Communication Networks 2
Exercise 5 - HTTP

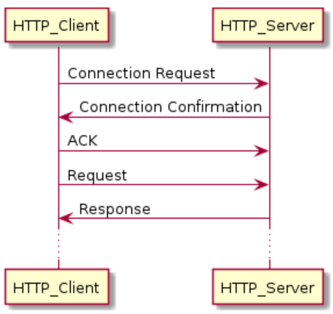
TECHNISCHE UNIVERSITÄT DARMSTADT

Multimedia Communications Lab
TU Darmstadt

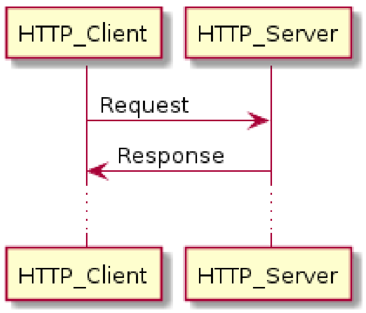
Problem 1 Messages in a HTTP session

Which Application Layer Messages are exchanged during a HTTP Session? Draw a Sequence Diagramm.

Solution:



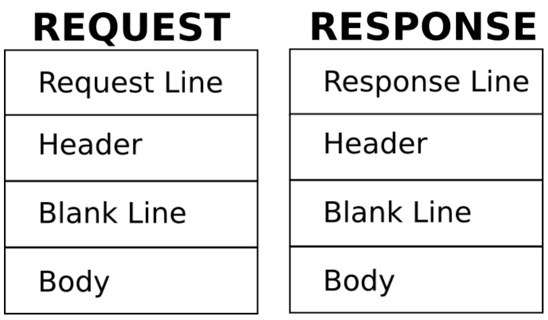
Is this Application Layer only?



Problem 2 HTTP Request and Response

How does a HTTP Request look like? And what is the structure of a HTTP Response?

Solution:



Problem 3 Transferduration of HTTP 1.0

Consider downloading a web page consisting of 100000 bytes of text and 10 jpeg images with a size of 500000 bytes each. You have a downlink speed of 2 Mbit/s and an uplink speed of 1Mbit/s. The roundtrip time between your host and the server is 100 milliseconds.

How long does it take to load the page including the images using HTTP Version 1.0 in the best(ten connections in parallel) and in the worst case(one connection)? Just consider the information given above and neglect all headers. Assume that the TCP connection setup takes one roundtrip.

Solution:

$+$ 100 ms for the TCP handshake

$+$ 100 ms for the request

$+$ 400 ms for the page download

Best case(10 connections in parallel):

$+$ 100 ms for the TCP handshake

$+$ 100 ms for the request

$+$ 2000 ms $\cdot$ 10 for the image download

$=$ 20.8 s

Worst case(1 connection):

$+$ 100 ms $\cdot$ 10 for the TCP handshake

$+$ 100 ms $\cdot$ 10 for the request

$+$ 2000 ms $\cdot$ 10 for the image download

$=$ 22.6 s

Problem 4 Transferduration of HTTP 1.1

How long does it take to load the page including the images using HTTP Version 1.1 and a keep-alive timeout of 10 seconds? How many tcp connections are opened to the server? The number of parallel connections per host is two. Just consider the information given above and neglect all headers.

Solution:
The Client opens two persistent connections to the server. It takes:

$+$ 100 ms for the TCP handshake

$+$ 100 ms for the page request

+ 400 ms for the page download

+ 100 ms for the request

+ 2000 ms · 10 for the image download

= 20.7s to fetch the page and all images.

Problem 5 Practical experiences with HTTP

The next tasks will be more practical to give you a better understanding about the http protocol. You will develop a Mensa-Meal server that serves the Mensa Meal as a digital calendar and also as a JSON string for further external projects.
First of all you need to write a crawler that fetches the Mensa program and transform it into a usable format. Before you start with the crawler you need to get a better understanding of the messages which will be exchanged.
You will do this by replaying messages (if you are familiar with wireshark, this could be done a lot easier). First start a HTTP Client and an HTTP Server in two different terminals using netcat with :

```
$nc www.studentenwerkdarmstadt.de 80
$nc -l 8000
```

open a browser and go to: `http://127.0.0.1:8000/index.php/de/essen-und-trinken/speisekarten/stadtmitte?ansicht=week`. Now take a look into the server console and enter the received code into the client console (remember to change the Host: line). If you like it, you can now copy the response to the server window and so on to observe the complete network traffic. Please take a look now at the HTTP request and the HTTP response headers you've seen and answer the following questions:

a) What is the observed Useragent?

b) What is the answer code of the server?

c) Does the Accepted content match the delivered content?

d) Which type of HTTP is used?

e) Does the page provides a cookie?

f) What does "Transfer-Encoding: chunked" mean?

Solution:
The request that is send to the server is:

```
GET /index.php/de/essen-und-trinken/speisekarten/stadtmitte?ansicht=week HTTP/1.1
Host: www.studentenwerkdarmstadt.de
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:32.0) Gecko/20100101 Firefox
    /32.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
```

```
Cookie: e0e1c37c0d906f9e370073a3657d04ea=de-DE; a9fc66977b9d5857d1177ecf6f356597=
    b1bvcgepb5crilkg3nbo7t60s1
X-Forwarded-For: 12.13.14.15
Connection: keep-alive
```

who replies with:

```
HTTP/1.1 200 OK
Date: Fri, 26 Sep 2014 13:53:01 GMT
Server: Apache/2.2.27 (Unix)
X-Powered-By: PHP/5.3.29
Set-Cookie: e0e1c37c0d906f9e370073a3657d04ea=de-DE; expires=Sat, 26-Sep-2015
    13:53:01 GMT; path=/
P3P: CP="NOI ADM DEV PSAi COM NAV OUR OTRo STP IND DEM"
Cache-Control: no-cache
Pragma: no-cache
Keep-Alive: timeout=3, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8

2000
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
    TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de-de" lang="de-de" >
<head>
...
```

Which unfolds (in our case) the following answers to the questions:

a) What is the observed Useragent?
   - Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:32.0) Gecko/20100101 Firefox/32.0

b) What is the answer code of the server?
   - 200 OK

c) Does the Accepted content match the delivered content?
   - Yes, text/html is accepted and delivered.

d) Which type of HTTP is used?
   - HTTP 1.1 with a Keep-Alive connection.

e) Does the page provides a cookie?
   - Yes, see the set-cookie line.

f) What does "Transfer-Encoding: chunked" mean?
   - It declares that the content is sended chunked, this makes the content length header obsolete as every chunk has an own length identifier.

Problem 6 Creating response codes

Now as you know how it should work, try to create error Messages as you see what can happen. What are the status reply codes that you have learned in the lecture? Do you have an idea how to create them? Try to netcat again onto the server and find out what you have to send to produce them. Hint: For some of the codes you need to use a different server.

Solution:
The status codes from the lecture and the messages you need to get them are:

- 200 OK
  Request succeeded, requested object contained in message
  ```
  echo "GET / HTTP/1.1\nHost : www.studentenwerkdarmstadt.de\n \n " | nc www.studentenwerkdarmstadt
  .de 80
  ```
- 301 Moved Permanently
  Requested object moved, new location specified in message
  ```
  echo "GET / HTTP/1.1\nHost : www.elitefachschaft.de\n \n " | nc www.elitefachschaft.de 80
  ```
- 400 Bad Request
  Request message not understood by server
  ```
  echo "GET / HTTP/1.1\n \n " | nc www.studentenwerkdarmstadt.de 80
  ```
- 404 Not Found
  Requested document not found on this server
  ```
  echo "GET /unavailable.html HTTP/1.1\nHost : www.studentenwerkdarmstadt.de\n \n " | nc www.studentenwerk
  .de 80
  ```
- 505 HTTP Version Not Supported

Problem 7 Automated client

As you have learned before how you can speak the HTTP protocol by hand, it's now time to do it completely automated. Your task is to write an http client (we recommend python) that connects to the Mensa page, fetches the meal of the week and prints it to stdout.
You can do this with a raw socket which works like the manual way in exercise Problem 5 or with an http library. As it is not our goal to test your html parsing skills we created a http_exercise_utils.py library that you are able to find in moodle and that contains a function which transforms the html code into pretty printable text.

Solution:
We provide here both solutions, first with a framework:

```python
#!/usr/bin/python
import urllib2
from http_exercise_utils import *
reload(sys)
sys.setdefaultencoding("utf-8")
response = urllib2.urlopen('http://www.studentenwerkdarmstadt.de/index.php/de/
    essen-und-trinken/speisekarten/stadtmitte?ansicht=week')
html = response.read()
print htmlToText(html)
```

code/http_framework.py

And then the socket version:

```
#!/usr/bin/python
import socket
from http_exercise_utils import *
import re
reload(sys)
sys.setdefaultencoding("utf-8")
#create an INET, STREAMing socket
s = socket.socket(
    socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
#now connect to the web server on port 80
# - the normal http port
s.connect(("www.studentenwerkdarmstadt.de", 80))
#send the request (HTTP 1.0 as it does not support chunked transfer encoding)
request = "GET /index.php/de/essen-und-trinken/speisekarten/stadtmitte?ansicht=
   week HTTP/1.0\nHost: www.studentenwerkdarmstadt.de\n\n"
s.send(request)
#receive the data
data = ""
while True:
    buf = s.recv(1024)
    if not len(buf):
        break
    data += buf
print(htmlToText(data))
#afterwards close the socket
s.shutdown(1)
s.close()
```

code/http_socket.py

Problem 8 Automated server

You have learned how the http protocoll works and how a http-client could be written. Now it is time to write an http server. Python has a build in http server which can be launched with:

```
python -m SimpleHTTPServer
```

But this is not our goal. What we only want to do is serving the Mensa page in 3 types:

- As html which is just relaying the queried material,
- as ical so that you can import it into your calendar and
- as json so that other applications are able to use it.

So your Server should react on a GET command to the files: mensa.html, mensa.ics and mensa.json which are then generated dynamically in python (the http_exercise_utils.py comes with functions that transform the html code into the needed format). Please send a 404 if any other page is requested and, think about the headers, are there any fields that indicate which data format the user requests, are there any fields that tell the users which type of data he gets (this is important so that other programs like calendars understand that which type of data they receive)?

Solution:

```python
#!/usr/bin/python
import socket
import re
import urllib2
from http_exercise_utils import *
reload(sys)
sys.setdefaultencoding("utf-8")
# define the error result
error = "HTTP/1.1 404 Not Found\n\n"
# open a socket
s =  socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 8080))
while True:
  try:
    s.listen(1)
    conn, addr = s.accept()
    # get some data
    data = conn.recv(1024)
    string = bytes.decode(data)
    # check for correct get command
    correct_get = re.compile(r'GET /mensa\.((ics)|(html)|(json)) HTTP/1\.1')
    reply = ""
    if correct_get.search(string) is not None:
      # deliver the correct content
      header = "HTTP/1.1 200 OK"
      html = urllib2.urlopen('http://www.studentenwerkdarmstadt.de/index.php/de/
   essen-und-trinken/speisekarten/stadtmitte?ansicht=week').read()
      if "/mensa.ics" in string:
        header += "Content-Type: text/calendar; charset=utf-8\n\n"
        reply = header+htmlToIcs(html)
      elif "/mensa.json" in string:
        header += "Content-Type: application/json; charset=utf-8\n\n"
        reply = header+htmlToJSON(html)
      elif "/mensa.html" in string:
        header += "Content-Type: text/html; charset=utf-8\n\n"
        reply = header+html
      else:
        reply = error #should not be possible
    else:
      reply = error
    conn.send(reply)
    conn.close()
  # on strg+c close the socket
  except KeyboardInterrupt:
    s.shutdown(1)
    s.close()
    sys.exit()
```

code/http_server.py