Communication over the Network

Principles

Services provided by Internet transport protocols

TCP service:

- *Connection-oriented:* setup required between client, server
- *Reliable transport* between sending and receiving process
- *Flow control:* sender won't overwhelm receiver
- *Congestion control:* throttle sender when network overloaded
- *Does not providing:* timing, minimum bandwidth guarantees

UDP service:

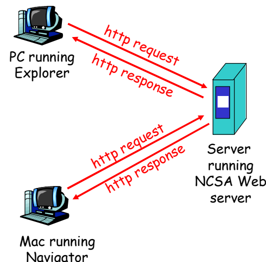
- Unreliable data transfer between sending and receiving process
- Does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee

Q: Why bother? Why is there a UDP?

WWW: the HTTP protocol

HTTP: hypertext transfer protocol

- WWW's application layer protocol
- Client/server model
  - *Client:* browser that requests, receives, "displays" WWW objects
  - *Server:* WWW server sends objects in response to requests



http Beispiel

Annahme der Benutzer gibt folgende URL ein www.someSchool.edu/someDepartment/home.index (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

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

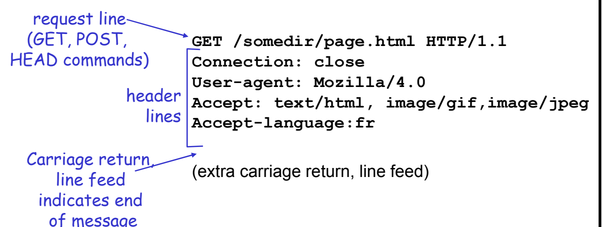
6. Wiederholung von Schritten 1-5 für jedes der 10 jpeg Objekte

Zeit

- Nicht-persistente Verbindungen: ein Objekt in jeder TCP Verbindung
  - Einige Browsers benutzen mehrere TCP Verbindungen *gleichzeitig* – eine pro Objekt
- Persistente Verbindungen: mehrere Objekte können über eine TCP Verbindung transferiert werden

http message format: request

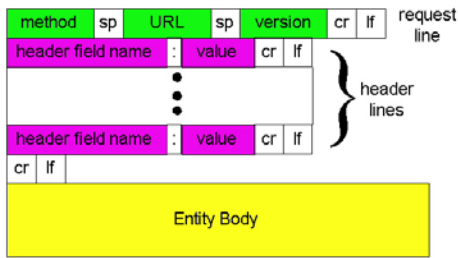
- Two types of http messages: *request, response*
- http request message:
  - ASCII (human-readable format)



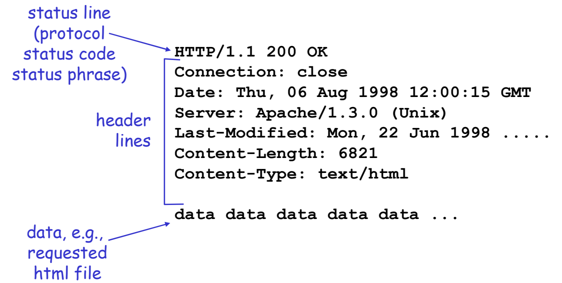

```
GET /somedir/page.html HTTP/1.1
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

(extra carriage return, line feed)

http request Nachricht: generelles Format



http message format: reply



http reply status codes

In first line in server → client response message.

A few sample codes:

200 OK
- request succeeded, requested object later in this message

301 Moved Permanently
- requested object moved, new location specified later in this message (Location:)

400 Bad Request
- request message not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

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 eintippet (drücke enter zweimal), sendet man diesen minimalen (aber kompletten) GET Request zum http Server

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

Socket programming

Goal: learn how to build client/server application that communicate using sockets

Socket API
- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

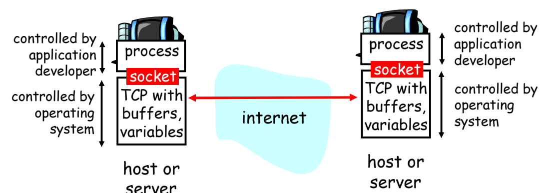
socket

a *host-local*, *application-created/owned*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another (remote or local) application process

Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

TCP service: reliable transfer of bytes from one process to another



Socket programming with TCP

Client must contact server
- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:
- creating client-local TCP socket
- specifying IP address, port number of server process

- When client creates socket: client TCP establishes connection to server TCP
- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients

application viewpoint

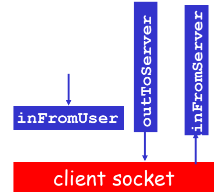
TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Socket programming with TCP

Example client-server app:
- client reads line from standard input (`inFromUser` stream), sends to server via socket (`outToServer` stream)
- server reads line from socket
- server converts line to uppercase, sends back to client
- client reads, prints modified line from socket (`inFromServer` stream)

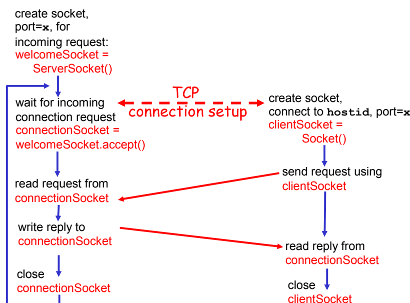
Input stream: sequence of bytes into process

Output stream: sequence of bytes out of process



Client/server socket interaction: TCP

Server (running on `hostid`) Client



Socket Programming

In Perl

Basic Elements

- Packing of host and port into C—like structure

```
use Socket;
$packed_ip   = inet_aton("208.146.240.1");
$socket_name = sockaddr_in($port, $packed_ip);
```

- Extraction of host and port out of structure

```
($port, $packed_ip) = sockaddr_in($socket_name);
```

- Manipulation of IP addresses

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


TCP Client (low-level)

(Socket) Handle — Constants

```
use Socket;
# create a socket
socket(TO_SERVER, PF_INET, SOCK_STREAM, getprotobyname('tcp'));
# build the address of the remote machine
$remote_addr = inet_aton($remote_host)
    or die "Couldn't convert $remote_host into an Internet address: $!\n";
$paddr = sockaddr_in($remote_port, $remote_addr);
# connect
connect(TO_SERVER, $paddr)
    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);
```

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);
```


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 $@;
```


TCP Server (low-level)

```
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);
```


Accept

```
# accept() takes 2 filehandles as argument: remote client and remote server
# Returns port and IP address of the client

use Socket;
socket(SERVER, ...);
bind(...);
listen(...);

while ($client_address = accept(CLIENT, SERVER)) {
   ($port, $packed_ip) = sockaddr_in($client_address);
   $client_ip = inet_ntoa($packed_ip);
   # do as thou wilt
}
```


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);
```


Sending/Receiving of Data

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

Socket Programming with UDP

UDP: no "connection" between client and server

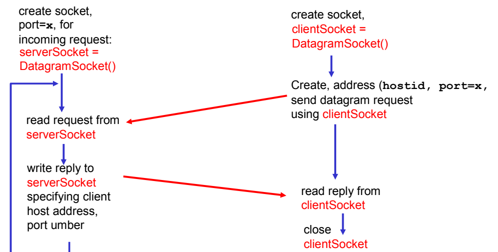
- No Handshake
- Sender explicitly specifies destination IP address and port number
- Receiver has to extract IP address and port number of the sender out of the received datagram

UDP: transmitted data can arrive out of order or be lost entirely

Application perspective

UDP provides unreliable transfer of groups of bytes ("datagrams") between sender and receiver

Client/server socket interaction: UDP



UDP Client (low-level)

```
# Creation of UDP sockets
socket(SOCKET, PF_INET, SOCK_DGRAM, getprotobyname("udp"))
   or die "socket: $!";
# Sending a message to host $HOSTNAME and portort $PORTNO
$packed_ipaddr   = inet_aton($HOSTNAME);
$destaddr = sockaddr_in($PORTNO, $packed_ipaddr);
send(SOCKET, $MSG, 0, $destaddr) == length($MSG)
      or die "cannot send to $HOSTNAME($PORTNO): $!";
# Receiving data of no more than $MAXLEN bytes
$packed_addr = recv(SOCKET, $MSG, $MAXLEN, 0) or die "recv: $!";
($portno, $ipaddr) = sockaddr_in($packed_addr);
$hostname = gethostbyaddr($ipaddr, AF_INET);
print "$hostname : $ipaddr:$portno said '$MSG'\n";
```


UDP Server (alternativ)

```
# Creation of a UDP Socket (anonymous)
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 : $@\n";
# Receiving data from socket
while ($him = $server->recv($datagram, $MAX_TO_READ,
    $flags)) {
   # do something
}
```


UDP Server (Example)

Wait for messages and sends another message based on the last received one

```
#!/usr/bin/perl –w
use strict;  use IO::Socket;
my($sock, $oldmsg, $newmsg, $hisaddr, $hishost, $MAXLEN, $PORTNO);
$MAXLEN = 1024;  $PORTNO = 5151;
$sock = IO::Socket::INET->new(LocalPort => $PORTNO, Proto => 'udp')
                  or die "socket: $@";
print "Awaiting UDP messages ($PORTNO)\n"; $oldmsg = „Starting message.";
while (defined($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, 0, sockaddr_in($port, $ipaddr));
   $oldmsg = "[$hishost] $newmsg";
}
```


Socket Programming using Select

Example server application:

- Server wants to handle multiple TCP connections simultanously
- But server blocks with every recv() call for one connection while there might be data ready on another one

- Solution: select()

Select

```
# select() determines from which file handles one can read (write) without
# blocking or where exception are pending
# Arguments for select():
#    Bitmask readable, bitmask writeable, 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
$nfound = select($rin, undef, undef, $timeout);
if (vec($rin, fileno(socket),1)){
   # data to be read on SOCKET
}
```


Select (alternative)

```
use IO::Select;

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

@read_from = $select->can_read($timeout);
foreach $handle (@read_from) {
   if($handle == \*STDIN) {
      # read from keyboard
   elsif($handle == \*FROM_SERVER) {
      # read the pending data from $socket
   }
}
```


Where to find documentation

man perlipc

perldoc -f {socket, bind, listen, send, recv}

perldoc Socket

perldoc IO::Socket

perldoc IO::Select