Kommunikation in Netzwerken

Einstieg: Aufbau des Internets

Aufgabe Webnetsim:

Du erhältst eine Rolle. Geh auf https://webnetsim-bayern.de/ und folge den Anweisungen auf dem Blatt für deine Rolle.

Netzwerk

Ein ***Netzwerk*** ist ein räumlich verteiltes System von Rechnern, die miteinander verbunden sind. Dadurch wird ein Austausch von Daten (Informationen, Dokumenten, E-Mails, …) möglich.

Visualisierung des Internets

https://www.youtube.com/watch?v=tiGMgU6_1x4

https://www.youtube.com/watch?v=-L1Zs_1VPXA

https://en.wikipedia.org/wiki/Internet_backbone

Client-Server-Prinzip

In der heutigen Zeit kommunizieren Rechner nicht einfach irgendwie (unkontrolliert) miteinander. Kommunikation findet nach dem ***Client-Server-Prinzip*** statt. Das bedeutet, dass ein ***Rechner (Server) im Netzwerk einen Dienst zur Verfügung stellt und die anderen Rechner (Client) greifen darauf zu***. Beispiele:

- Ein ***Webserver*** stellt Webseiten zur Verfügung. Andere Rechner sehen sich die Seiten mit ***Webbrowser*** (Client-Programm) an.
- Ein ***E-Mail-Server*** stellt die Möglichkeit des Versendens von Nachrichten bzw. das Abrufen von empfangenen Nachrichten zur Verfügung. Die anderen Rechner verwenden ***Mail-Client***-Programme, um diesen Dienst zu nutzen.
- Ein ***Dateiserver*** stellt Speicherplatz zur Verfügung. Andere Rechner können mit ihrem ***Datei-Client*** Dateien darauf speichern oder von dort herunter laden (wie z. B. bei einer Cloud-Anwendung).
- Ein ***DNS-Server*** (Domain Name System) übersetzt Domain-Namen in IP-Adressen (z.B. www.joachimhofmann.org in 85.214.106.113) und umgekehrt. Andere ***„Client-“Rechner*** nutzen diesen Dienst, wenn man im Internet surft.

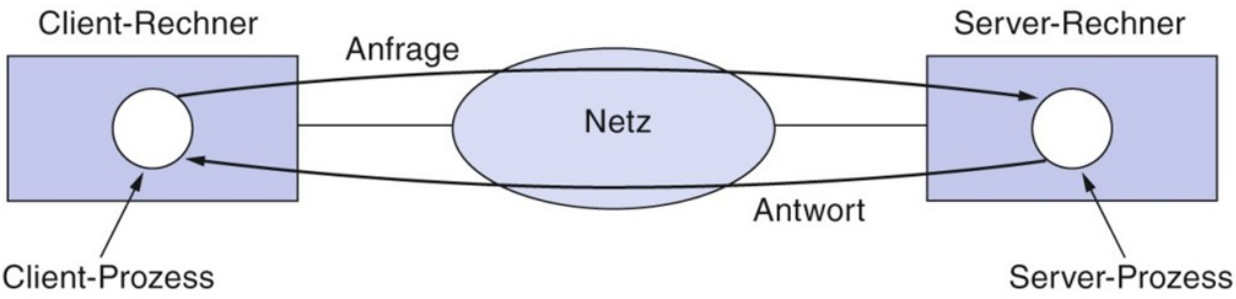


Abbildung 1.2: Das Client-Server-Modell enthält Anforderungen und Antworten.

Aufbau des Schichtenmodells TCP/IP

Beim TCP/IP-Modell gibt es vier Schichten, deren Sinn und Funktionalität wir Schicht für Schicht von unten nach oben kennenlernen werden.

Anwendungsschicht (Application Layer)
Transportschicht (Transport Layer)
Internetschicht (Internet Layer)
Netzwerkschicht (Network Layer)

Kommunikation zwischen Rechnern

Aufgrund der hohen technischen Komplexität bei der Kommunikation zwischen Rechnern teilt man die verschiedenen Aufgaben in übereinanderliegende Schichten auf. Die Grundidee ist die ***Modularität.*** Das bedeutet:

- Jede Schicht hat nur eine Aufgabe (→ geringere Komplexität)
- Es werden Dienste an höhere Schichten bereitgestellt (durch Schnittstellen/Klassen/Methoden) werden und höhere Schichten benutzen die Dienste der unteren Schichten. (→ Implementierung bleibt verborgen)
- Die einzelnen Schichten sind (mit geringer Anpassung des Codes) austauschbar. (→ Man muss nicht immer alles von Grund auf programmieren, sondern kann auf bereits implementierte Funktionalitäten zurückgreifen.)

Netzwerkschicht (Network Layer)

Diese Schicht ist für die konkrete ***physikalische*** (elektrisch / optisch / Funk) ***Übertragung*** von einzelnen Bits über eine Leitung zuständig. Hierbei werden z. B. elektrische Signale über ein Kabel gesendet und an der Gegenstelle erkannt. Dabei wird ebenfalls dafür gesorgt, dass die ***Übertragung der Daten*** (durch Erkennen von Fehlern und Korrektur

von Fehlern) ***fehlerfrei bereitgestellt*** wird. Zudem wird auch die ***Zugriffssteuerung*** auf die Leitung geregelt, falls es durch mehrere Signale gleichzeitig zu einer sogenannten Kollision kommt.

Diese Schicht ist fast ausschließlich physikalisch anspruchsvoll, weshalb wir diese Schicht nicht weiter vertiefen werden.

MAC-Adressen

Netzwerkschicht (Network Layer)

Jede Netzwerk-Karte verfügt vom Hersteller über eine (eigentlich weltweit eindeutige) ***MAC-Adresse***. Diese hat die Form ***90:e6:ba:c1:76:fe***. Es handelt sich um 6 mal 2 Hexadezimal-Zeichen (0, 1, 2, … , 9, a, b, … , f) bzw. um 6 Bytes. In einem ***lokalen Netz*** wird ausschließlich über diese ***MAC-Adressen kommuniziert***. Die MAC-Adressen der Rechner sind es auch, die sich ein Switch „merkt“, um Datensendungen nur an den entsprechenden Rechner und nicht an alle anderen zu schicken.

→ Multiple Choice in Teams

Internetschicht (Internet Layer)

Das Internet ist ein großes Netzwerk, welches man als Graph darstellen können. Die Knoten wären die Geräte und die Kanten die Verbindungen/Kabel.

Die ***Internetschicht*** ist hauptsächlich zuständig für die ***Auswahl der Routen***/Wege zwischen den Rechnern/Knoten im Internet/Graph. Dabei macht sich diese Schicht die Netzwerkschicht zunutze, um eine verlustfreie Übertragen zwischen den einzelnen Geräten zu gewährleisten.

Um über lokale Netzwerkgrenzen ein Datenpaket zu verschicken, benötigt man eine adäquate Adressierung.

IP-Adressen (IPv4)

Internetschicht (Internet Layer)

→ siehe Präsentation IP-Adressen

IP-Adresse sind für die Adressierung über lokale Netzwerkgrenzen hinweg notwendig. Sie haben die Form ***91.66.185.199***. Es handelt sich um vier durch Punkte getrennte Bytes (je 8 Bit = 32 Bit) in Dezimaldarstellung. Jedes Byte kann also Werte von 0 bis 255 annehmen ($2^8 = 256$).

Zu jeder IP-Adresse gibt es eine ***Subnetzmaske*** (für uns nur ***/8 = 255.0.0.0***, ***/16 = 255.255.0.0*** und ***/24 = 255.255.255.0***), in derselben Darstellung wie die IP-Adresse. Sie teilt die IP-Adresse in ***Netzanteil*** und ***Host-Anteil*** auf. Beispiel:

91 . 66 . 185 . 199
255 . 255 . 0 . 0
Netzanteil: 91.66 Hostanteil: 185.199

Alle Rechner mit ***demselben Netzanteil befinden sich in demselben Netz*** (***Adresse des Netzwerks: 91.66.0.0***) und können direkt miteinander (über einen Switch – und damit über die ***MAC-Adressen***) kommunizieren.

Die ***erste IP-Adresse*** in einem Netzwerk ist die ***IP-Adresse für das Netzwerk selbst***. Die ***letzte IP-Adresse*** ist für den ***Broadcast*** (engl. Rundruf) reserviert:

91.66.185.199 /16 → Netzwerkadresse: 91.66.0.0 Broadcast-Adresse: 91.66.255.255

→ **Multiple Choice in Teams**

Konventionen zur Vergabe von IP-Adressen im Netzwerk: Die ***Router*** bekommen die ***ersten freien*** (nicht die Netz- und Broadcast-) ***Adressen*** im Netzwerk. Die ***Server*** belegen die ***letzten freien Adressen***. Die „normalen“ Rechner bekommen beliebige freie IP-Adressen in der Mitte.

→ **Multiple Choice in Teams**

Aufgabe:

Bearbeite das Arbeitsskript „2 Netzwerkaufbau mit Filius“.

Private Netzwerke

Aufgabe:

Versuche in WebNetSim als Haushalt einen Webserver zu betreiben, dessen Dienst andere Teilnehmer nutzen können.

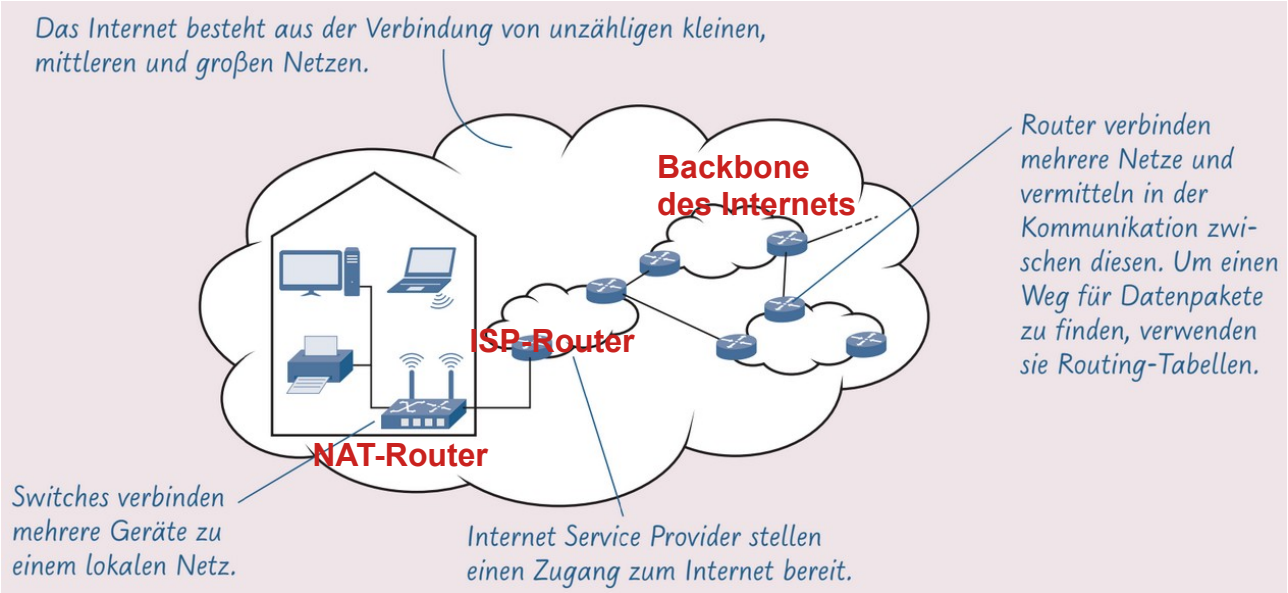
a) Welche Schritte sind dafür zusätzlich nötig im Vergleich zu den Rollen Provider bzw. Webseitenanbieter?

b) Warum könnten dieses zusätzlichen Schritte nötig sein?

c) Unter welcher IP-Adresse ist dieser Webserver erreichbar?

Heimrouter sind normale Router mit Zusatzfunktionen. Sie haben genau zwei Anschlüsse (LAN und WAN → Local Area Network vs. Wide Area Network), spannen daheim ein ***privates Netzwerk*** auf und schützen es durch eine ***Firewall*** (sie lassen keinen Verbindungsaufbau von außen zu). Zudem stellen sie ***WLAN zur Verfügung*** und ***„übersetzen" die lokalen IP-Adressen*** (***Network Address Translation***). Daher nennt man sie auch ***NAT-Router***.

Ein weiterer besonderer Router ist der ***ISP-Router*** (***Internet Service Provider***). Sie stellen uns eine Internetverbindung gegen Bezahlung zur Verfügung z. B. Telekom, Vodafone, M-net usw. https://en.wikipedia.org/wiki/Internet_backbone



Aufgabe zum Aufbau des Internets (Nachverfolung Routing):

Falls du ein eigenes Gerät hast, kannst du selbst mitmachen.

Öffne die Eingabeaufforderung und tippe folgendes ein:

tracert **domain z.B. evbg.de**

Sieh dir die Ausgabe an und versuche anzugeben, welche IP-Adresse zu welchem Gerät gehören. Falls du keine Idee hast, google die IP-Adresse.

Transportschicht (Transport Layer)

Die ***Transportschicht*** ist dafür zuständig eine ***Ende-zu-Ende-Verbindung*** (Übertragungskanal) über Netzwerkgrenzen hinweg aufzubauen und die zu übertragenden Daten in ***Pakete zerlegt*** und diese einzeln verschickt.

Für die Verbindung zwischen den Endgeräten macht sich die Transportsicht die Routenauswahl/Wegsuche der Internetschicht zunutze, welche wiederum die Netzwerkschicht für die sichere Übertragung zwischen zwei Geräten nutzt.

Transportprotokolle TCP und UDP

Transportschicht (Transport Layer)

Bei Datenübertragungen über ein Netzwerk unterscheidet man prinzipiell zwischen ***TCP*** (***transmission control protocol***) und ***UDP*** (***user datagram protocol***).

- ***TCP*** ist eine ***verlustsichere*** Datenübertragung. Sicher in dem Sinne, dass für jedes abgeschickte Datenpaket von der Gegenstelle eine Empfangsbestätigung verlangt wird. Erfolgt diese nicht in einer bestimmten Zeit, so wird das entsprechende Paket erneut gesendet. TCP verwendet man immer dann, wenn eine unvollständige Datensendung keinen Sinn macht, z. B. also bei Webseiten (http), Datenbank-Anfragen, E-Mails (smtp, imap, pop), ...
- ***UDP*** ist eine ***verlustunsichere*** Verbindung. Unsicher in dem Sinne, dass die Datenpakete einfach versendet werden ohne sich Gedanken darüber zu machen, ob sie auch bei der Gegenstelle ankommen. UDP verwendet man häufig bei zeitkritischen Übertragungen, bei denen man auch mal auf einen kleinen Teil der Information verzichten kann, z. B. beim Streamen von Audio- oder Video-Dateien. Dort fällt es am Client nicht auf, wenn einmal eins von 24 Bildern pro Sekunde des Films nicht ankommt. Ein Nachreichen eines im Netz verunglückten Bildes gäbe hier wenig Sinn.

Ports

Transportschicht (Transport Layer)

Eine Verbindung z. B. via TCP wird immer auf einem bestimmten Port aufgebaut, damit dem anderen Rechner/Server klar ist, welcher Dienst angefragt wird. Denn auf einem Rechner können viele (Server-)Dienste gleichzeitig laufen oder umgekehrt ist ein ganzer Rechnerverbund (Serverfarm) für einen Dienst zuständig (z. B. Google, …).

Ein ***Port*** ist eine Zahl zwischen ***0 und 65535*** (also 2^{16} = 2 Bytes) und es gibt eine Liste Liste der standardisierten/well-known Ports für bekannte Dienste.

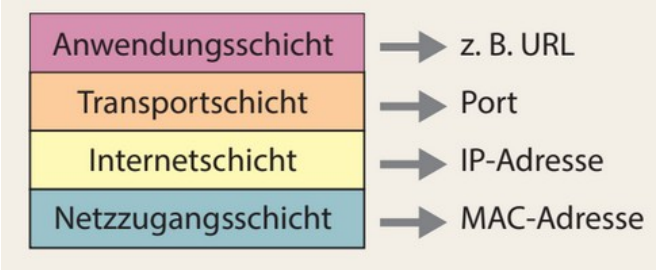
Ports aus den nicht reservierten Bereichen stehen für eigene Anwendungen oder die Kommunikation nach dem Verbindungsaufbau zur Verfügung. Diese Port-Nummern können dann bei jeder einzelnen Verbindung wechseln und sind nicht lange gebunden.

Anwendungsschicht (Application Layer)

Die ***Anwendungsschicht*** ist zuständig für den ***Austausch anwendungsspezifischer Daten*** (Webseite, Mail, Spieldaten, …) in Zusammenarbeit mit Anwendungsprogrammen (Mail-Client/Server, Webserver – Webclient, Spiele-Server mit Spiel, …).

Dabei nutzt die Anwendungsschicht die darunterliegenden Schichten für die Kommunikation.

Anwendungsschicht (Application Layer)



Protokolle der Anwendungsschicht

So gibt es z.B. folgende gut bekannten Protokolle und diese bekannten Dienste liegen auf wohl bekannten Ports (https://de.wikipedia.org/wiki/Liste_der_standardisierten_Ports):

- ***http*** (hypertext transfer protocol) regelt die Kommunikation zwischen Webserver und Web-Client und benutzt Port 80; ***https*** (hypertext transfer protocol secure) benutzt Port 443 bei verschlüsselter Webkommunikation
- ***smtp*** (simple mail transfer protocol) zum Verschicken, ***imap*** (internet message access protocol) und ***pop*** (post office protocol) zum Abrufen von Mails. (***smtp*** nutzt Port 25, ***imap*** nutzt Port 143, ***pop*** nutzt Port 110, ***tls*** (z. B. für smtp mit Verschlüsselung) nutzt Port 587)
- ***ftp*** (file transfer protocol) regelt die Kommunikation zwischen einem Datei-Server und Datei-Client und benutzt Port 20 (data transfer) und 21 (command)
- ***dns*** (domain name system) regelt die Kommunikation zwischen Name-Server und Name-Client und nutzt Port 53

Aufgabe zum Domain-Name-System:

Öffne die Eingabeaufforderung und tippe folgendes ein:

nslookup **domain IP-Adresse-des-DNS**

Beispiel: nslookup joachimhofmann.org 8.8.8.8 ← IP-Adresse des Google-DNS-Servers

Aufgabe zur Portangabe (http + https):

Ein Port wird mit : an die IP-Adresse gehängt!

→ **Multiple Choice in Teams**

Gib folgende Werte in die Browserzeile ein, notiere das Ergebnis und ***deute die Ergebnisse***:

Eingabe	Funktioniert? Und warum?
http://joachimhofmann.org:80	
https://joachimhofmann.org:443	
https://joachimhofmann.org:80	
http://joachimhofmann.org:443	

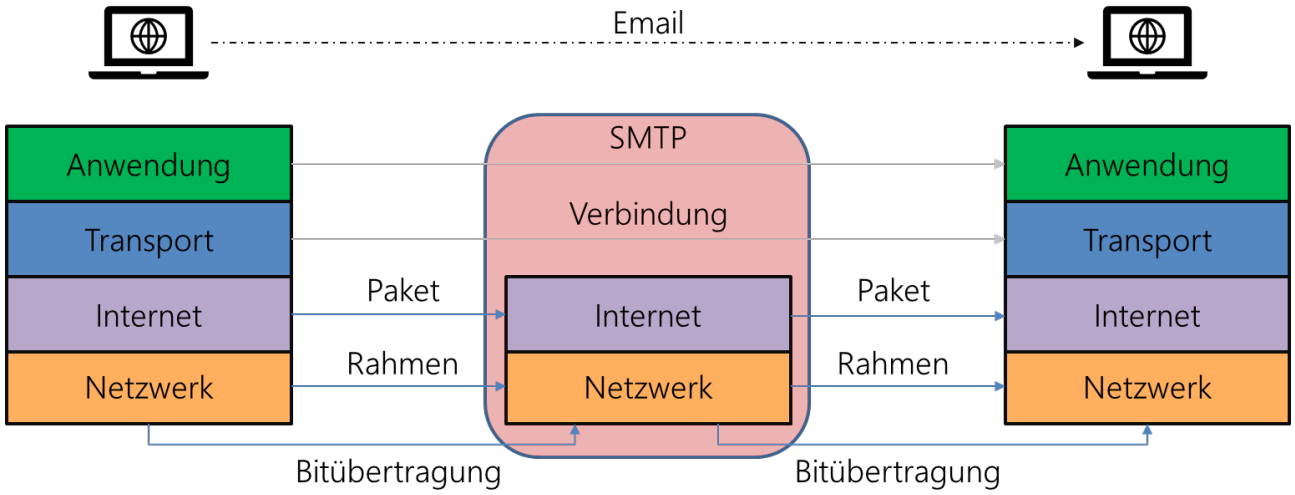
Bei keiner Protokollangabe wird erstmal http verwendet!

joachimhofmann.org:80
joachimhofmann.org:443

Bei keiner Portangabe wird erstmal Port 80 verwendet!

joachimhofmann.org

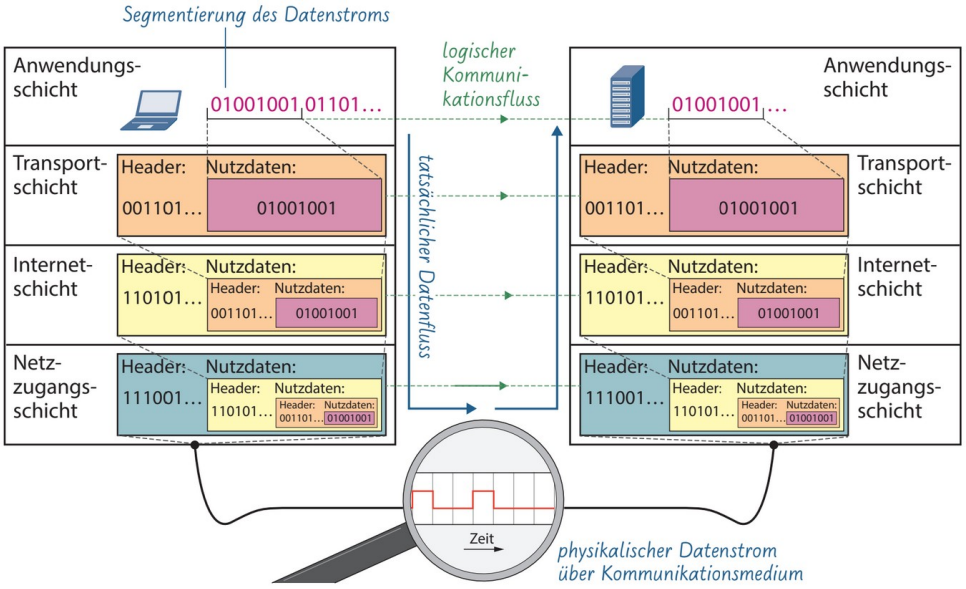
Kommunikation im Schichtenmodell



Protokolle

Jede Schicht nutzt verschiedene Protokolle, um ihre Aufgabe für die entsprechende Schicht zu erfüllen. Unter einem ***Protokoll*** versteht man in der Informatik einen Satz von Regeln, welche die Kommunikation eines Server-Dienstes exakt regeln.

TCP/IP-Modell	Wichtige Protokolle
Anwendungsschicht	HTTP(S), SSH, NTP, IMAP, SMTP, IPP, POP3, RTSP, …und viele weitere
Transportschicht	TCP, UDP
Internetschicht	IP
Netzzugangsschicht	1000Base-T (Kupferkabel), 1000Base-SX/LX (Glasfaserkabel), IEEE 802.11 (Funk)



Aufgabe:

Bearbeite das Arbeitsskript „3 Kommunikation mit Filius“.

Aufgabe zur Mailkommunikation (smtp):

Wie kommunizieren jetzt die Rechner bzw. was schreiben sie sich?

Öffne dein Filiusprojekt und wechsel in den Aktionsmodus. Versende eine Mail, bei der du in alle Felder etwas einfügst. Wechsel zum Mailserver und klicke auf Log-Fenster.

a) Analysiere die Kommunikation zwischen Client und Server, wie der Ablauf des smtp-Protokolls aussieht.

In der Realität läuft dies genauso wie in Filius ab. Zum Verbindungsaufbau können die Programme ***Telnet*** (ohne Verschlüsselung) und ***openssl*** (mit Verschlüsselung) genutzt werden, die eine TCP-Verbindung zu einem Server aufbauen können. **So ist es möglich mit einem Server auf Anwendungsprotokollebene zu kommunizieren.**

Unsere Schul-E-Mail (@student.evbg.de) wird vom Outlook-Mailserver von Microsoft gehostet und wir wollen uns jetzt mit dem Outlook-Server mittels openssl verbinden und eine Mail über das Anwendungsprotokoll smtp versenden. Zur Authentifizierung müssen wir Benutzername und Passwort in Base64-codiert angeben. Daher:

b) Ermittle zuerst dein Benutzername und Passwort in Base64 mithilfe des BlueJ-Projekts ***Base64***.

c) Sende jetzt deine Mail mit openssl. Die Verbindungskonfiguration ergibt sich aus den Daten auf den Supportseite von Microsoft. (***Office365-SMTP-Einstellungen***)

s_client -connect smtp-mail.outlook.com:587 -crlf -starttls smtp -nocommands

ehlo (Du kannst die Befehle hier aus dem PDF herauskopieren!)

auth login

dein Base64-Benutzername (Strg+c kopieren; Strg+v oder Rechtsklick zum Einfügen)

dein Base64-Passwort

mail from: **deine Klartext-Mailadresse**

rcpt to: **Empfänger Klartext-Mailadresse**

data (jetzt beginnt die Mail: zuerst Metadaten der Mail)

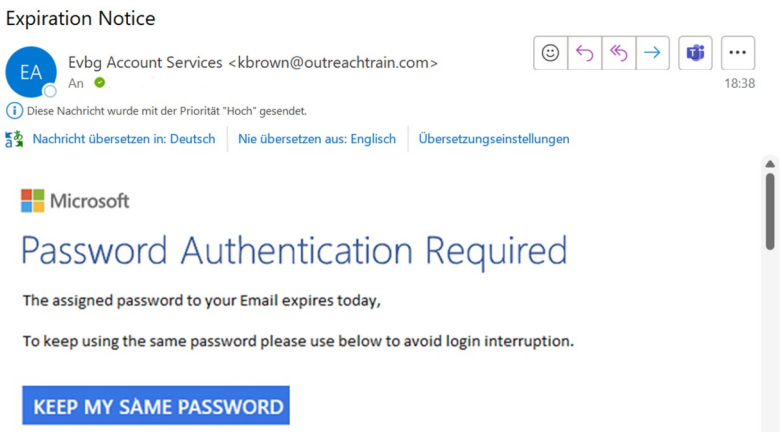
From: **Anzeigename <deine Klartext-Mailadresse>** **oder** **deine Klartext-Mailadresse**
To: **Empfänger Klartext-Mailadresse**
Subject: **Betreff**
(Leerzeile)
Eigentlicher Mail-Inhalt (Vermeide Umlaute und ß)

. (Mail beenden mit <Taste Enter> . <Taste Enter>)

QUIT

d) Überprüfe, ob deine Mail bei deiner Empfängeradresse ankam.

e) **Woran erkennt man ganz schnell, dass dies eine Phishing-Mail ist?**



Mit der Authentifizierung ist es nicht mehr möglich, dass man bei ***mail from*** bzw. auch in der Mail bei ***From:*** eine andere Mail-Adresse angibt, als die Adresse mit der man sich authentisiert hat (vor ein paar Jahren ging das tatsächlich noch). Aber man kann den Anzeigename geschickt nutzen, um den Empfänger Seriosität vorzugaukeln und dazu bewegt auf einen Link zu klicken oder ein Passwort einzugeben und „abzuphishen“.

Aufgabe zur Webkommunikation (http + https):

a) Zurück zu Filius zur Kommunikation im Webserver. Was schickt der Client in Filius an den Server und was antwortet der Server? Tipp: es sind zwei Anfragen!

b) Öffne openssl und gib folgenden Befehl für den Verbindungsaufbau ein:

Tipp: Der Timeout kommt sehr schnell, daher kann man die GET-Anfrage im Texteditor vorbereiten und reinkopieren!

s_client -connect joachimhofmann.org:443 -crlf

Leerzeichen (Du kannst die Befehle hier aus dem PDF herauskopieren!)

```
GET / HTTP/1.1
Host: joachimhofmann.org
```

zweimal Enter drücken

Wurde die Webseite zurückgeschickt?

c) Was ist das alles, was nach dem Verbindungsaufbau angezeigt wird? Tipp: vorheriges Themengebiet Codierung & Verschlüsselung!

d) Rufe mit deinem Webbrowser die folgenden Webseiten auf:

https://joachimhofmann.org
https://joachimhofmann.org/index.html
https://joachimhofmann.org/hallo.html
https://joachimhofmann.org/Klasse11/
https://joachimhofmann.org/Klasse11/3%20Kommunikation%20in%20Netzwerken/
https://joachimhofmann.org/Klasse10/Git.pdf

Beantworte aufgrund deiner obigen Ergebnisse folgende Fragen:

- Welche Datei wird bei einer GET-Anfrage angefordert, wenn keine explizite Datei angegeben wird, sprich nur ein / ?

- Was könnten die Werte nach dem Domainnamen auf dem Server entsprechen?

- Woran erkennt man, dass eine bestimmte Datei angefordert wird?

e) Formuliere im Texteditor die GET-Abfrage, welche die pdf-Datei unter diesem Link https://joachimhofmann.org/Klasse10/Git.pdf abfragt. Teste deine GET-Abfrage mit openssl.

Netzwerkprogrammierung in Java

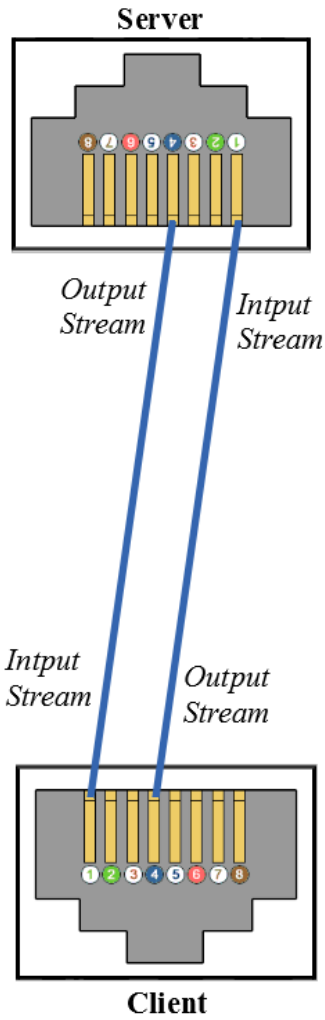
Grundlegendes

Ein ***Socket*** ist ein Objekt, welches eine TCP-Verbindung mittels IP-Adresse und Port zu einem anderen Socket eines anderen (entfernten) Programm aufbaut. Anschließend kann man darüber auf Anwendungsebene mit dem Server kommunizieren (analog zu openssl).

Es gibt zwei Arten von Sockets:

- ***ServerSocket*** bzw. ***SSLServerSocket*** lauscht beim Server auf einem Port und nimmt Verbindungsanfragen an. Der ServerSocket ist nur für Verbindungsannahme zuständig.
- Die Kommunikation läuft bei Client und Server über ein ***Socket*** bzw. ***SSLSocket***.

Aufbau einer Client-Server-Verbindung mit Sockets



1. ***Der Server*** erzeugt ein ***ServerSocket***, welches auf dem angegeben ***Port*** lauscht.
2. ***Der Client*** erstellt sich ***Socket*** und gibt die entsprechende ***IP-Adresse*** und den (lauschenden) ***Port*** des Servers an und sendet damit eine Verbindungsanfrage an den ServerSocket.
3. Erhält ***der Server*** diese Anfrage, so kann er diese akzeptieren und erzeugt damit ***ebenfalls ein Socket***, welcher mit dem Socket des Clients verbunden ist.
   Der ServerSocket kann nun ***erneut lauschen*** und weitere Verbindungen annehmen.
4. Jeder der beiden Sockets besitzt nun einen Sendekanal (***OutputStream***) und einen Empfangskanal (***InputStream***). Was die eine Seite sendet, landet bei der anderen Seite „im Briefkasten“. Dort kann es dann herausgenommen und verarbeitet werden. (siehe Bild)
5. Die konkrete Kommunikation erfolgt nach ***Protokoll*** (z. B. http, smtp oder ein selbst geschriebenes) wie bereits mit openssl kennengelernt.

Aufgabe Webclient:

Programmiere einen Webclient → **siehe Präsentation „4 Implementierung eines Webclients“**

Aufgabe Cookies:

a)
- Öffne online die Webseite Cookie.html.
- Gib einen Wert für den Cookie an und setze ihn.
- Schließe den Browser.
- Öffne den Browser erneut und gehe erneut auf die Webseite Cookie.html.
- Was siehst du?

b) Sieh dir den Cookie im Browser an: (funktioniert je nach Browser unterschiedlich)
Für Edge: … → Einstellungen → Cookies und Webseitenberechtigungen → Verwalten und Löschen von Cookies und Websitedaten → Alle Cookies und Websitedaten anzeigen → suche joachimhofmann.org → klappe den Reiter auf und klicken bei dem einen Cookie auf > → Klappe die Daten des Cookies auf:

- Welchen Namen, Inhalt, Domäneneintrag und Pfad hat der Cookie?
- Wie lange ist dieser Cookie gültig?
- Wird der Cookie nach Ablauf der Gültigkeit gelöscht? (Internetrecherche)

c) Sieh dir den Inhalt weiterer Cookies von anderen Webseiten an. Kannst du deren Inhalt nachvollziehen? Falls nein, warum nicht?

Vertiefungsbereich zu Kommunikation in Netzwerken

Optionale Implementierungen für ganz Schnelle

Eingabezeilen-basierter Webbrowser

Öffne deine Implementierung deines Webbrowsers. Erstelle dir im Konstruktor einen neuen Scanner, der auf System.in (das ist die Konsoleneingabe) lauscht.

Schreibe eine Methode starteWebbrowser(), in welcher der Scanner mittels einer while(true){}-Wiederholung auf (neue) Eingaben wartet. Für jede Eingabe soll, sofern sinnvoll, eine Webanfrage gestartet werden.

Nimm Anpassungen vor, dass verschiedene Varianten der Domainangabe funktionieren als auch Pfadangaben verarbeitet werden:

joachimhofmann.org
https://joachimhofmann.org
joachimhofmann.org/index.html
https://joachimhofmann.org/Klasse11/3%20Kommunikation%20in%20Netzwerken/cookie.html

Nutze hierfür Methoden, die die Klasse String bereitstellt:
https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

Falls der Nutzer einen nicht korrekten Link eingibt, dann nutze einen try-catch-Block, um die entstandene Exception abzufangen und den Fehler auszugeben (System.err.println()) gefolgt von einem continue, damit das Prorgramm nicht abstürzt und einfach auf die nächste Eingabe wartet.