. 995 (verschlüsselt)
- Hypertext Transfer Protocol (HTTP):
  - Anfordern von Webseiten
  - TCP an Port 80

SMTP-Beispielkommunikation (aus RFC 5321)

```
S: 220 foo.com Simple Mail Transfer Service Ready
C: EHLO bar.com
S: 250-foo.com greets bar.com
S: 250-8BITMIME
S: 250-SIZE
S: 250-DSN
S: 250 HELP
C: MAIL FROM:<JQP@bar.com>
S: 250 OK
C: RCPT TO:<Jones@XYZ.COM>
S: 250 OK
C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Date: Thu, 21 May 1998 05:33:29 -0700
C: From: John Q. Public <JQP@bar.com>
C: Subject: The Next Meeting of the Board
C: To: Jones@xyz.com
C:
C: Bill:
C: The next meeting of the board of directors will be
C: on Tuesday.
C: John.
C: .
S: 250 OK
C: QUIT
S: 221 foo.com Service closing transmission channel
```

EHLO = Extended Hello
HELO = Hello

S = Server, C = Client

SMTP-Client (1/3)

```
import socket

from kodierung import string2bytes, bytes2string

smtp_client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

smtp_client_socket.bind((socket.gethostbyname(socket.gethostname()), 0))

smtp_server_ip_adresse = socket.gethostbyname("mail.hsg-kl.de")

smtp_server_portnummer = 25

smtp_client_socket.connect((smtp_server_ip_adresse, smtp_server_portnummer))

smtp_client_socket.send(string2bytes("HELO sesamstrasse.de\n"))

data = bytes2string(smtp_client_socket.recv(1024))
```

SMTP-Client (2/3)

```
smtp_client_socket.send(string2bytes("MAIL FROM:<msavoric@t-online.de>\n"))

data += bytes2string(smtp_client_socket.recv(1024))

smtp_client_socket.send(string2bytes("RCPT TO:<msavoric@t-online.de>\n"))

data += bytes2string(smtp_client_socket.recv(1024))

smtp_client_socket.send(string2bytes("DATA\n"))

data += bytes2string(smtp_client_socket.recv(1024))

smtp_client_socket.send(string2bytes("Date: wird nicht verraten\n"))
smtp_client_socket.send(string2bytes("From: <grafzahl@sesamstrasse.de>\n"))
smtp_client_socket.send(string2bytes("Subject: Hallo\n"))
smtp_client_socket.send(string2bytes("To: <msavoric@t-online.de>\n\n"))
```

SMTP-Client (3/3)

```
smtp_client_socket.send(string2bytes("Hallo,\nwie geht es Dir?\n\nGrüße vom
   Grafen\n"))

smtp_client_socket.send(string2bytes("\n.\n"))

data += bytes2string(smtp_client_socket.recv(1024))

smtp_client_socket.send(string2bytes("QUIT\n"))

data += bytes2string(smtp_client_socket.recv(1024))

smtp_client_socket.close()

del smtp_client_socket

print(data)
```

Beispielausgabe des SMTP-Clients

```
220 spielwiese.hsg-kl.de ESMTP
250 spielwiese.hsg-kl.de
250 2.1.0 Ok
250 2.1.5 Ok
354 End data with <CR><LF>.<CR><LF>
250 2.0.0 Ok: queued as 2768312E174
```

HTTP: Anforderung einer Datei (siehe Wikipedia-Artikel)

- Beispiel-HTTP-Server: www.example.net

- Beispiel-Datei: infotext.html

- Anforderung an den HTTP-Server:

```
GET /infotext.html HTTP/1.1
Host: www.example.net
```

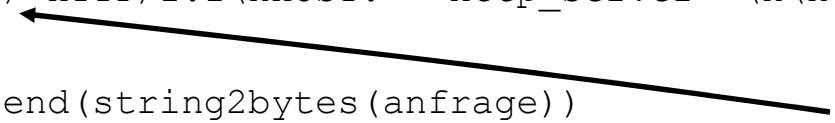
  Wichtig: Leerzeile nach der GET-Anforderung

HTTP-Client (1/3)

```
import socket, string

from kodierung import string2bytes, bytes2string

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

print("\nEigener Rechner: %s (%s)\n"
      % (socket.gethostname(), socket.gethostbyname(socket.gethostname())))

http_server = "www.hsg-kl.de"

http_server_ip_adresse = socket.gethostbyname(http_server)

http_server_portnummer = 80

print("Hole WWW-Startseite von: %s (%s)\n"
      % (http_server, http_server_ip_adresse))

client_socket.connect((http_server_ip_adresse, http_server_portnummer))
```

HTTP-Client (2/3)

```
anfrage = "GET / HTTP/1.1\nHOST: "+http_server+"\n\n"

client_socket.send(string2bytes(anfrage))

daten = ""

while True:
    block = client_socket.recv(1024)
    if not block:
        break
    else:
        daten = daten+str(block, "iso-8859-1") # daten+bytes2string(block)
                                               # scheitert wegen der
                                               # ISO-8859-1-Kodierung
                                               # der Webseite

client_socket.close()

del client_socket
```



Anfordern der WWW-Startseite

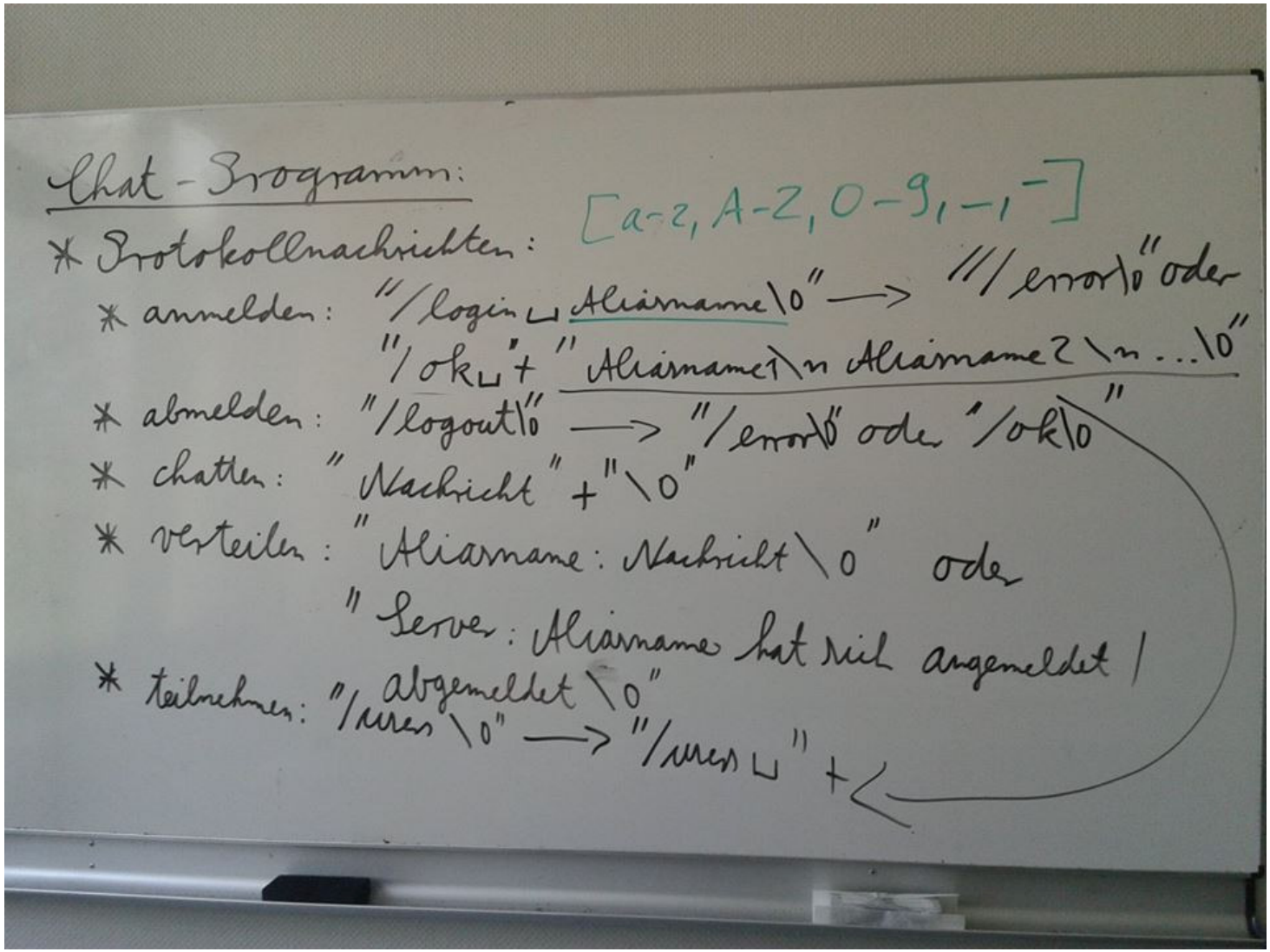
HTTP-Client (3/3)

```
grenze = daten.find("<")

if grenze == -1:
    print("Fehler bei der Übertragung!\n")
else:
    print("HTTP-Overhead:\n"+"--------------\n"+daten[:grenze-1]+"\n")

    print("HTTP-Inhalt:\n"+"------------\n"+daten[grenze:]+"\n")
```

Beispielausgabe des HTTP-Clients

```
Eigener Rechner: Helferlein2 (10.211.112.51)

Hole WWW-Startseite von: www.hsg-kl.de (131.246.120.81)

HTTP-Overhead:
--------------
HTTP/1.1 200 OK
Content-Length: 4525
Date: Fri, 22 Oct 2010 01:48:36 GMT
Server: Apache/2.2.9 (Debian) mod_fastcgi/2.4.6 PHP/5.2.6-1+lenny9 with
   Suhosin-Patch mod_python/3.3.1 Python/2.5.2
X-Powered-By: PHP/5.2.6-1+lenny9
Content-Type: text/html
Connection: keep-alive


HTTP-Inhalt:
------------
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" ...
```

Chatprogramm – Das Protokoll (Tafelbild)



Chatprogramm – Das Protokoll

- Anmeldung (Client und Server):
  `'/login Aliasname\0' → '/error\0' oder`
  `'/ok Aliasname1\nAliasname2\n...\0'`
- Abmeldung (Client und Server):
  `'/logout\0' → '/error\0' oder '/ok\0'`
- Chatten (Client):
  `Nachricht + '\0'`
- Verteilen (Server):
  `'Aliasname:' + Nachricht + '\0'`
- teilnehmende Chatter (Client und Server):
  `'/users\0' →`
  `'/users Aliasname1\nAliasname2\n...\0'`
- erlaubtes Alphabet: `[a-z,A-Z,0-9,_,-]`

Literatur

- Internet-Recherche (Wikipedia, ...)
- RFC 5321, 2008
- Eigene Unterlagen
- M. Summerfield, Programming in Python 3, Addison-Wesley, 2009, ISBN-13: 978-0-13-712929-4, 44.99 $