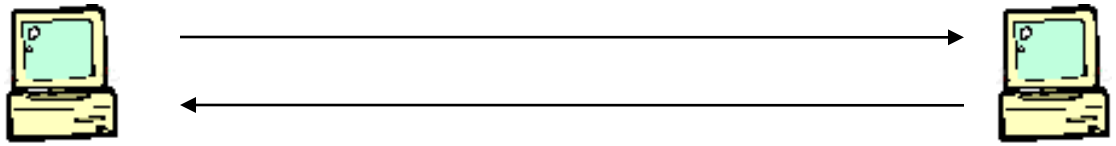
TCP/IP Sockets API

Veerle Ongenae

Computer Chat

- Hoe laten we computers praten?



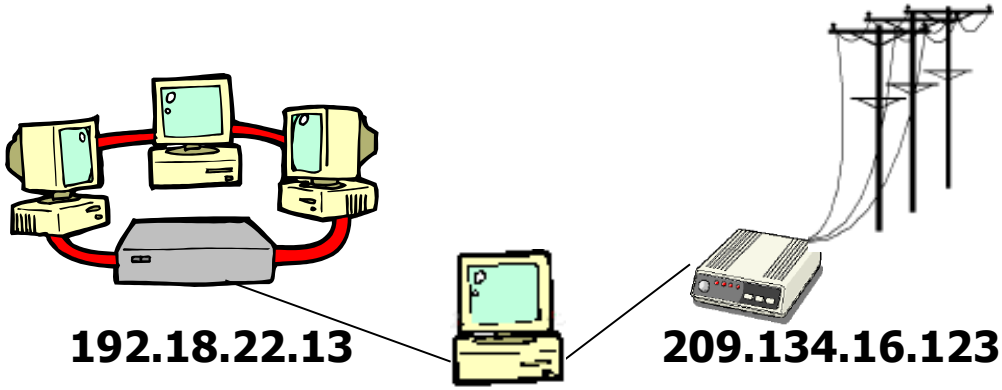
- Hoe zijn ze verbonden?
- Internet Protocol (IP)

Internet Protocol (IP)

- Datagrammen doorsturen via routers
- Best-effort service
  - Verlies
  - Herordering
  - Duplicatie
  - Vertraging
- Host-to-host delivery

IP Adres

- Unieke identificatie
- 32-bit identifier
- dotted-decimal: 192.118.56.25
- www.google.be →167.208.101.28
- Specifieert een host interface (geen host)



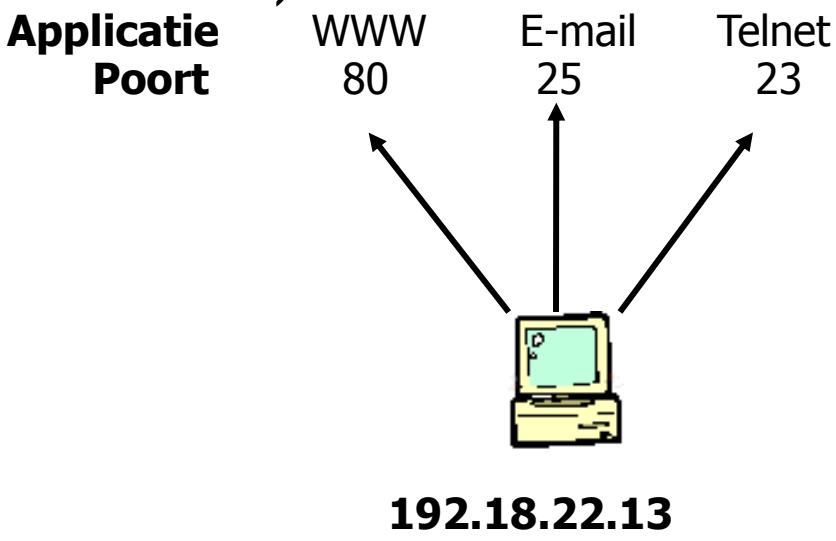
Transport Protocollen

- Best-effort niet voldoende!

- Diensten toevoegen aan IP
- User Datagram Protocol (UDP)
  - Verbindingsloos
  - Verlies, herordening, … mogelijk
- Transmission Control Protocol (TCP)
  - Betrouwbare tweeweg communicatiestromen
  - Sequentieel
  - Foutvrij

Poorten

- De ultieme bestemming
- IP adressen identificeren interface/hosts
- Host heeft vele applicaties
- Poorten (16-bit identifier)



Socket

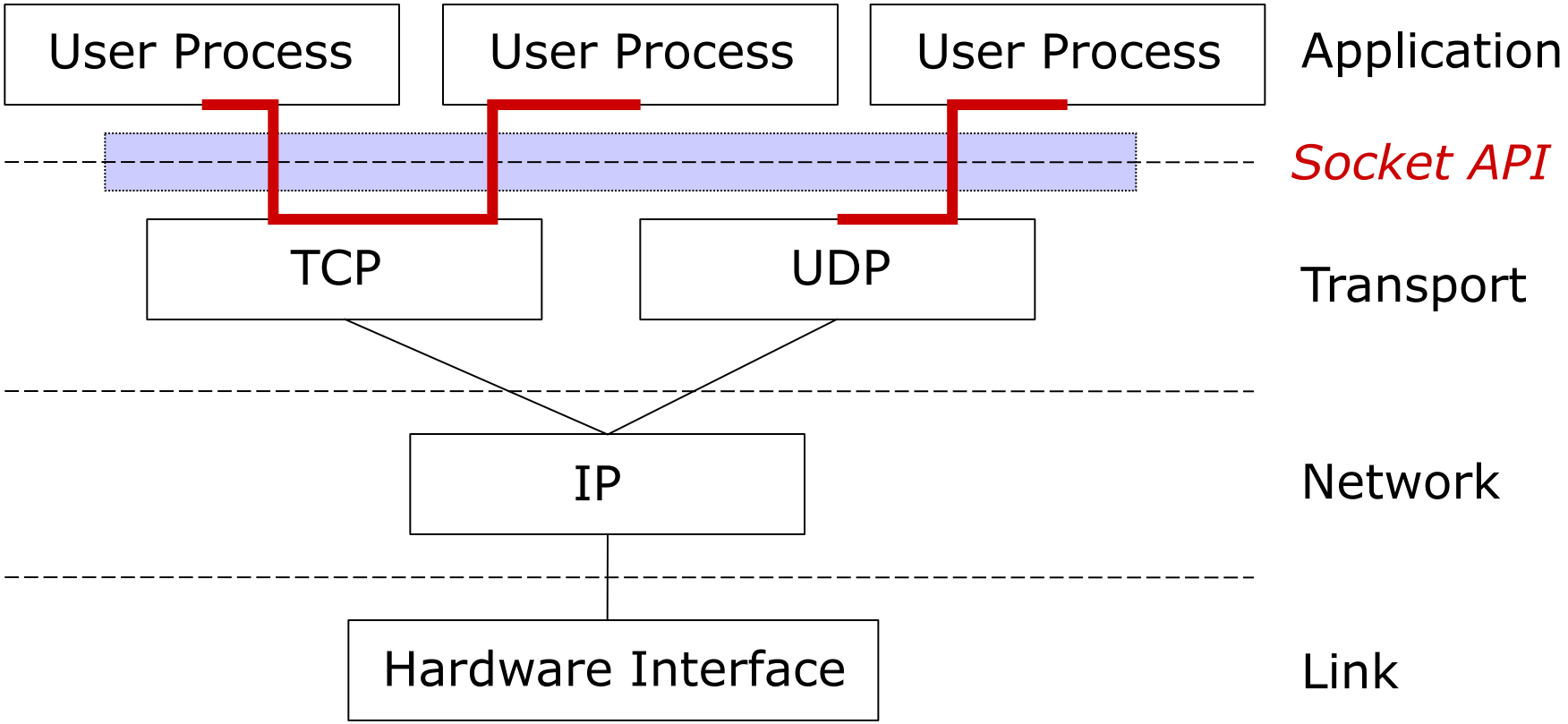
- Hoe praten met TCP/IP?

- Sockets zijn interface voor TCP/IP
- Generieke interface, talloze protocollen

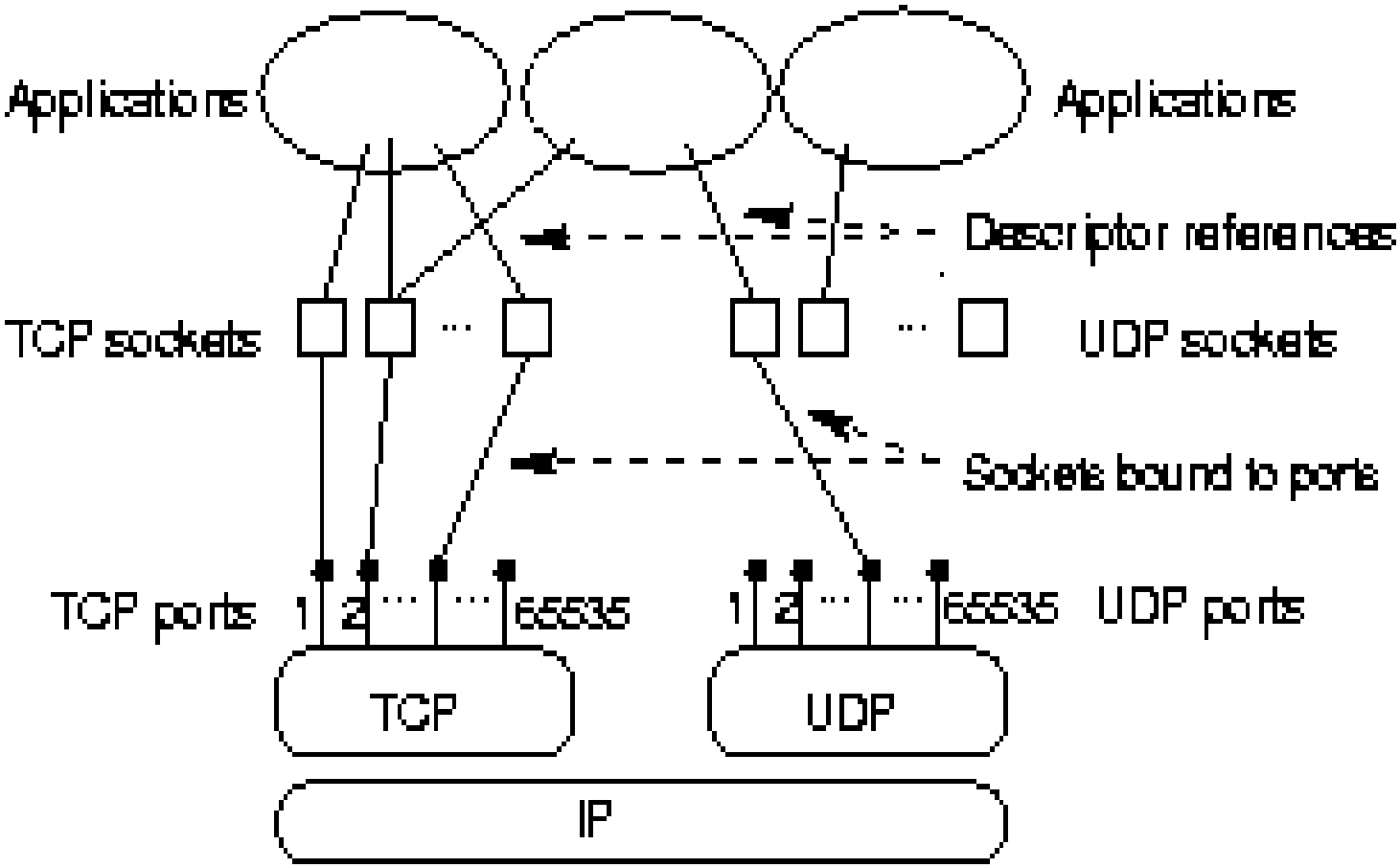
Sockets

- Bepaald door
  - Protocol
  - Local/remote adres
  - Poort
- Eén applicatie kan meerdere sockets aanspreken
- Eén socket kan worden aangesproken door meerdere applicaties

Sockets en TCP/IP



Sockets – TCP–IP

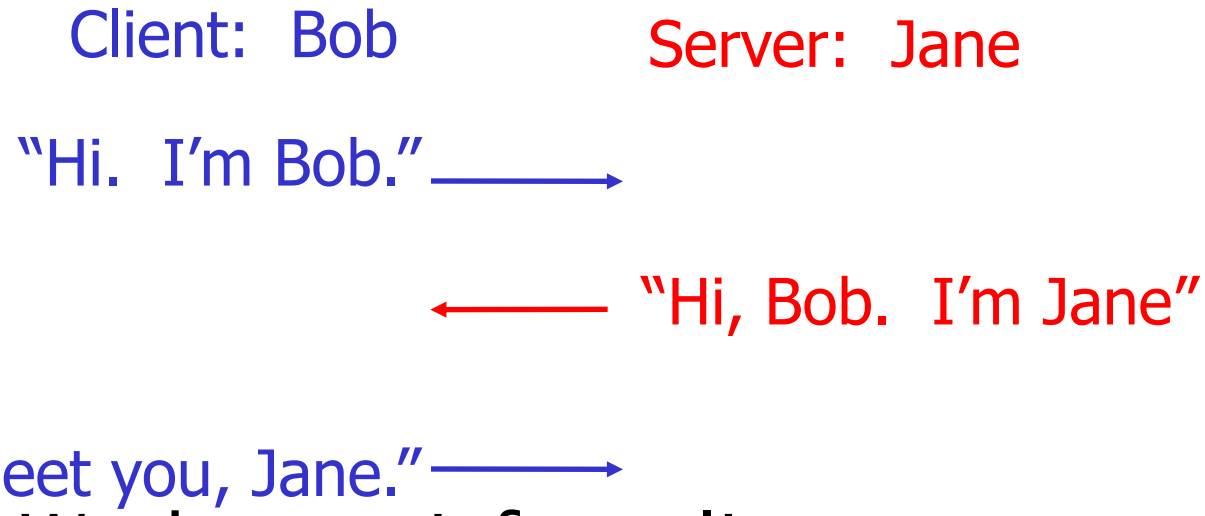


Socket API

- UNIX, Linux platform
- C-bibliotheken
- C++ programma's

Clients en Servers

- Client: Start de connectie/verbinding



- Server: Wacht passief op client
- Vb. printserver, DNS-server, ...

Sockets

- Gebruikt door client en server
- Abstractie
  - Gegevens sturen
  - Gegevens ontvangen
- Analoog open bestand

File Descriptors

- Unix
- Invoer- en uitvoerkanalen
  - Bestanden
  - Sockets
  - Toetsenbord en scherm
  - ...
- Kleine gehele getallen
  - Identificeren kanaal

Standaard File Descriptors

File Descriptor	Kanaal
0	Standaard invoerkanaal
1	Standaard uitvoerkanaal
2	Standaard foutkanaal

Aanmaak TCP/IP Sockets

- Kanaal geassocieerd met file descriptor (=geheel getal)

```
int mySock = socket(familie, type, protocol);
```

- TCP/IP-specifieke sockets

	Familie	Type	Protocol
TCP	PF_INET	SOCK_STREAM	IPPROTO_TCP
UDP		SOCK_DGRAM	IPPROTO_UDP

Aanmaak TCP/IP Sockets

#include <sys/socket.h>
int **socket**(int family, int type, int protocol);
Return :
descriptor (>=0) indien OK,
-1 bij fout

- int mySock = socket(familie, type, protocol);
- Resultaat: socket referentie
  - File (socket) descriptor in UNIX
  - Socket handle in WinSock

Adressen specifiëren

- Socket
  - Protocol
  - Adres + poort
- Adressen
  - Algemeen
    - Alle type sockets (ook niet IP)
    - sockaddr
  - Internetadressen
    - sockaddr_in
    - Casten naar sockaddr

Adressen specifiëren

Generic

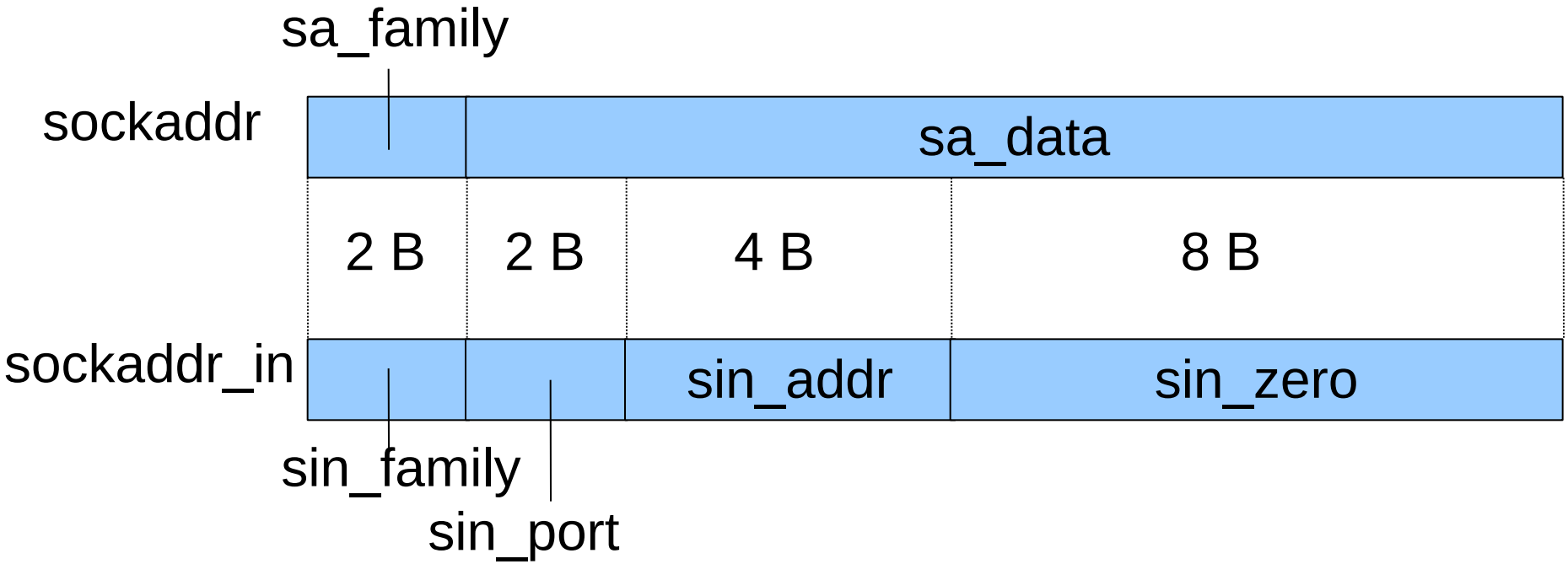
```
struct sockaddr {
    unsigned short sa_family;  /* Address family */
    char sa_data[14];          /* Address information */
};
```

IP Specific

```
struct sockaddr_in {
    unsigned short sin_family; /* Address family (AF_INET) */
    unsigned short sin_port;   /* Port (16-bits) */
    struct in_addr sin_addr;   /* Internet address (32-bits) */
  char sin_zero[8];          /* Not used */
};

struct in_addr {
    unsigned long s_addr;      /* Internet address (32-bits) */
};
```

Adressen


sa_family
sockaddr
sa_data
2 B
2 B
4 B
8 B
sockaddr_in
sin_addr
sin_zero
sin_family
sin_port

Adressen: voorbeeld

```
int poort = 7;
char *IP_adres = “127.0.0.1”;
sockaddr_in servaddr;
memset(&servaddr, 0, sizeof(servaddr)); /* maak struct leeg */
servaddr.sin_family = AF_INET;          /* Internet adres */
inet_aton(IP_adres,&servaddr.sin_addr); /* server adres */
servaddr.sin_port = htons(poort);       /* echo poort */
```

- AF_INET (Address Family InterNet) : const
  - Type adres
- inet_aton (ascii to network)
  - IP-adres (string) --> in_addr
  - Indien fout: resultaat is 0

Adressen: voorbeeld

- Functie inet_addr
  - Oudere versie van inet_aton
  - in_addr_t inet_addr(const char *cp);

servaddr.sin_addr.s_addr = inet_addr(“127.0.0.1”);

  - Foutgevoelig
  - Meerdere platforms

Adressen: voorbeeld

- htons (host to network short)
  - short (byte volgorde host computer) --> short (byte volgorde netwerk)
  - IP-adres en poort worden meegestuurd over het netwerk
- Analoog htonl (host to network long)

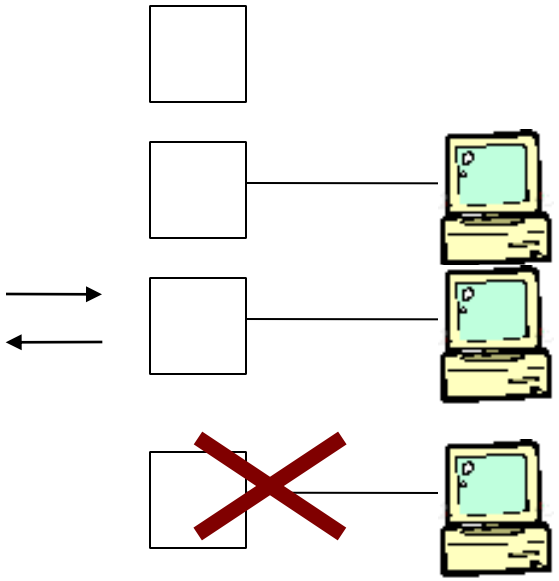
Adressen: voorbeeld

- memset
  - Vul een stuk van het geheugen met een constante
  - Eerste argument: pointer naar startplaats
  - Tweede argument: de constante
  - Derde argument: aantal bytes
- sizeof
  - Bepaalt de lengte van het argument in bytes

TCP Client

Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie



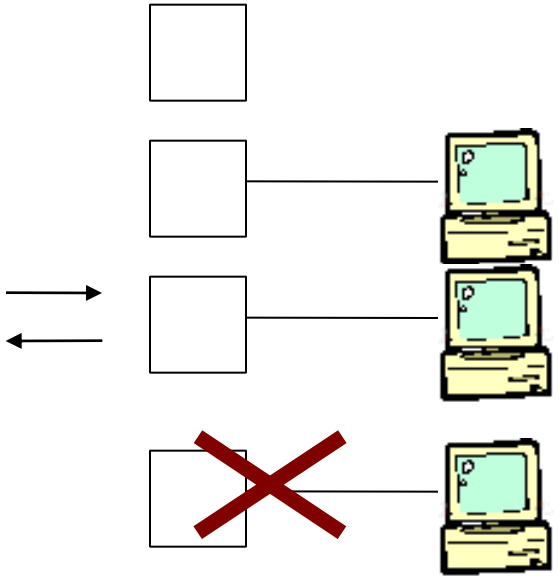
TCP Client

```
/* Create a reliable, stream socket using TCP */

int sockfd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
```

Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie

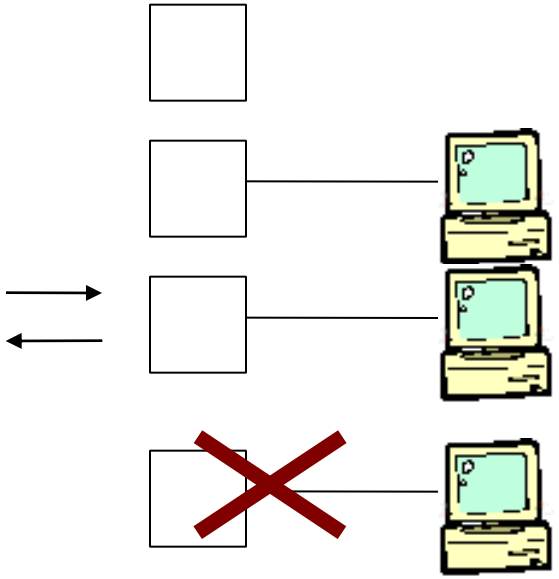


TCP Client

```
int  sockfd ...
sockaddr_in servaddr ...
connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
```

Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie



connect-functie

```
#include <sys/socket.h>
int connect(int sockfd,
        const struct sockaddr *servaddr,
        int addrlen);
```

Return : O als OK, -1 bij fout

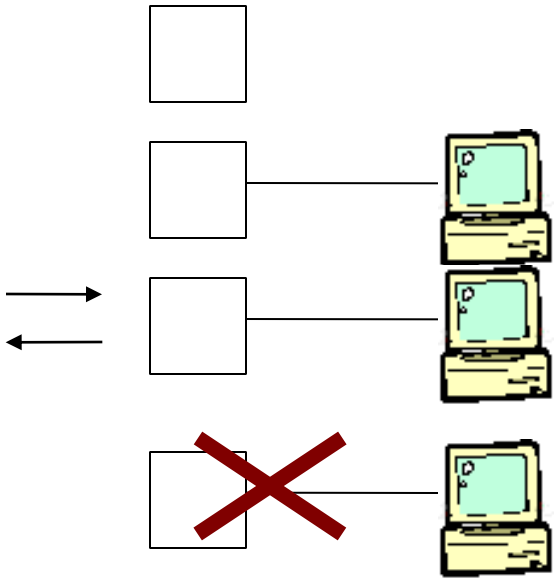
- sockfd : Socket descriptor
- servaddr: IP-adres + poort
- addrlen: Adreslengte
- Initialiseert 3-way-handshake bij TCP

TCP Client

```
int  sockfd ...
char *bericht  ...
// bericht sturen naar server
send(sockfd, bericht, strlen(bericht), 0);
char buffer[RCVBUFSIZE]; // RCVBUFSIZE is constante
// bericht ontvangen van server
int bytesRcvd = recv(sockfd, buffer, RCVBUFSIZE-1, 0);
```

Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie



send- en recv-functie

#include <sys/types.h>
#include <sys/socket.h>
ssize_t **send**(int sockfd, const void *msg, size_t msglength, int flags);
Return: aantal gezonden bytes, -1 bij error

ssize_t **recv**(int sockfd, void *buffer, size_t bufflength, int flags);
Return: aantal ontvangen bytes, 0 bij einde connectie, -1 bij error

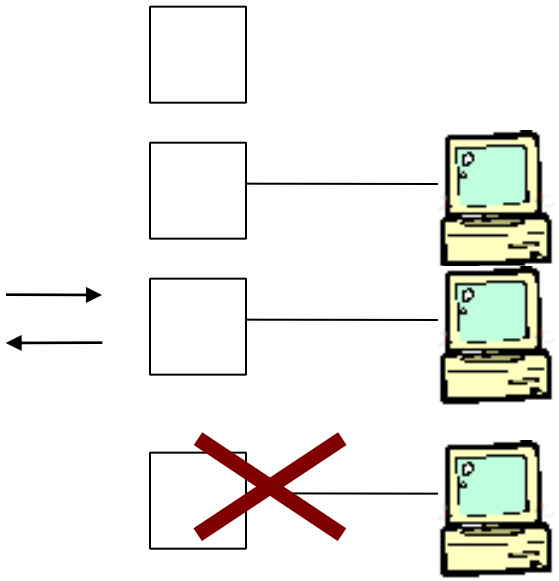
- sockfd : Socket descriptor
- msg, msglength: bericht + lengte (naar server)
- buffer, bufflength: opslaan ontvangen bytes + grootte buffer
- flags: gedrag functie (0)

TCP Client

```
int  sockfd ...
close(sockfd);          /* sluit de verbinding */
```

Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie



close-functie

- int close(int sockfd);
  - sockfd : Socket descriptor
  - Resultaat
    - 0 indien OK
    - -1 indien fout
  - Verbinding wordt gesloten (bij TCP)
    - De andere kant kan niet meer lezen van het kanaal
      - Resultaat recv is 0

Voorbeeld Client

- TCPEchoClient.cpp

Foutafhandeling

- Fout?
  - Resultaat functie -1
  - Foutboodschap in globale gehele variabele errorno
  - perror
    - Drukt foutboodschap (~errorno) af
    - Argument: foutbericht

Conversie

- atoi (ascii to integer)
  - String --> int
- atol (ascii to long)
  - String --> long

TCP Client/Server Interactie

Server begint door zich klaar te maken voor het ontvangen van connectie-aanvragen

Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie

Server

1. Maak TCP socket
2. Ken poort toe aan socket
3. Laat socket luisteren
4. Herhaal:
   a. Aanvaard nieuwe connectie
   b. Communiceer
   c. Sluit de connectie

TCP Client/Server Interactie

/* Maak socket voor binnenkomende connecties */

int listenfd = = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie

Server

1. Maak TCP socket
2. Ken poort toe aan socket
3. Laat socket luisteren
4. Herhaal:
   a. Aanvaard nieuwe connectie
   b. Communiceer
   c. Sluit de connectie

TCP Client/Server Interactie

```
sockaddr_in servaddr; int poort ...
memset(&servaddr, 0, sizeof(servaddr));          /* maak struct leeg */
servaddr.sin_family = AF_INET;                    /* Internet adres */
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);     /* server adres */
servaddr.sin_port = htons(poort);                 /* server poort  */

bind(listenfd, (struct sockaddr *) &servAddr, sizeof(servAddr) );
```

- Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie

- Server

1. Maak TCP socket
2. Ken poort toe aan socket
3. Laat socket luisteren
4. Herhaal:
   a. Aanvaard nieuwe connectie
   b. Communiceer
   c. Sluit de connectie

bind-functie

```
#include <sys/socket.h>
int bind(int sockfd,
        const struct sockaddr *myaddr,
        int addrlen);
Return : 0 bij OK, -1 bij fout
```

- sockfd : Socket descriptor
- servaddr: IP-adres + poort server
- addrlen: Adreslengte
- Poort = 0 --> kernel kiest
- IP-adres = INADDR_ANY --> kernel kiest

TCP Client/Server Interactie

/* laat socket luisteren voor binnenkomende connecties */

listen(listenfd, MAXPENDING) ;

Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie

Server

1. Maak TCP socket
2. Ken poort toe aan socket
3. Laat socket luisteren
4. Herhaal:
   a. Aanvaard nieuwe connectie
   b. Communiceer
   c. Sluit de connectie

listen-functie

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

Return: 0 als OK, -1 bij fout

- sockfd : Socket descriptor
- backlog: maximum aantal aanvragen in wachtrij
- Maakt socket passief
  - Inkomende aanvragen verwerken

TCP Client/Server Interactie

```
sockaddr_in cliaddr;
while(true) {
    unsigned int len = sizeof(cliaddr);
    int connfd = accept(listenfd, (sockaddr *) &cliaddr, &len);
    ...
}
```

- Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie

- Server

1. Maak TCP socket
2. Ken poort toe aan socket
3. Laat socket luisteren
4. Herhaal:
   a. Aanvaard nieuwe connectie
   b. Communiceer
   c. Sluit de connectie

accept-functie

#include <sys/types.h>

#include <sys/socket.h>

int **accept**(int sockfd, struct sockaddr *clientaddr, int *addrlen);

Return: positieve int (descriptor verbonden socket), -1 bij error

- sockfd : Socket descriptor luisterende socket
- clientaddr: adresinformatie client
- addrlen: adreslengte clientadres (pointer!)
- Haalt de eerste van een lijst van aanvragen
  - Maakt een nieuwe socket

TCP Client/Server Interactie

Server is geblokkeerd en wacht op een connectie van een client

Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie

Server

1. Maak TCP socket
2. Ken poort toe aan socket
3. Laat socket luisteren
4. Herhaal:
   a. Aanvaard nieuwe connectie
   b. Communiceer
   c. Sluit de connectie

TCP Client/Server Interactie

een beetje later...

een client wil iets naar de server sturen

- Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie

- Server

1. Maak TCP socket
2. Ken poort toe aan socket
3. Laat socket luisteren
4. Herhaal:
   a. Aanvaard nieuwe connectie
   b. Communiceer
   c. Sluit de connectie

TCP Client/Server Interactie

```
/* ontvang boodschap */
 int connfd;
 char buff[RCVBUFSIZE];
 recvMsgSize = recv(connfd,buff, RCVBUFSIZE-1, 0);
```

- Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie

- Server

1. Maak TCP socket
2. Ken poort toe aan socket
3. Laat socket luisteren
4. Herhaal:
   a. Aanvaard nieuwe connectie
   b. Communiceer
   c. Sluit de connectie

TCP Client/Server Interactie

```
// sluit de connectie met de client
close(connfd);
```

- Client

1. Maak een TCP socket
2. Maak connectie
3. Communiceer
4. Sluit de connectie

- Server

1. Maak TCP socket
2. Ken poort toe aan socket
3. Laat socket luisteren
4. Herhaal:
   a. Aanvaard nieuwe connectie
   b. Communiceer
   c. Sluit de connectie

Voorbeeld TCP Server

- TCPEchoServer.cpp

Voorbeeld TCP/IP Server

- inet_ntoa
  - IP-adres in bytes (netwerkvolgorde) --> IP-adres in dotted-decimal formaat
  - Network to ascii
- Geen exit() bij afhandelen client --> server sluit af!!

TCP varia

- Client kent adres en poort van server
- Geen samenloop `send()` en `recv()`

Client

- send(“Hello Bob”)
- recv() -> “Hi Jane”

Server

- recv() -> “Hello ”
- recv() -> “Bob”
- send(“Hi ”)
- send(“Jane”)

Een connectie sluiten

- close() sluit verbinding
- Analoog aan EOF

- Client
  - send(*string*)
  - while (not received entire string)
  - recv(*buffer*)
  - send(*buffer*)
  - close(*socket*)
- Server
  - recv(*buffer*)
  - while(client has not closed connection)
  - send(*buffer*)
  - recv(*buffer*)
  - close(*client socket*)

Sockets api

- overzicht van de functies
  - socket
  - connect
  - listen
  - bind
  - accept
  - send / recv
- man pages