

Elisabeth-von-Thüringen-Gymnasium
Schuljahr 2021/22

Facharbeit im Fach Informatik

Betreuerin: Frau Becker

**Netzwerktechnik im lokalen Netzwerk anhand der Implementation
einer beispielhaften Netzwerkanwendung**

vorgelegt von

Lennart Protte

Köln, den 23. Februar 2022

Lennart Protte
Heinrich-Imig-Str. 50
50354 Hürth
Tel: 0163 6827482
E-Mail: lenni.protte@gmx.de

Inhalt:	Seite:
1. Einleitung	4
2. Der Messenger “Quap”	
2.1. Verwendete Programme	5
2.2. Datenbanken	
2.3. Nebenläufigkeit und Multithreading	
2.3.1. Nebenläufige Programmierung	6
2.4. Netzwerktechnik	
2.4.1. Das Package java.net	7
3. Architektur von Netzwerken	
3.1. Netzwerktopologie	8
4. Kommunikation in Netzwerken	
4.1. Das OSI Schichtmodell	9
4.2. Protokolle	
4.2.1. Transfer Control Protocol - TCP	
4.2.2. User Datagramm Protocol - UDP	10
4.2.3. Internet Protocol - IP	
4.3. Adressierung	
5. Client-Server Modell	11
5.1. Verbindungsaufbau	
5.2. Datentransfer	12
5.3. Verbindungsabbau	
6. Schlussbemerkung	13
6.1. Ausblick	
7. Literaturverzeichnis	14
8. Anhang	

Vorwort:

Aufgrund der Tatsache, dass ich mich bereits in einem vorherigen praktischen Projekt mit dem Thema „Kommunikation im lokalen Netzwerk“ auseinandergesetzt habe, habe ich mich dazu entschlossen, meine Facharbeit auf diesem Interesse aufzubauen und so auch meine Kenntnisse weiter zu vertiefen.

Im Rahmen meiner Facharbeit habe ich die Idee eines Messenger wieder aufgegriffen und mit dem neuen Ansatz einer TCP/IP basierten Kommunikation, anstatt der Verwendung von UDP, als auch neuen Erkenntnissen umgesetzt.

Im praktischen Teil konnte ich meine guten Vorkenntnisse in der Sprache Java und der Umsetzung von Multithreading, nutzen.

Während der Entwicklung meines praktischen Teils habe ich neben den Themen Netzwerktechnik und Netzwerkarchitektur auch FXML in der Kombination mit JavaFX sowie Datenbanken erlernt.

1. Einleitung

Diese Arbeit behandelt die Grundlagen der Netzwerktechnologie im lokalen Netzwerk anhand der Implementation einer Netzwerkanwendung in Java zur Demonstration der Kommunikation und Architektur innerhalb eines solchen Netzwerkes. Es handelt sich dabei um ein allgegenwärtiges Thema, da Menschen seit jeher kommunizieren und dies zunehmend auch digital.

Diese Facharbeit soll die dahintersteckenden Technologien erklären und deren praktische Umsetzung anhand der Java Applikation „Quap“ demonstrieren. Dabei konzentriert sich diese Facharbeit ausschließlich auf die Technologie und Architektur eines lokalen Netzwerkes, mit besonderem Fokus auf der Client-Server Kommunikation über TCP/IP.

Um zu einem abschließenden und nachvollziehbaren Ergebnis zu gelangen, wird im Verlaufe dieser Facharbeit die Theorie und die Umsetzung, innerhalb des praktischen Teiles dieser Facharbeit, schrittweise nähergebracht.

Zunächst wird die grundlegende Architektur von Netzwerken erläutert, als auch relevante Klassen der Java-Bibliothek dargestellt.

Zudem wird im ersten Teil der Facharbeit noch auf weitere Technologien eingegangen, die oft in Kombination mit der Netzwerktechnik, so auch im praktischen Teil dieser Arbeit, Verwendung finden. Anschließend wird auf die Kommunikation in Netzwerken näher eingegangen, welche ebenfalls mit Beispielen aus dem praktischen Teil veranschaulicht wird. Hiernach wird das Client-Server Modell, in Hinblick auf die verschiedenen Prozesse, welche innerhalb eines Lebenszyklus eines Clients stattfinden, untersucht.

2. Der Messenger “Quap”

„Quap“ ist der Name des Messengers, der im Rahmen dieser Arbeit entwickelt wurde.¹ Es handelt sich dabei um eine Client-Server Anwendung, die es dem Nutzer ermöglicht über eine grafische Oberfläche mit anderen Nutzern, von verschiedenen Geräten aus Nachrichten auszutauschen. Das Projekt liegt als „Multimodule Maven Projekt“ vor und wurde in Java, FXML, SQL und CSS geschrieben.

Entwickelt wurde das gesamte Projekt mit dem Fokus auf Anonymität. So werden Nachrichten nur auf den Geräten selbst gespeichert und selbst Einladungen zu Chatgruppen, oder Freundschaftsanfragen werden nie zwischen gespeichert.

¹ Anhang - VIII

2.1. Verwendete Programme

Im Rahmen der Entwicklung wurde vor allem in der IDE „IntelliJ Ultimate“ entwickelt, da der größte Teil des Projektes in Java geschrieben wurde. Für die „Postgres-Datenbank“, welche auf dem Server verwendet wurde, hat das Programm „DataGrip“ Verwendung gefunden. Die Nachrichten, werden auf den jeweiligen Geräten in einer „SQLite-Datenbank“ gespeichert, so dass dafür keine weitere Installation eines Programmes notwendig ist. Für die Gestaltung der Benutzeroberfläche in Echtzeit wurde „SceneBuilder“ genutzt. Darüber hinaus ist für das gesamte Projekt das Tool „Git“ zum Einsatz gekommen. Git ist ein Versionskontrollsystem, welches Möglichkeiten für „branching“, „merging“ und arbeiten mit der „Commit-History“ bietet.² Das Repository des Projektes liegt auf „Github“, einer „Code-Hosting-Plattform“ für Versionskontrolle und Zusammenarbeit, so können Änderungen überwacht als auch jederzeit wieder rückgängig gemacht werden.^{3 4 5}

2.2. Datenbanken

Im Hinblick auf die Datenspeicherung im Projekt Quap, wo bei der Nutzung eine Reihe von Nutzerdaten anfallen, ergeben sich Vorteile und Nachteile, die aus der Verwendung von relationalen Datenbanken zur Speicherung dieser Daten hervorgehen.

Relationale Datenbanken eignen sich gut für die Datenspeicherung im Projekt Quap, da es dort auf eine genaue und nachvollziehbare Datenspeicherung ankommt und weniger auf die schnelle Datenspeicherung von großen Datensätzen. In dem Projekt greift der Server⁶ auf eine „Postgres-Datenbank“ zu, in der die Nutzeraccounts, Chatgruppen Zugehörigkeiten als auch Freundschaftsbeziehungen gespeichert werden, um so eine dauerhafte und zentrale Speicherung von Daten zu ermöglichen. Auf dem Client⁷ verwaltet eine „SQLite-Datenbank“ die Nachrichten sowie bekannte Accounts des Nutzers.

2.3. Nebenläufigkeit und Multithreading

Durch Multithreading wird im Projekt Quap, sowohl die Ausführgeschwindigkeit erhöht als

² Vgl. <https://git-scm.com>

³ Vlg. <https://docs.github.com/en/get-started/quickstart/hello-world>

⁴ Vgl. https://www.thomas-krenn.com/de/wiki/Git_Grundbegriffe

⁵ <https://github.com/learpro/Quap/tree/term-paper>

⁶ Anhang - X

⁷ Anhang - IX

auch ist es möglich, dass mehrere Prozesse gleichzeitig ablaufen.

*“Es handelt es sich um Verfahren oder Techniken, mit denen sich mehrere Threads eines Prozesses gleichzeitig oder quasi-gleichzeitig (pseudo-gleichzeitig) ausführen lassen. Als Threads werden einzelne Ausführungsstränge von Prozessen bezeichnet”.*⁸

Die parallele Abarbeitung mehrerer Threads wird Nebenläufigkeit genannt, was die grundlegende Eigenschaft, von Multithreading und Multitasking darstellt.⁹

Dabei gilt für Prozesse, welche nebenläufig ablaufen, das Kausalitätsprinzip der Prozesssynchronität. Dieses Prinzip besagt, dass Prozesse nebenläufig ablaufen können, wenn diese nicht kausal voneinander abhängig sind, als auch wenn keine dieser Prozesse das Resultat eines anderen benötigt.¹⁰ Es wird dabei zwischen scheinbarer Nebenläufigkeit und echter Nebenläufigkeit unterschieden.

2.3.1. Nebenläufige Programmierung

Im Projekt Quap kümmern sich die „ClientHandler“¹¹ als „Threads“ um die Anfragen der Clients. Diese werden vom Server in einem „ExecutorService“ verwaltet.¹²

Dies wird in Java ermöglicht, indem bestimmte Code-Abschnitte als „Thread“ laufen (ein Java Thread ist eine Instanz von `java.lang.Thread` oder eine Unterklasse davon). Dabei können Threads über gemeinsame Variablen kommunizieren und laufen so lange im Java-Scheduler, bis sie entweder unterbrochen werden oder sich von selbst beenden.

Bei bestimmten Anwendungsfällen kann es dazu kommen, dass mehrere Threads zeitgleich auf dieselbe Methode zugreifen, wodurch Fehler entstehen können. Diese wird als kritischer Bereich bezeichnet. Um dies zu verhindern, bietet Java das Schlüsselwort „synchronized“, welches nur einen Thread zur gleichen Zeit auf diesen Abschnitt zugreifen lässt.¹³

2.4. Netzwerktechnik

Netzwerke bilden die Grundlage für die Kommunikation zwischen einzelnen Geräten und sind damit eine Grundvoraussetzung für Messenger wie Quap.

Als Netzwerk wird der physikalische und logische Verbund aus mehreren Geräten definiert,

⁸ <https://www.storage-insider.de/was-ist-multithreading-a-1017586/>

⁹ Vgl. <https://www.storage-insider.de/was-ist-multithreading-a-1017586/>

¹⁰ Vlg. http://www4.cs.fau.de/Lehre/WS13/V_SP2/Vorlesung/Folien/SP2-101-A6.pdf

¹¹ Anhang - III

¹² Anhang – II, Z.18

¹³ Vlg. <https://www.lehre.dhbw-stuttgart.de/~kfg/java/ccjava.pdf>, S.102, 7.5 Synchronisation

welche in einem Netzwerk auch als Knoten oder Netzknoten bezeichnet werden.¹⁴ Basierend auf Übertragungstechniken, Protokollen und Systemen, wird die Kommunikation zwischen den Knoten ermöglicht. Dadurch ergeben sich beispielsweise Vorteile wie die Möglichkeit Daten gemeinsam zu nutzen und zu verwalten, den Austausch von Daten, das Aufteilen von Rechenleistung und Speicherkapazität als auch das Setzen von Berechtigungen und Zuständigkeiten.

2.4.1. Das Package java.net

Sockets stellen die Kommunikation zwischen zwei Geräten zur Verfügung und finden damit Verwendung in Quap. So wird eine „java.net.Socket Klasse“ auf dem Client erstellt, welche versucht sich mit der „java.net.ServerSocket“ auf dem Server zu verbinden.¹⁵ ¹⁶ Wenn diese Verbindung zu Stande kommt, wird auf dem Server ein neues „Socket Objekt“ erzeugt, über welches die Kommunikation zum Client abläuft.¹⁷

Die „ServerSocket“ kann mit den Parametern Port (als int), „Backlog“ (als int) und IP-Adresse (als java.net.InetAddress, welche eine IP-Adresse repräsentiert) initialisiert werden. Dabei gibt der „Backlog“ an, wie viele eingehende Verbindungsanfragen gespeichert werden können, bis diese akzeptiert werden. Insgesamt besitzt die Klasse „ServerSocket“ vier Konstruktoren und die „Socket Klasse“, welche für den Client verwendet wird, besitzt fünf Konstruktoren. Die Adresse und der Port des Servers müssen bekannt sein, um eine Verbindung zu der Socket des Servers zu ermöglichen.

3. Architektur von Netzwerken

Die Architektur von Netzwerken bildet den konzeptionellen Rahmen, der für die Kommunikation innerhalb eines Netzwerkes und somit auch für Quap unerlässlich ist. Dabei ist es für ein Netzwerk nicht zwingend erforderlich ist, dass sich darin ein Server, Switch oder ein anderes Gerät befindet, welches als zentraler Knotenpunkt fungiert.¹⁸ Ein wichtiger Aspekt neben der Topologie eines Netzwerkes und dem Zugangsverfahren, welches anhand des OSI Referenzmodells erklärt wird, ist der Datendurchsatz in einem

¹⁴ Vgl. <https://www.elektronik-kompendium.de/sites/net/0503271.htm>

¹⁵ Vgl. <https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html>

¹⁶ Anhang – VII, Z.68

¹⁷ Anhang – II, Z.41-68

¹⁸ Vgl. https://lehrerfortbildung-bw.de/u_matnatech/informatik/gym/bp2016/fb1/3_rechner_netze/1_hintergrund/3_kommunikation/1_netzwerk/

Netzwerk. Der Datendurchsatz wird üblicherweise in Einheiten wie Megabit pro Sekunde (MBit/s), Gigabit pro Sekunde (GBit/s) oder Megabyte pro Sekunde (MB/s) angegeben. Dabei entsprechen tausend MBit/s einem GBit/s und 125 MB/s.

3.1. Netzwerktopologie

„Wenn viele Geräte miteinander verbunden werden sollen, stellt sich die Frage, wie diese miteinander zu verbinden sind. Es geht hier nicht darum, ob mit oder ohne Kabel verbunden wird, sondern um einen rein topologischen (=die Anordnung im Raum betreffenden) Sachverhalt.“¹⁹.

Der topologische Aufbau eines Netzwerkes ist sowohl für die Belastbarkeit des Netzwerkes als auch für die Kommunikationsfähigkeit mit anderen Netzwerken zuständig.

Es gibt drei gängige Topologien, darunter die Stern-Topologie, bei welcher alle Verbindungen über einen zentralen Netzknoten führen, wodurch einfaches Routing im zentralen Netzwerknoten ermöglicht wird, aber auf der anderen Seite eine hohe Ausfallwahrscheinlichkeit besteht. Die Sterntopologie ist die gängige Topologie von lokalen Netzwerken, wobei der Router den zentralen Netzknoten stellt. Bei einer Client-Server Kommunikation im lokalen Netzwerk würde daher jeder Datentransfer über den Router laufen.

Eine Erweiterung der Stern-Topologie, bei welcher für jede Verbindung ein Switch verwendet wird, da so viele Knoten in einem Netzwerk verbunden werden können, ist die Baum-Topologie.

Bei der Vermischten-Topologie können einzelne Knoten auch über mehrere Wege miteinander verbunden sein, wodurch auch bei einem Ausfall einzelner Knoten noch eine Verbindung besteht.

4. Kommunikation in Netzwerken

Kommunikation erfolgt in Netzwerken durch Netzwerkprotokolle und ist dabei nach dem OSI-Referenzmodell strukturiert. Für das Internet ist dabei das DoD-Schichtmodell, welches aus vier aufeinander aufbauenden Schichten besteht und eine vereinfachte Version des OSI-Referenzmodells darstellt, maßgebend.

¹⁹ https://lehrerfortbildung-bw.de/u_matnatech/informatik/gym/bp2016/fb1/3_rechner_netze/1_hintergrund/3_kommunikation/1_netzwerk/

Das Internet-Protokolle überträgt Datenpakete über mehrere Punkt-zu-Punkt-Verbindungen und stellt auf dieser Basis Verbindungen zwischen den einzelnen Netzknoten her.

Bei der Kommunikation zwischen zwei Instanzen muss in vertikale, der Kommunikation zwischen Diensten und in horizontale, der Kommunikation über Protokolle unterschieden werden.²⁰

4.1. Das OSI Referenzmodell

Wie bereits in Kapitel 3. beschrieben, ist das OSI-Referenzmodell, bei dem es sich um ein Schichtmodell handelt, für das Zugangsverfahren bei Netzwerken zuständig da es die Kommunikation zwischen Geräten und Netzwerken definiert.²¹ Das Referenzmodell lässt sich in sieben einzelne Schichten, mit jeweils einer spezifischen Aufgabe unterteilen, darunter die Anwendungsschicht, Darstellungsschicht, Kommunikationsschicht, Transportschicht, Vermittlungsschicht, Sicherungsschicht und die Bitübertragungsschicht, wobei die ersten vier Schichten die transportierenden und die anderen Schichten die anwendungsorientierten Schichten darstellen.²² Das DoD-Schichtmodell besitzt lediglich die Anwendungsschicht, Transportschicht, Internetschicht und die Netzzugangsschicht.

4.2. Protokolle

In Quap wird die TCP/IP Protokoll-Familie für den Transport von Datenpaketen verwendet. Die Abkürzung TCP/IP steht für die beiden Protokolle Transmission Control Protocol (TCP) und Internet Protocol (IP), die im Folgenden näher erläutert werden.

Protokolle können als Regelwerk der Kommunikation zwischen Geräten angesehen werden, womit sie vergleichbar mit der Syntax der menschlichen Sprache wären.²³

Damit fällt unter die Aufgabe von Protokollen, das Übertragen von Daten, das Format, die Vollständigkeit und die Reihenfolge der Übertragung als auch die Zugriffskontrolle.²⁴

4.2.1. Transfer Control Protocol – TCP

Aufgrund der im folgendem erläuterten Eigenschaften findet das TCP in Quap Verwendung.

²⁰ Vgl. <https://docplayer.org/18936587-Referat-netzwerkprogrammierung-in-java.html>

²¹ Vgl. <https://oinf.ch/kurs/vernetzung-und-systeme/kommunikation-in-netzwerken/>

²² Vlg. <https://www.ip-insider.de/was-ist-das-osi-modell-a-605831/>

²³ Vlg. <https://www.tarife.at/wissen/netzwerkprotokoll>

²⁴ Vgl. <https://www.elektronik-kompendium.de/sites/net/2507261.htm>

Bei dem TCP handelt sich dabei um ein Netzwerkprotokoll, welches für eine verbindungsorientierte und datenverlustfreie Datenübertragung zuständig ist, womit es sowohl zum Verbindungsaufbau als auch für den Verbindungsabbau dient.

Das Transmission Control Protocol teilt den ausgehenden Datenstrom in einzelne Segmente auf und fügt eingehende Segmente wieder zusammen. Aufgrund der Tatsache, dass beim Übermitteln der Segmente (IP-Routing) die eingehende Reihenfolge nicht der ursprünglichen entsprechen kann, findet eine Sequenzierung der Segmente statt.²⁵

4.2.2. User Datagramm Protocol - UDP

UDP stellt eine Alternative zu TCP dar, da es sich dabei im Gegensatz zum TCP um ein verbindungsloses und ungesichertes Netzwerkprotokoll handelt.²⁶ Es baut auf dem IP auf und versendet Datenpakete, die auch Datagramme genannt werden.

Es eignet sich für Netzwerkanwendungen, wie Videoanrufe oder Onlinespiele, die eine besonders niedrige Latenz benötigen und mit geringen Datenverlusten zureckkommen. Da dies bei einem Messenger wie Quap nicht zutrifft, findet es im praktischen Teil keine Verwendung.

4.1.3. Internet Protocol - IP

Unter dem Internet Protocol, oder auch kurz IP, versteht man ein Netzwerkprotokoll, welches die Funktion und Grundlage des Internets, als auch die Implementierung der Internetschicht des TCP/IP Modells darstellt.

Innerhalb der TCP/IP Familie ist es auf der dritten Schicht des OSI-Referenzmodells für die Vermittlung und die Adressierung von Datenpaketen und die Übertragung in einem paketorientierten Netzwerk zuständig ist.²⁷ Damit ist es unerlässlich für die zielgerichtete Übertragung von Daten zwischen einzelnen Netzketten.

4.3. Adressierung

Im Messenger Quap ist es für einen Verbindungsaufbau notwendig, dass der Client die exakte und eindeutige IP-Adresse des Servers kennt, damit er seine Nachrichten dem

²⁵ Vgl. <https://www.elektronik-kompendium.de/sites/net/0812271.htm>

²⁶ Vgl. <https://www.tarife.at/wissen/udp-user-datatype-protocol>

²⁷ Vgl. <https://www.elektronik-kompendium.de/sites/net/0811271.htm>

richtigen Netzketten zukommen lässt.

Um diese Übertragung in einem paketorientierten Netzwerk zu ermöglichen, besitzt jeder Netzketten eine eigene IP-Adresse durch welcher er eindeutig identifiziert werden kann. Grundsätzlich wird dabei zwischen IPv4 und IPv6 Adressen unterschieden.²⁸

Die IPv4 Adressen setzen sich aus vier Zahlen zusammen.

Dabei kann jede Zahl in einem Bereich zwischen 0 und 255 liegen. Allerdings ist damit die Anzahl begrenzt, denn mit dieser Einschränkung können lediglich ca. 4,3 Milliarden verschiedene und gültige IPv4 Adressen existieren.

Um die Aufteilung eines lokalen Netzwerkes in mehrere, kleine Netzwerke zu ermöglichen, wird die Subnetzmaske verwendet. Sie besteht ebenfalls aus vier Zahlen, welche entweder 0 oder 255 sind. Über sie kann bestimmt werden, zu welchem Netzwerk eine IP-Adresse gehört. Für jede 255 der Subnetzmaske wird die entsprechende Stelle der IP der Netzwerk-ID zugeordnet und jede 0 der Host-ID.²⁹ So wären zum Beispiel bei einer IP von 192.168.178.74 und einer Subnetzmaske von 255.255.255.0 die ersten drei Zahlen der IP (192.168.178.) die Netzwerk-ID und die 74 würde der Host-ID zugeordnet werden.

5. Client-Server Modell

Am Beispiel des Messengers Quap lassen sich bespielhafte Client-Interaktionen simulieren, um die verschiedenen Prozesse, die innerhalb eines Lebenszyklus eines Clients stattfinden, zu demonstrieren. In Quap können mehrere Clients gleichzeitig eine Verbindung mit dem Server herstellen und in Echtzeit in Chatgruppen miteinander kommunizieren. Dies wird dadurch gewährleistet, dass jedem Client ein eigener Handler vom Server zugewiesen wird, welcher die Anfragen seines Clients bearbeitet.³⁰

5.1. Verbindungsauflaufbau

Der Verbindungsauflaufbau setzt voraus, dass der Server an seiner ServerSocket auf eingehend Verbindungsanfragen der Client Sockets lauscht.³¹ Daher ist die Voraussetzung für einen erfolgreichen Verbindungsauflaufbau, ein offener Socket des Servers, andernfalls würde jeder Verbindungsversuch des Clients einen Fehler verursachen.

Zunächst wird eine neue ServerSocket mit einer Kombination aus einer IPv4-Adresse und

²⁸ Vgl. <https://www.edv-lehrgang.de/ip-adressen-in-netzwerken/>

²⁹ Vgl. <https://www.edv-lehrgang.de/ip-adressen-in-netzwerken/>

³⁰ Anhang – II, Z.41-68

³¹ Vgl. https://www.informatik.uni-leipzig.de/~meiler/MuP.dir/MuPWS14.dir/Vorlesung/Kap16_Netz.pdf

einem TCP Port gebunden, woraufhin der Server in einem Thread auf eingehende Verbindungen wartet.³² Danach kann auf der Seite des Clients eine Socket mit den Konstruktorparametern, der Zieladresse und dem Zielport instanziert werden, wovon die jeweiligen Streams einem Writer und einem Reader zugewiesen werden.³³

Nun wird auf dem Server für jeden Client ein neuer Socket aufgebaut, welcher im Anschluss daran an einen ClientHandler übergeben wird, der von einem ExecutorService verwaltet wird. Jeder ClientHandler kümmert sich um die Interaktion mit seinem Client, indem er ebenfalls Writer und Reader initialisiert und ein Thread zum Lesen der einkommenden Nachrichten auswirft.³⁴

Aufgrund der Tatsache, dass auf einen Server mehrere Clients gleichzeitig zugreifen, müssen deren Anfragen mithilfe der ClientHandler nebenläufig abgearbeitet werden.

5.2. Datentransfer

Ein Datentransfer findet im Messenger Quap sowohl bei der Authentifizierung des Clients am Server, dem Austausch von Nachrichten, als auch beim Verbindungsabbau statt und ist somit allgegenwärtiger Prozess in einem Client-Server Modell.

Zur Interaktion zwischen Server und Client werden über den erzeugten Sockets die Streams aufgebaut. So hat sowohl der ClientHandler als auch der Client einen Thread, der die eingehenden Streams verarbeitet, ebenso wie einen Writer, über den Nachrichten zur jeweils anderen Socket gesendet werden können.³⁵ ³⁶

5.3. Verbindungsabbau

Der Verbindungsabbau geht in der Regel, so auch im Beispiel Quap, vom Client aus.

Sobald das Nutzerinterface geschlossen wird, teilt der Client den Verbindungsabbau dem ClientHandler mit, wartet nicht länger auf eingehende Nachrichten vom Server und die Socket des Clients wird über die close-Methode der Socket geschlossen.³⁷

Nachdem der ClientHandler den Verbindungsabbau von seinem Client mitgeteilt bekommt, wird die Verbindung des ClientHandlers zu der Datenbank geschlossen und das Warten auf

³² Anhang – II, Z.41-68

³³ Anhang – VII, Z.67-72

³⁴ Anhang – III, Z.40-58

³⁵ Anhang – II, Z.151-158

³⁶ Anhang – VII, Z.302-313

³⁷ Anhang – VII, Z.111-115

eingehende Nachrichten beendet, womit sich der ClientHandler selbständig schließt.³⁸

6. Schlussbemerkung

In dieser Facharbeit wurden Technologien wie Nebenläufigkeit und Datenbanken in Ansätzen betrachtet, als auch theoretische Aspekte der Netzwerktechnik in einem lokalen Netzwerk, wie Netzwerkarchitektur oder das Client-Server-Modell eingehend erläutert und deren Zusammenhang verdeutlicht.

Auch wurde deren Vorteile für unsere Kommunikation anhand von Beispielen festgestellt, wie beispielsweise die gemeinsame Nutzung von Daten, womit sich die Relevanz des Themas Netzwerktechnik als Grundlage für unsere unerlässliche digitale Kommunikation begründen lässt.

Anhand der Demonstration der Netzwerkanwendung Quap konnte die Kommunikation in einem Client-Server-Modell nachvollziehbar dargestellt werden, so dass sich abschließend die Komplexität und Vielseitigkeit von Netzwerktechnologie bestätigt.

Auch wurde die Sinnhaftigkeit einer Nutzung von Technologien wie Nebenläufigkeit oder Datenbanken im praktischen Teil und die Eignung der Sprache Java für eine solche Netzwerkanwendung deutlich. Unter dem Aspekt der Kommunikation zwischen mehreren Clients zu einem Server in einem lokalen Netzwerk wurde der Nutzen des TCPs dargestellt.

6.1. Ausblick

Um ihm Rahmen der Vorgaben dieser Facharbeit zu bleiben, wurden einige Technologien der Netzwerktechnik als auch weiterführende Features im praktischen Teil nur kurz angeschnitten oder vollständig weggelassen, da sie keine Relevanz für die Aufgabenstellung dieser Arbeit haben.

Dazu zählen beispielsweise Technologien wie DNS, unterschiedliche Netzwerkdimensionen, Sicherheitskonzepte der Netzwerktechnik oder Cloud Computing. Obwohl der praktische Teil ein abgeschlossenes Projekt darstellt, gäbe es auch hier Möglichkeiten, den Umfang durch den Aspekt der Datensicherheit in Form von kryptographischer Verschlüsselung, Nutzerprofilen und App-Einstellungen oder einer dezentralen Datensynchronisation zwischen Geräten des gleichen Nutzeraccounts zusätzlich zu erweitern.

³⁸ Anhang – III, Z.318-329

7. Literaturverzeichnis

<https://git-scm.com> (26.02.2022)

<https://docs.github.com/en/get-started/quickstart/hello-world> (26.02.2022)

https://www.thomas-krenn.com/de/wiki/Git_Grundbegriffe (26.02.2022)

<https://www.storage-insider.de/was-ist-multithreading-a-1017586/> (26.02.2022)

http://www4.cs.fau.de/Lehre/WS13/V_SP2/Vorlesung/Folien/SP2-101-A6.pdf
(26.02.2022)

<https://wwwlehre.dhbw-stuttgart.de/~kfg/java/ccjava.pdf> (26.02.2022)

<https://www.elektronik-kompendium.de/sites/net/0503271.htm> (26.02.2022)

<https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html> (26.02.2022)

https://lehrerfortbildung-bw.de/u_matnatech/informatik/gym/bp2016/fb1/3_rechner_netze/1_hintergrund/3_kommunikation/1_netzwerk/ (26.02.2022)

<https://oinf.ch/kurs/vernetzung-und-systeme/kommunikation-in-netzwerken/> (26.02.2022)

<https://www.ip-insider.de/was-ist-das-osi-modell-a-605831/> (26.02.2022)

<https://www.tarife.at/wissen/netzwerkprotokoll> (26.02.2022)

<https://www.elektronik-kompendium.de/sites/net/0812271.htm> (26.02.2022)

<https://www.tarife.at/wissen/udp-user-datatype-protocol> (26.02.2022)

<https://www.edv-lehrgang.de/ip-adressen-in-netzwerken/> (26.02.2022)

<https://docplayer.org/18936587-Referat-netzwerkprogrammierung-in-java.html>
(26.02.2022)

https://www.informatik.uni-leipzig.de/~meiler/MuP.dir/MuPWS14.dir/Vorlesung/Kap16_Netz.pdf (26.02.2022)

Anhang:

I – Main.java

```
01 package com.quap;
02
03 import com.quap.server.Server;
04
05 import java.io.IOException;
06 import java.net.InetSocketAddress;
07 import java.net.ServerSocket;
08
09 /**
10  * Die Klasse stellt die Mainklasse des Servers dar und enthält die main-Methode
11  */
12 public class Main {
13     private static ServerSocket socket;
14     private static int port;
15
16     /**
17      * Die Methode verarbeitet die Argumente, mit denen die Klasse aufgerufen wurde.
18      * Wenn die Socket sich zu der ihr zugewiesenen Adresse binden kann, wird die Serverklasse aufgerufen
19      * @param args entweder ohne Argument, oder ein gültiger TCP Port (Standart ist 8192)
20      */
21     public static void main(String[] args) {
22         System.setProperty("java.net.preferIPv4Stack", "true");
23
24         if (args.length==0) {
25             Main.port = 8192;
26         } else if(args.length==1) {
27             Main.port = Integer.parseInt(args[0]);
28         } else {
29             System.err.println("Usage1: java -jar QuapServer.jar");
30             System.err.println("Usage2: java -jar QuapServer.jar <PORT[int]>");
31         }
32         try {
33             socket = new ServerSocket();
34             socket.bind(new InetSocketAddress("192.168.178.43", Main.port));
35         } catch (IOException e) {
```

```

36             e.printStackTrace();
37         }
38         new Server(socket);
39     }
40 }
```

II – Server.java

```

001 package com.quap.server;
002
003 import java.io.IOException;
004 import java.net.ServerSocket;
005 import java.net.Socket;
006 import java.time.LocalTime;
007 import java.util.ArrayList;
008 import java.util.List;
009 import java.util.concurrent.ExecutorService;
010 import java.util.concurrent.Executors;
011 import java.util.concurrent.TimeUnit;
012
013 /**
014 * Wartet auf eingehende Verbindungen an der ServerSocket
015 * und verwaltet die ClientHandler in einem ExecutorService
016 */
017 public class Server{
018     private final ExecutorService service;
019     private final ServerSocket socket;
020     private Thread receive;
021
022     private final List<ClientHandler> handler = new ArrayList<>();
023     private boolean status;
024
025
026 /**
027 * Im Konstruktor wird die Socket und der Status gesetzt,
028 * als auch die receiveConnection() Methode gestartet
029 * @param socket die ServerSocket
030 */
031 public Server(ServerSocket socket) {
032     service = Executors.newCachedThreadPool();
```

```

033         this.socket = socket;
034         status = true;
035         receiveConnection();
036     }
037
038     /**
039      * Die Methode wirft einen Thread aus, indem auf eingehende Verbindungsanfragen gewartet wird
040      */
041     public void receiveConnection() {
042         Server server = this;
043         receive = new Thread("Receive") {
044             public void run() {
045                 while (status) {
046                     Socket client = null;
047                     try {
048                         System.out.println("Listening on: " + socket.getLocalSocketAddress());
049                         client = socket.accept();
050                     } catch (IOException e) {
051                         e.printStackTrace();
052                         terminate(false);
053                     }
054                     assert client != null;
055                     System.out.print("\r\nNew connection from " + client.getInetAddress() + ":" +
client.getPort());
056                     System.out.println(" to " + socket.getInetAddress() + ":" + socket.getLocalPort());
057                     ClientHandler clientHandler = new ClientHandler(client, UniqueIdentifier.getIdentifier(),
server);
058                     service.submit(clientHandler);
059                     handler.add(clientHandler);
060                     if(socket.isClosed()) {
061                         status = false;
062                         terminate(true);
063                     }
064                 }
065             }
066         };
067         receive.start();
068     }
069
070     /**
071      * Bringt den ClientHandler mit der entsprechenden Id, dazu die Verbindung zum Client zu trennen

```

```

072     * @param id Die UUID des ClientHandlers
073     */
074     private void disconnect(int id) {
075         ClientHandler c;
076         for (int i = 0; i < handler.size(); i++) {
077             if (handler.get(i).getID() == id) {
078                 c = handler.get(i);
079                 c.disconnect(true);
080                 handler.remove(i);
081                 break;
082             }
083         }
084     }
085
086 /**
087 * Bricht alle Verbindungen ab und schließt den Server
088 * @param status Der status, mit dem der Server sich schließen soll
089 */
090 public void terminate(boolean status) {
091     receive.interrupt();
092     for (ClientHandler clientHandler : handler) {
093         disconnect(clientHandler.getID());
094     }
095     try {
096         socket.close();
097     } catch (IOException e) {
098         e.printStackTrace();
099     }
100    if (status) {
101        System.out.println(LocalTime.now().toString().substring(0, LocalTime.now().toString().indexOf(".")))
102            + " :Attempting to shutdown";
103        service.shutdown();
104        try {
105            if (service.awaitTermination(4, TimeUnit.SECONDS)) {
106                System.out.println(LocalTime.now().toString().substring(0,
LocalTime.now().toString().indexOf(".")))
107                    + " :Pool terminate";
108                System.out.println(Thread.activeCount() + ", " + Thread.getAllStackTraces() + ", "
109                    + Thread.getAllStackTraces().hashCode() + ", " + Thread.currentThread().getState());
110                service.shutdownNow();
111            } else if (!service.awaitTermination(4, TimeUnit.SECONDS)) {

```

```

112         System.err.println("Pool did not terminate");
113         System.out.println(Thread.activeCount() + ", " + Thread.getAllStackTraces() + ", "
114                         + Thread.getAllStackTraces().hashCode() + ", " + Thread.currentThread().getState());
115         service.shutdownNow();
116     }
117 } catch (InterruptedException ie) {
118     Thread.currentThread().interrupt();
119 }
120 } else {
121     Runtime.getRuntime().addShutdownHook(new Thread("shutdown") {
122         public void run() {
123             if (!service.isShutdown() || !service.isTerminated()) {
124                 service.shutdown();
125                 try {
126                     if (service.awaitTermination(4, TimeUnit.SECONDS)) {
127                         System.out.println("Main: Pool terminate");
128                         service.shutdownNow();
129                     } else if (!service.awaitTermination(4, TimeUnit.SECONDS)) {
130                         System.err.println("Pool did not terminate");
131                         service.shutdownNow();
132                         System.out.println(Thread.activeCount() + ", " + Thread.getAllStackTraces() + ", "
133                                         + Thread.getAllStackTraces().hashCode() + ", " +
134                                         Thread.currentThread().getState());
135                     } catch (InterruptedException ie) {
136                         Thread.currentThread().interrupt();
137                     }
138                 }
139             }
140         }
141     });
142 }
143 }
144 /**
145 * Sendet über den ClientHandler, welcher den Nutzer
146 * mit der entsprechenden userID verwaltet die Nachricht
147 * @param userID die Id des Nutzers
148 * @param message Die zu sendene Nachricht
149 */
150
151 public void forwardMessage(int userID, String message) {

```

```

152         for (ClientHandler clientHandler : handler) {
153             if (clientHandler.getUserID() == userID) {
154                 clientHandler.send(message);
155             }
156         }
157     }
158 }
```

III – ClientHandler.java

```

001 package com.quap.server;
002
003 import com.quap.data.UserdataReader;
004 import org.json.JSONArray;
005 import org.json.JSONException;
006 import org.json.JSONObject;
007
008 import java.io.*;
009 import java.net.Socket;
010 import java.net.SocketException;
011 import java.sql.SQLException;
012 import java.time.LocalTime;
013 import java.util.ArrayList;
014 import java.util.List;
015
016 /**
017 * Die Klasse ist für die eingehenden Anfragen des übergebenen Clients zuständig.
018 * Die ClientHandler laufen Nebenläufig voneinander ab, werden im Server verwaltet und verwalten jeweils
019 * ein Object der Klasse UserdataReader.
020 */
021 public class ClientHandler implements Runnable {
022     private final int ID;
023     private int userID;
024     private final Socket socket;
025     private final Server server;
026     private Thread listen;
027
028     private InputStream input;
029     private final BufferedReader reader;
030     private OutputStream output;
```

```

031     private final PrintWriter writer;
032     private String name;
033
034     /**
035      * Im Konstruktor werden die Streams der Socket Writer und Reader zugewiesen.
036      * @param socket Die Socket, welche bei einer eingehenden Verbindung erzeugt wurde
037      * @param ID eine UUID, generiert bei der UniqueIdentifier Klasse
038      * @param server eine Referenz auf den Server
039     */
040     public ClientHandler(Socket socket, int ID, Server server) {
041         this.socket = socket;
042         this.ID = ID;
043         this.server = server;
044         userID = -1;
045
046         try {
047             input = socket.getInputStream();
048         } catch (IOException e) {
049             e.printStackTrace();
050         }
051         try {
052             output = socket.getOutputStream();
053         } catch (IOException e) {
054             e.printStackTrace();
055         }
056         writer = new PrintWriter(output, true);
057         reader = new BufferedReader(new InputStreamReader(input));
058     }
059
060     /**
061      * Die Methode wirft einen Thread aus, welcher den eingehenden Stream liest
062     */
063     private void listen() {
064         listen = new Thread(() -> {
065             String message;
066             while (!socket.isClosed() && reader != null) {
067                 try {
068                     message = reader.readLine();
069                     if (message != null) {
070                         process(message);
071                     }
072                 }
073             }
074         });
075     }

```

```

072         } catch (SocketException e) {
073             e.printStackTrace();
074             this.disconnect(false);
075             try {
076                 reader.close();
077             } catch (IOException ex) {
078                 ex.printStackTrace();
079             }
080         } catch (IOException e) {
081             e.printStackTrace();
082             this.disconnect(false);
083         }
084     }
085 });
086 listen.start();
087 }
088
089 /**
090 * Diese Methode differenziert die einkommenden Daten anhand ihrer Metadaten
091 * und führt für verschiedene Werte unterschiedliche Aktionen aus.
092 * @param message den Input des eingehenden Streams
093 */
094 private void process(String message) {
095     String content = message.substring(5, (message.length() - 5));
096     System.out.println(message);
097     System.out.println(content);
098     switch (message.charAt(2)) {
099         case 'm' -> {
100             System.out.println("message found:" + content);
101             UserdataReader dbReader = null;
102             try {
103                 dbReader = new UserdataReader();
104             } catch (SQLException e) {
105                 e.printStackTrace();
106             }
107             JSONObject input = new JSONObject(content).getJSONObject("data");
108             int chatID = input.getInt("chat_id");
109             assert dbReader != null;
110             List<Integer> userIDs = new ArrayList<>(dbReader.userIDsByChat(chatID));
111             for (Integer id : userIDs) {
112                 server.forwardMessage(id, content);

```

```

113         }
114     }
115     case 'c' -> {
116         System.out.println("command found");
117         JSONObject data = new JSONObject(content).getJSONObject("data");
118         int senderID = data.getInt("sender_id");
119         UserdataReader dbReader = null;
120         try {
121             dbReader = new UserdataReader();
122         } catch (SQLException e) {
123             e.printStackTrace();
124         }
125         switch (new JSONObject(content).getString("type")) {
126             case "create-chat" -> {
127                 String chatName = data.getString("chatname");
128                 assert dbReader != null;
129                 JSONObject result = dbReader.addChat(senderID, chatName, false);
130                 JSONObject json = new JSONObject();
131                 json.put("return-value", "command");
132                 if (result != null) {
133                     JSONObject returnValue = new JSONObject();
134                     returnValue.put("statement", "create-chat");
135                     returnValue.put("result", result);
136                     json.put("data", returnValue);
137                 } else {
138                     json.put("error", "can not create this chat");
139                 }
140                 send(json.toString());
141             }
142             case "invite-user" -> {
143                 System.out.println("Invite user to chat...");
144                 try {
145                     dbReader = new UserdataReader();
146                 } catch (SQLException e) {
147                     e.printStackTrace();
148                 }
149                 String username = data.getString("username");
150                 String senderName = name;
151                 assert dbReader != null;
152                 int userID = dbReader.userIDByName(username);
153                 int chatID = data.getInt("chat_id");

```

```

154         boolean isParticipant = false;
155         for(int participant : dbReader.userIDsByChat(chatID)) {
156             if (userID == participant) {
157                 isParticipant = true;
158                 break;
159             }
160         }
161         if(!isParticipant) {
162             JSONObject chat = dbReader.getChatByID(chatID);
163             JSONArray participants = dbReader.usersByChat(chatID);
164             JSONObject json = new JSONObject();
165             json.put("return-value", "command");
166             if (chat != null) {
167                 JSONObject returnValue = new JSONObject();
168                 returnValue.put("statement", "invite-chat");
169                 returnValue.put("chat", chat);
170                 returnValue.put("sender_id", senderID);
171                 returnValue.put("sender_name", senderName);
172                 returnValue.put("participants", participants);
173                 json.put("data", returnValue);
174             } else {
175                 json.put("error", "can not create this chat");
176             }
177             server.forwardMessage(userID, json.toString());
178         }
179     }
180     case "join-chat" -> {
181         int chatID = data.getInt("chatroom_id");
182         try {
183             dbReader = new UserdataReader();
184         } catch (SQLException e) {
185             e.printStackTrace();
186         }
187         assert dbReader != null;
188         dbReader.addUserToChat(chatID, senderID);
189         JSONObject chat = dbReader.getChatByID(chatID);
190         JSONObject json = new JSONObject();
191         json.put("return-value", "command");
192         if (chat != null) {
193             JSONObject returnValue = new JSONObject();
194             returnValue.put("statement", "join-chat");

```

```

195                     returnValue.put("chat", chat);
196                     json.put("data", returnValue);
197                 } else {
198                     json.put("error", "can not join this chat");
199                 }
200             send(json.toString());
201             json = new JSONObject();
202             json.put("return-value", "message");
203             JSONObject returnValue = new JSONObject();
204             returnValue.put("message", "User " + dbReader.userNameById(senderID) + " joined the
chatroom.");
205             returnValue.put("chat_id", chatID);
206             returnValue.put("sender_id", 0);
207             returnValue.put("sender_name", "Server");
208             json.put("data", returnValue);
209             List<Integer> userIDs = new ArrayList<>(dbReader.userIDsByChat(chatID));
210             for (Integer id : userIDs) {
211                 server.forwardMessage(id, json.toString());
212             }
213         }
214     case "delete-chat" -> {
215         int chatID = data.getInt("chat_id");
216         assert dbReader != null;
217         dbReader.deleteUserFromChat(senderID, chatID);
218         JSONObject json = new JSONObject();
219         json.put("return-value", "message");
220         JSONObject returnValue = new JSONObject();
221         returnValue.put("message", "User " + dbReader.userNameById(senderID) + " left the
chatroom.");
222         returnValue.put("chat_id", chatID);
223         returnValue.put("sender_id", 0); //0 == server
224         returnValue.put("sender_name", "Server");
225         json.put("data", returnValue);
226         List<Integer> userIDs = new ArrayList<>(dbReader.userIDsByChat(chatID));
227         for (Integer id : userIDs) {
228             server.forwardMessage(id, json.toString());
229         }
230     }
231     case "request-user" -> {
232         String username = data.getString("username");
233         assert dbReader != null;

```

```

234         int userID = dbReader.userIDByName(username);
235         String senderName = name;
236         JSONObject json = new JSONObject();
237         json.put("return-value", "command");
238         JSONObject returnValue = new JSONObject();
239         returnValue.put("statement", "friend-request");
240         returnValue.put("sender_id", senderID);
241         returnValue.put("sender_name", senderName);
242         json.put("data", returnValue);
243         server.forwardMessage(userID, json.toString());
244     }
245     case "accept-friend" -> {
246         int friendID = data.getInt("friend_id");
247         assert dbReader != null;
248         int chatID = dbReader.insertFriends(senderID, friendID);
249         JSONObject json = new JSONObject();
250         json.put("return-value", "command");
251         JSONObject returnValue = new JSONObject();
252         returnValue.put("statement", "add-friend");
253         JSONObject friend = new JSONObject();
254         friend.put("name", dbReader.userNameById(friendID));
255         friend.put("user_id", friendID);
256         friend.put("created_at", LocalTime.now().toString());
257         friend.put("chatrooms_id", chatID);
258         returnValue.put("friend", friend);
259         json.put("data", returnValue);
260         server.forwardMessage(senderID, json.toString());
261
262         friend = new JSONObject();
263         returnValue.remove("friend");
264         friend.put("name", dbReader.userNameById(senderID));
265         friend.put("user_id", userID);
266         friend.put("created_at", LocalTime.now().toString());
267         friend.put("chatrooms_id", chatID);
268         returnValue.put("friend", friend);
269         json.put("data", returnValue);
270         server.forwardMessage(friendID, json.toString());
271     }
272     case "unfriend-user" -> {
273         int friendID = data.getInt("friend_id");
274         int chatID = data.getInt("chat_id");

```

```

275             assert dbReader != null;
276             dbReader.unfriendUsers(chatID);
277             JSONObject json = new JSONObject();
278             json.put("return-value", "command");
279             JSONObject returnValue = new JSONObject();
280             returnValue.put("statement", "delete-friend");
281             returnValue.put("friend_id", senderID);
282             returnValue.put("chat_id", chatID);
283             json.put("data", returnValue);
284             server.forwardMessage(friendID, json.toString());
285         }
286     }
287 }
288 case 'a' -> {
289     System.out.println("authentication found");
290     UserdataReader dbReader = null;
291     try {
292         dbReader = new UserdataReader();
293     } catch (SQLException e) {
294         e.printStackTrace();
295     }
296     assert dbReader != null;
297     JSONObject json = new JSONObject(content);
298     String name = json.getString("name");
299     String password = json.getString("password");
300     boolean existing = json.getBoolean("existing");
301     JSONObject result;
302     if (existing) {
303         result = dbReader.verifyUser(name, password);
304     } else {
305         result = dbReader.insertUser(name, password);
306     }
307     this.name = name;
308     try {
309         userID = result.getJSONObject("data").getInt("id");
310     } catch (JSONException e) {
311         System.err.println("The user " + name + " was not found.");
312         result = new JSONObject();
313         result.put("error", "The user " + name + " was not found.");
314         result.put("return-value", "authentication");
315     }

```

```

316             send(result.toString());
317         }
318     case 'd' -> {
319         UserdataReader dbReader = null;
320         try {
321             dbReader = new UserdataReader();
322         } catch (SQLException e) {
323             e.printStackTrace();
324         }
325         assert dbReader != null;
326         listen.interrupt();
327         dbReader.disconnect();
328     }
329 }
330 }
331 /**
332 * Die Methode sendet beim aufruf, über den Writer, die Nachricht zum Client
333 * @param message die Nachricht im JSON-Format, mit allen relevanten Metadaten
334 */
335 public void send(String message) {
336     System.out.println("Server Message to Client: " + message);
337     writer.println(message);
338 }
339 }
340
341 public int getID() {
342     return ID;
343 }
344
345 public int getUserId() {
346     return userID;
347 }
348 /**
349 * Teilt beim Methodenaufruf den Verbindungsabbruch dem Client mit
350 * @param status false bei Fehler
351 */
352 public void disconnect(boolean status) {
353     JSONObject json = new JSONObject();
354     json.put("return-value", "disconnect");
355     JSONObject data = new JSONObject();

```

```

357         data.put("status", status);
358         json.put("data", data);
359         send(json.toString());
360     }
361
362     /**
363      * Startet die listen-Methode und wird durch das Runnable Interface vorgeschrieben
364      */
365     @Override
366     public void run() {
367         listen();
368     }
369 }
```

IV – Launcher.java

```

01 package com.quap.desktopapp;
02
03
04 import com.quap.controller.scene.ConnectionWindowController;
05 import javafx.application.Application;
06 import javafx.stage.Stage;
07
08 /**
09  * Die Klasse stellt die Main Klasse des Clients dar, welche die main-Methode enthält
10  *
11  */
12 public class Launcher extends Application {
13
14     private static String adress = null;
15     /**
16      * Die Methode sorgt für die Initialisierung im Verbindungsfenster
17      */
18     @Override
19     public void init() {
20         ConnectionWindowController init = new ConnectionWindowController();
21         init.connect(adress);
22         init.launchWindow();
23     }
24 }
```

```

25     @Override
26     public void start(Stage primaryStage) {} 
27
28     /**
29      * Beim Methodenaufruf, wird der Preloader des Clients geladen
30      * @param args Argumente der Client Anwendung (Keine Argumente erwartet)
31      */
32     public static void main(String[] args) {
33         System.setProperty("javafx.preloader", LauncherPreloader.class.getCanonicalName());
34         if(args.length == 1) {
35             adress = args[0];
36         }
37         launch(args);
38     }
39 }
```

V – ConnectionWindowController.java

```

001 package com.quap.controller.scene;
002
003 import com.quap.client.Client;
004 import com.quap.controller.VistaController;
005 import com.quap.desktopapp.LauncherPreloader;
006 import com.quap.utils.WindowMoveHelper;
007 import javafx.application.Platform;
008 import javafx.fxml.FXML;
009 import javafx.fxml.FXMLLoader;
010 import javafx.fxml.Initializable;
011 import javafx.scene.Parent;
012 import javafx.scene.Scene;
013 import javafx.scene.control.Label;
014 import javafx.scene.control.ProgressBar;
015 import javafx.stage.Stage;
016 import javafx.stage.StageStyle;
017
018 import java.io.IOException;
019 import java.net.URL;
020 import java.util.ResourceBundle;
021
022 public class ConnectionWindowController implements Initializable {
```

```

023     private Client client;
024
025     @FXML
026     private Label lblLoading;
027
028     @FXML
029     private ProgressBar progressBar;
030
031     private static Label loadingLabel;
032     private static ProgressBar prBar;
033     double progress;
034
035
036     @Override
037     public void initialize(URL url, ResourceBundle resourceBundle) {
038         loadingLabel = lblLoading;
039         prBar = progressBar;
040     }
041
042     public void connect(String adress) {
043         increaseProgress();
044         Platform.runLater(() -> loadingLabel.setText("Open Connection"));
045         try {
046             if (adress != null) {
047                 client = new Client(adress, 8192);
048             } else {
049                 client = new Client("192.168.178.43", 8192);
050             }
051         } catch (IOException e) {
052             e.printStackTrace();
053         }
054     }
055
056     public void increaseProgress() {
057         if (progress < 1) {
058             for (int i = 0; i < 50; i++) {
059                 progress += 0.01;
060                 prBar.setProgress(progress);
061                 //System.out.println(Double.toString(Math.round(progress * 100)) + "%");
062                 try {
063                     Thread.sleep(10);

```

```

064             } catch (InterruptedException e) {
065                 e.printStackTrace();
066             }
067         }
068     }
069 }
070
071 public void launchWindow() {
072     increaseProgress();
073     Platform.runLater(() -> loadingLabel.setText("Loading Login"));
074     Platform.runLater(new Runnable() {
075         @Override
076         public void run() {
077             Stage stage = new Stage();
078             Parent root = null;
079             FXMLLoader loader = new FXMLLoader(getClass().getResource(VistaController.LOGIN));
080             try {
081                 root = loader.load();
082             } catch (IOException e) {
083                 e.printStackTrace();
084             }
085             Scene scene;
086             final String osName = System.getProperty("os.name");
087             if (osName != null && osName.startsWith("Windows")) {
088                 scene = (new LauncherPreloader.ShadowScene()).getShadowScene(root);
089                 stage.initStyle(StageStyle.TRANSPARENT);
090             } else {
091                 assert root != null;
092                 scene = new Scene(root);
093                 stage.initStyle(StageStyle.UNDECORATED);
094             }
095             stage.setScene(scene);
096             LoginWindowController loginWindowController = loader.getController();
097             loginWindowController.setClient(client);
098             VistaController.loadVista(VistaController.SignIn, loginWindowController);
099             stage.show();
100             WindowMoveHelper.addMoveListener(stage);
101         }
102     });
103 }
104 }
```

VI – MainWindowController.java

```
001 package com.quap.controller.scene;
002
003 import com.quap.client.Client;
004 import com.quap.client.domain.Chat;
005 import com.quap.client.domain.Friend;
006 import com.quap.client.domain.Message;
007 import com.quap.client.domain.UserContent;
008 import com.quap.client.utils.MainClientObserver;
009 import com.quap.controller.SceneController;
010 import com.quap.controller.VistaController;
011 import com.quap.controller.vista.MainVistaObserver;
012 import com.quap.controller.vista.VistaNavigator;
013 import com.quap.controller.vista.main.ChatController;
014 import com.quap.controller.vista.main.MainVistaNavigator;
015 import javafx.application.Platform;
016 import javafx.event.ActionEvent;
017 import javafx.fxml.FXML;
018 import javafx.fxml.FXMLLoader;
019 import javafx.geometry.Rectangle2D;
020 import javafx.scene.Node;
021 import javafx.scene.Parent;
022 import javafx.scene.control.Button;
023 import javafx.scene.control.Label;
024 import javafx.scene.control.ToggleButton;
025 import javafx.scene.control.ToggleGroup;
026 import javafx.scene.layout.StackPane;
027 import javafx.scene.layout.VBox;
028 import javafx.stage.Screen;
029 import javafx.stage.Stage;
030 import javafx.stage.Window;
031
032 import java.io.IOException;
033 import java.util.ArrayList;
034 import java.util.Collections;
035 import java.util.List;
036 import java.util.Scanner;
```

```

038     import static com.quap.controller.VistaController.CHAT;
039
040     /**
041      * Diese Klasse verwaltet die Benutzerinteraktion im Hauptfenster.
042      * Dabei verwaltet diese Klasse die Klasse Client.
043      * Über Events können Updates im Client an das Nutzerinterface übergeben werden
044      */
045     public class MainWindowController extends WindowController implements MainClientObserver, MainVistaObserver {
046
047         private MainVistaNavigator currentNode;
048         private double lastX = 0.0d, lastY = 0.0d, lastWidth = 0.0d, lastHeight = 0.0d;
049         private ToggleGroup submenuGroup;
050         public static Client client;
051         public String currentNodeID;
052
053
054         @FXML
055         private VBox vBoxButtonHolder;
056         @FXML
057         private Label lblName;
058         @FXML
059         private ToggleButton btnFriends;
060         @FXML
061         private ToggleButton btnChatrooms;
062         @FXML
063         private ToggleButton btnProfil;
064         @FXML
065         private ToggleButton btnSettings;
066         @FXML
067         private StackPane stackContent;
068         @FXML
069         private Label lblServer_IP;
070
071         @FXML
072         public void initialize() {
073             Parent node;
074             FXMLLoader loader;
075             try {
076                 loader = new FXMLLoader(VistaController.class.getResource(VistaController.LIST));
077                 node = loader.load();
078                 setVista(node, loader.getController());

```

```

079         } catch (IOException e) {
080             e.printStackTrace();
081         }
082     ToggleGroup menuGroup = new ToggleGroup();
083     submenuGroup = new ToggleGroup();
084     btnFriends.setToggleGroup(menuGroup);
085     btnChatrooms.setToggleGroup(menuGroup);
086     btnProfil.setToggleGroup(menuGroup);
087     btnSettings.setToggleGroup(menuGroup);
088
089     menuGroup.selectedToggleProperty().addListener(observable, oldValue, newValue) -> {
090         if (newValue != null) {
091             newValue.setSelected(true);
092             if (oldValue != null) {
093                 oldValue.setSelected(false);
094             }
095         }
096     );
097 }
098
099     public void setVista(Parent node, VistaNavigator controller) { //set the current node is called by
VistaController
100         if (node.getId().equals("chat") || node.getId().equals("list")) {
101             if (currentNode != null) {
102                 currentNode.removeObserver(this);
103             }
104             currentNodeID = node.getId();
105             currentNode = (MainVistaNavigator) controller;
106             currentNode.setClient(client);
107             currentNode.addObserver(this);
108             currentNode.setType("unknown");
109         } else {
110             throw new IllegalArgumentException();
111         }
112         stackContent.getChildren().setAll(node);
113     }
114
115     @FXML
116     public void maximize(ActionEvent actionEvent) {
117         Node n = (Node) actionEvent.getSource();
118         Window w = n.getScene().getWindow();

```

```

119         double currentX = w.getX(), currentY = w.getY(), currentWidth = w.getWidth(), currentHeight =
w.getHeight();
120         Screen screen = Screen.getPrimary();
121         Rectangle2D bounds = screen.getVisualBounds();
122         if (currentX != bounds.getMinX() &&
123             currentY != bounds.getMinY() &&
124             currentWidth != bounds.getWidth() &&
125             currentHeight != bounds.getHeight()) {
126             w.setX(bounds.getMinX());
127             w.setY(bounds.getMinY());
128             w.setWidth(bounds.getWidth());
129             w.setHeight(bounds.getHeight());
130             lastX = currentX; // save old dimensions
131             lastY = currentY;
132             lastWidth = currentWidth;
133             lastHeight = currentHeight;
134         } else { // de-maximize the window (not same as minimize)
135             w.setX(lastX);
136             w.setY(lastY);
137             w.setWidth(lastWidth);
138             w.setHeight(lastHeight);
139         }
140         actionEvent.consume(); // don't bubble up to title bar
141     }
142
143     @FXML
144     public void minimize(ActionEvent actionEvent) {
145         Stage stage = (Stage) ((Button) actionEvent.getSource()).getScene().getWindow();
146         stage.setIconified(true);
147     }
148
149     @FXML
150     public void close(ActionEvent actionEvent) {
151         client.removeMainObserver(this);
152         client.disconnect(true);
153         ((Stage) (((Button) actionEvent.getSource()).getScene().getWindow())).close();
154     }
155
156     @FXML
157     public void friends() {
158         VistaController.loadVista(VistaController.LIST, this);

```

```

159         currentNode.loadContent(new ArrayList<>(client.getFriends()));
160         currentNode.setType("friends");
161         loadButtons(client.getFriends());
162     }
163
164     public void loadButtons(List<UserContent> data) {
165         vBoxButtonHolder.getChildren().clear();
166         submenuGroup.getToggles().clear();
167         for (UserContent content : data) {
168             ToggleButton b = new ToggleButton(content.content());
169             b.setOnAction(e -> {
170                 VistaController.loadVista(CHAT, this);
171                 currentNode.loadContent(
172                     client.getMessagesByChat(content.chatID())
173                 );
174                 client.setCurrentChatID(content.chatID());
175             });
176             b.setToggleGroup(submenuGroup);
177             b.setMinWidth(90);
178             b.setMaxWidth(90);
179             vBoxButtonHolder.getChildren().add(b);
180         }
181     }
182
183     public void setName(String name) {
184         lblName.setText(name);
185     }
186
187     @FXML
188     public void chatrooms() {
189         VistaController.loadVista(VistaController.LIST, this);
190         currentNode.loadContent(new ArrayList<>(client.getChats()));
191         currentNode.setType("chatrooms");
192         loadButtons(client.getChats());
193     }
194
195     public void setClient(Client client) {
196         MainWindowController.client = client;
197         MainWindowController.client.addMainObserver(this);
198         lblServer_IP.setText(client.getConnectionInfo());
199         friends();

```

```

200     }
201
202     @Override
203     public void messageEvent(Message message) {
204         System.out.println("messageEvent");
205         if (currentNodeID.equals("chat")) {
206             ((ChatController) currentNode).addMessage(message);
207         }
208     }
209
210     @Override
211     public void createChatEvent(Chat chat) {
212         System.out.println("createChatEvent");
213         if (currentNode.getType().equals("chatrooms")) {
214             Platform.runLater(() -> loadButtons(client.getChats()));
215             if (currentNodeID.equals("list")) {
216                 Platform.runLater(() -> currentNode.loadContent(Collections.singletonList(chat)));
217             }
218         }
219     }
220
221     @Override
222     public void inviteEvent(Chat chat, int senderID, String senderName, List<String> participants) {
223         FXMLLoader loader = new
FXMLLoader(getClass().getResource("/com/quap/desktopapp/popup/requestPopup.fxml"));
224         Stage primaryStage = (Stage)
Stage.getWindows().stream().filter(Window::isShowing).findFirst().orElse(null);
225         List<String> info = new ArrayList<>();
226         info.add("Invited by: " + senderName + "#" + senderID);
227         Scanner scanner = new Scanner(chat.display());
228         while (scanner.hasNextLine()) {
229             String line = scanner.nextLine();
230             info.add(line);
231         }
232         StringBuilder sb = new StringBuilder();
233         sb.append("participants: " + "\n");
234         for (String participant : participants) {
235             sb.append(participant).append("\n");
236         }
237         info.add(sb.toString());
238         scanner.close();

```

```

239         Platform.runLater(() -> {
240             boolean decision;
241             decision = SceneController.submitRequestPopup(loader, primaryStage, info, "Your are invited to a
chatroom");
242             if (decision) {
243                 System.out.println("send join-chat-request to the server with chat and sender_id...");
244                 client.joinChat(chat.chatID());
245             }
246         });
247     }
248
249     @Override
250     public void joinChatEvent(Chat chat) {
251         System.out.println("joinChatEvent");
252         if (currentNode.getType().equals("chatrooms")) {
253             Platform.runLater(() -> loadButtons(client.getChats()));
254             if (currentNodeID.equals("list")) {
255                 Platform.runLater(() -> currentNode.loadContent(Collections.singletonList(chat)));
256             }
257         }
258     }
259
260     @Override
261     public void friendRequestEvent(Friend friend) {
262         FXMLLoader loader = new
FXMLLoader(getClass().getResource("/com/quap/desktopapp/popup/requestPopup.fxml"));
263         Stage primaryStage = (Stage)
Stage.getWindows().stream().filter(Window::isShowing).findFirst().orElse(null);
264         List<String> info = new ArrayList<>();
265         info.add("Invited by: " + friend.name() + "#" + friend.id());
266         Scanner scanner = new Scanner(friend.display());
267         while (scanner.hasNextLine()) {
268             String line = scanner.nextLine();
269             info.add(line);
270         }
271         scanner.close();
272         Platform.runLater(() -> {
273             boolean decision;
274             decision = SceneController.submitRequestPopup(loader, primaryStage, info, "You have a friend invite");
275             if (decision) {
276                 client.acceptFriend(friend.id());

```

```

277         }
278     });
279 }
280
281 @Override
282 public void addFriendEvent(Friend friend) {
283     System.out.println("createChatEvent");
284     System.out.println(currentNode.getType());
285     if (currentNode.getType().equals("friends")) {
286         Platform.runLater(() -> loadButtons(client.getFriends()));
287         if (currentNodeID.equals("list")) {
288             Platform.runLater(() -> currentNode.loadContent(Collections.singletonList(friend)));
289         }
290     }
291 }
292
293 @Override
294 public void deleteChatEvent() {
295     System.out.println("deleteChatEvent");
296     if (currentNode.getType().equals("chatrooms")) {
297         Platform.runLater(() -> loadButtons(client.getChats()));
298         if (currentNodeID.equals("list")) {
299             Platform.runLater(() -> currentNode.loadContent(new ArrayList<>(client.getChats())));
300         }
301     }
302 }
303
304 @Override
305 public void unfriendEvent() {
306     System.out.println("unfriendEvent");
307     if (currentNode.getType().equals("friends")) {
308         Platform.runLater(() -> loadButtons(client.getFriends()));
309         if (currentNodeID.equals("list")) {
310             Platform.runLater(() -> currentNode.loadContent(new ArrayList<>(client.getFriends())));
311         }
312     }
313 }
314
315 @Override
316 public void serverDisconnectEvent(String content, String header) {
317     Platform.runLater(() -> SceneController.submitPopup(

```

```

318             new FXMLLoader(getClass().getResource("/com/quap/desktopapp/popup/popup.fxml")),
319             (Stage) Stage.getWindows().stream().filter(Window::isShowing).findFirst().orElse(null),
320             content,
321             header));
322         }
323     }

```

VII – Client.java

```

001 package com.quap.client;
002
003 import com.quap.client.data.UserdataReader;
004 import com.quap.client.domain.*;
005 import com.quap.client.utils.LoginClientObserver;
006 import com.quap.client.utils.MainClientObserver;
007 import com.quap.client.utils.Prefixes;
008 import com.quap.client.utils.Suffixes;
009 import org.json.JSONArray;
010 import org.json.JSONObject;
011
012 import java.io.BufferedReader;
013 import java.io.IOException;
014 import java.io.InputStreamReader;
015 import java.io.PrintWriter;
016 import java.net.InetAddress;
017 import java.net.Socket;
018 import java.net.SocketException;
019 import java.net.UnknownHostException;
020 import java.util.ArrayList;
021 import java.util.Date;
022 import java.util.HashMap;
023 import java.util.List;
024
025 /**
026  * Die Klasse Client ist für die netzwerktechnischen Aspekte des Clients zuständig.
027  * Darunter auch eingehende Nachrichten verarbeiten und ausgehende senden.
028  */
029 public class Client {
030     private final HashMap<Prefixes, String> prefixes = new HashMap<>();
031     private final HashMap<Suffixes, String> suffixes = new HashMap<>();

```

```

032     private final Socket socket;
033     private Thread listen;
034     private final BufferedReader reader;
035     private final PrintWriter writer;
036     private final ArrayList<Friend> friends = new ArrayList<>();
037     private final ArrayList<Chat> chats = new ArrayList<>();
038     private int id, chatID;
039     private String username;
040     private String password;
041     private UserdataReader dataReader;
042     private final List<MainClientObserver> mainClientObservers = new ArrayList<>();
043     private final List<LoginClientObserver> loginClientObservers = new ArrayList<>();
044
045     {
046         try {
047             String name = InetAddress.getLocalHost().getHostName();
048             System.out.println(name);
049         } catch (UnknownHostException e) {
050             e.printStackTrace();
051         }
052         for (Prefixes p : Prefixes.values()) {
053             prefixes.put(p, "/" + p.name().toLowerCase().charAt(0) + "+/");
054         }
055         for (Suffixes s : Suffixes.values()) {
056             suffixes.put(s, "://" + s.name().toLowerCase().charAt(0) + "-/");
057         }
058     }
059
060     /**
061      * Im Konstruktor wird versucht eine Verbindung zum Server herzustellen
062      * und Writer und Reader werden auf die Socket Streams gesetzt.
063      * @param address Die Server-IP
064      * @param port Der Server-Port
065      * @throws IOException bei ungültigen Parametern
066      */
067     public Client(String address, int port) throws IOException {
068         socket = new Socket(InetAddress.getByName(address), port);
069         writer = new PrintWriter(socket.getOutputStream(), true);
070         reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
071         listen();
072     }

```

```

073     public List<UserContent> getFriends() {
074         return new ArrayList<>(friends);
075     }
076
077     public List<UserContent> getChats() {
078         return new ArrayList<>(chats);
079     }
080
081     public String getUsername() {
082         return username;
083     }
084
085
086     /**
087      * Diese Methode verpackt die Daten, welche für die Authentifizierung benötigt werden, in ein Json-Format,
088      * welches der Server verarbeiten kann.
089      * @param name Nutzernname
090      * @param password Nutzerpasswort
091      * @param existing ob es sich um einen existierenden oder neuen Nutzer handelt (SignIn/SignUp)
092      */
093     public void authorize(String name, String password, boolean existing) {
094         this.username = name;
095         this.password = password;
096         JSONObject json = new JSONObject();
097         json.put("name", this.username);
098         json.put("password", this.password);
099         json.put("existing", existing);
100         sendAuthentication(json.toString());
101     }
102
103     public void connectDB() {
104         dataReader = new UserdataReader(username, password);
105     }
106
107     /**
108      * Beim Aufruf dieser Methode wird dem Server der Verbindungsabbruch mitgeteilt und die Socket des Clients
109      * wird geschlossen.
110      * @param status false bei Fehler
111      */
112     public void disconnect(boolean status) {
113         sendDisconnect(status);

```

```

113     listen.interrupt();
114     new Thread(this::closeSocket).start();
115 }
116
117 /**
118 * Diese Methode wirft einen Thread aus, welche auf eingehende Daten wartet
119 * und diese an die process-Methode zum Verarbeiten übergibt.
120 */
121 public void listen() {
122     listen = new Thread(() -> {
123         String message;
124         while (!socket.isClosed() && reader != null) {
125             try {
126                 message = reader.readLine();
127                 if (message != null) {
128                     process(message);
129                 }
130             } catch (SocketException se) {
131                 disconnect(false);
132             } catch (IOException e) {
133                 e.printStackTrace();
134                 listen.interrupt();
135             try {
136                 reader.close();
137                 writer.close();
138             } catch (IOException io) {
139                 io.printStackTrace();
140             }
141         }
142     });
143     listen.start();
144 }
145
146 /**
147 * Diese Methode differenziert die einkommenden Daten anhand ihrer Metadaten
148 * und führt für verschiedene Werte unterschiedliche Aktionen aus.
149 * @param content Der ganze Inhalt inclusive der Metadaten
150 */
151 private void process(String content) {
152     System.out.println(content);
153 }
```

```

154     JSONObject root = new JSONObject(content);
155     String returnValue = root.getString("return-value");
156     if (!returnValue.equals("void")) {
157         if (root.has("error") && !root.has("data")) {
158             System.err.println(root.getString("error"));
159             if ("authentication".equals(returnValue)) {
160                 for (LoginClientObserver c : loginClientObservers) {
161                     c.authFailedEvent(root.getString("error"));
162                 }
163             }
164         } else if (root.has("data")) {
165             JSONObject data = root.getJSONObject("data");
166             switch (returnValue) {
167                 case "authentication" -> {
168                     this.id = data.getInt("id");
169                     JSONArray chatrooms = data.getJSONArray("chatrooms");
170                     for (int i = 0; i < chatrooms.length(); i++) {
171                         Chat chat = new Chat(chatrooms.getJSONObject(i));
172                         chats.add(chat);
173                     }
174                     JSONArray privates = data.getJSONArray("private");
175                     for (int i = 0; i < privates.length(); i++) {
176                         Friend friend = new Friend(privates.getJSONObject(i));
177                         friends.add(friend);
178                     }
179                     LoginClientObserver r = null;
180                     for(LoginClientObserver c : loginClientObservers) {
181                         c.authSuccessEvent();
182                         r = c;
183                     }
184                     removeLoginObserver(r);
185                 }
186                 case "message" -> {
187                     int senderID = data.getInt("sender_id");
188                     String senderName = data.getString("sender_name");
189                     int chatID = data.getInt("chat_id");
190                     String messageContent = data.getString("message");
191                     for (MainClientObserver c : mainClientObservers) {
192                         c.messageEvent(new Message(messageContent, new Date(), senderID, senderName));
193                     }
194                     System.out.println("senderID: " + senderID + ", chatID: " + chatID + ", message: " +

```

```

messageContent);
195         dataReader.addMessage(chatID, senderID, senderName, messageContent);
196     }
197     case "command" -> {
198         System.out.println("command found");
199         String statement = data.getString("statement");
200         switch (statement) {
201             case "create-chat" -> {
202                 JSONObject result = data.getJSONObject("result");
203                 Chat chat = new Chat(
204                     result.getString("name"),
205                     result.getInt("chatroom_id"));
206                 chats.add(chat);
207                 for (MainClientObserver c : mainClientObservers) {
208                     c.createChatEvent(chat);
209                 }
210             }
211             case "invite-chat" -> {
212                 JSONObject chatObject = data.getJSONObject("chat");
213                 int senderID = data.getInt("sender_id");
214                 String senderName = data.getString("sender_name");
215                 JSONArray participants = data.getJSONArray("participants");
216                 List<String> participantsList = new ArrayList<>();
217                 for (int i = 0; i < participants.length(); i++) {
218                     String name = participants.getJSONObject(i).getString("name");
219                     int id = participants.getJSONObject(i).getInt("id");
220                     participantsList.add(name + "#" + id);
221                 }
222                 Chat chat = new Chat(
223                     chatObject.getString("name"),
224                     chatObject.getInt("id"),
225                     chatObject.getString("created_at"))
226                 ;
227                 for (MainClientObserver c : mainClientObservers) {
228                     c.inviteEvent(chat, senderID, senderName, participantsList);
229                 }
230             }
231             case "join-chat" -> {
232                 JSONObject chatObject = data.getJSONObject("chat");
233                 Chat chat = new Chat(
234                     chatObject.getString("name"),

```

```

235                         chatObject.getInt("id"),
236                         chatObject.getString("created_at")
237                     );
238                     chats.add(chat);
239                     for (MainClientObserver c : mainClientObservers) {
240                         c.joinChatEvent(chat);
241                     }
242                 }
243             case "friend-request" -> {
244                 int senderID = data.getInt("sender_id");
245                 String senderName = data.getString("sender_name");
246                 Friend friend = new Friend(
247                     senderName, senderID
248                 );
249                 for (MainClientObserver c : mainClientObservers) {
250                     c.friendRequestEvent(friend);
251                 }
252             }
253             case "add-friend" -> {
254                 Friend friend = new Friend(data.getJSONObject("friend"));
255                 friends.add(friend);
256                 for (MainClientObserver c : mainClientObservers) {
257                     c.addFriendEvent(friend);
258                 }
259             }
260             case "delete-friend" -> {
261                 friends.removeIf(friend -> friend.id() == data.getInt("friend_id"));
262                 for(Friend friend : friends) {
263                     if(friend.id() == data.getInt("friend_id")) {
264                         friends.remove(friend);
265                         dataReader.deleteMessagesByChat(friend.chatID());
266                     }
267                 }
268                 for (MainClientObserver c : mainClientObservers) {
269                     c.unfriendEvent();
270                 }
271             }
272         }
273     }
274     case "disconnect" -> {
275         for (MainClientObserver c : mainClientObservers) {

```

```

276                     c.serverDisconnectEvent("The server does not respond anymore and has disconnected with
the status '" + data.getString("status") + "'", "Warning");
277                 }
278             listen.interrupt();
279             new Thread(this::closeSocket).start();
280         }
281     }
282 } else {
283     System.err.println("Unknown package content");
284 }
285 }
286 }
287
288 /**
289 * Diese Methode gibt die Client-Adresse und die Server Adresse zurück.
290 * @return Connection Information
291 */
292 public String getConnectionInfo() {
293     return socket.getLocalSocketAddress()+":"+socket.getLocalPort() + " --> " +
socket.getRemoteSocketAddress();
294 }
295
296 /**
297 * Diese Methode verpackt die Daten, welche zum übermitteln der Nachricht benötigt werden,
298 * damit der Server diese verarbeiten kann.
299 * @param message Die Nachricht, die im Chat angezeigt wird
300 */
301 public void sendMessage(String message) {
302     JSONObject json = new JSONObject();
303     json.put("return-value", "message");
304     JSONObject data = new JSONObject();
305     data.put("message", message);
306     data.put("chat_id", chatID);
307     data.put("sender_id", id);
308     data.put("sender_name", username);
309     json.put("data", data);
310     String output = prefixes.get(Prefixes.MESSAGE) + json + suffixes.get(Suffixes.MESSAGE);
311     writer.println(output);
312 }
313 }
314

```

```

315     /**
316      * Diese Methode kennzeichnet ihren Input als Authentifikation und sendet die Daten an den Server.
317      * @param authentication Die Daten im JSON-Format
318      */
319     public void sendAuthentication(String authentication) {
320         String output = prefixes.get(Prefixes.AUTHENTICATION) + authentication +
321         suffixes.get(Suffixes.AUTHENTICATION);
322         writer.println(output);
323     }
324
325     /**
326      * Diese Methode kennzeichnet ihren Input als Befehl und sendet die Daten an den Server.
327      * @param command Die Daten im JSON-Format
328      */
329     public void sendCommand(String command) {
330         String output = prefixes.get(Prefixes.COMMAND) + command + suffixes.get(Suffixes.COMMAND);
331         writer.println(output);
332     }
333
334     /**
335      * Diese Methode kennzeichnet ihren Input als Verbindungsabbau und sendet die Daten an den Server.
336      * @param status false = Fehler
337      */
338     private void sendDisconnect(boolean status) {
339         JSONObject json = new JSONObject();
340         json.put("status", status);
341         String output = prefixes.get(Prefixes.DISCONNECT) + json + suffixes.get(Suffixes.DISCONNECT);
342         writer.println(output);
343     }
344
345     /**
346      * Diese Methode enthält einen kritischen Codebereich, welcher Threadsafe geschützt ist.
347      * Beim Aufruf dieser Methode wird die Socket geschlossen.
348      */
349     private void closeSocket() {
350         synchronized (socket) {
351             try {
352                 socket.close();
353             } catch (IOException e) {
354                 e.printStackTrace();
355             }
356         }
357     }

```

```

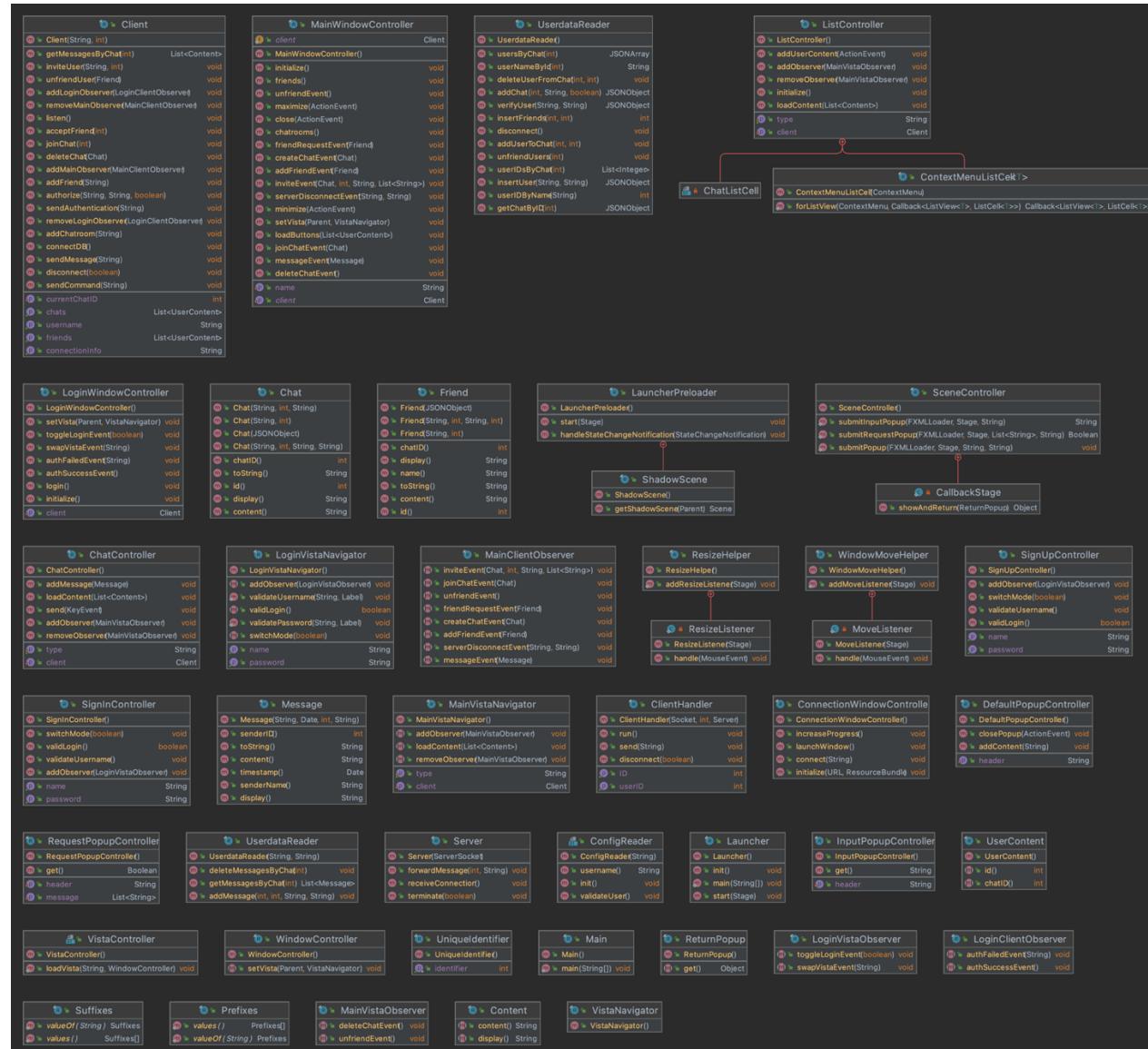
355         }
356     }
357
358     public List<Content> getMessagesByChat(int id) {
359         return new ArrayList<>(
360             dataReader.getMessagesByChat(id)
361         );
362     }
363
364     public void setCurrentChatID(int chatID) {
365         this.chatID = chatID;
366     }
367
368     public void addMainObserver(MainClientObserver observer) {
369         mainClientObservers.add(observer);
370     }
371
372     public void removeMainObserver(MainClientObserver observer) {
373         mainClientObservers.remove(observer);
374     }
375
376     public void addLoginObserver(LoginClientObserver observer) {
377         loginClientObservers.add(observer);
378     }
379
380     public void removeLoginObserver(LoginClientObserver observer) {
381         loginClientObservers.remove(observer);
382     }
383
384
385
386     public void addChatroom(String input) {
387         JSONObject json = new JSONObject();
388         json.put("type", "create-chat");
389         JSONObject data = new JSONObject();
390         data.put("chatname", input);
391         data.put("sender_id", id);
392         json.put("data", data);
393         sendCommand(json.toString());
394     }
395

```

```
396     public void addFriend(String input) {
397         JSONObject json = new JSONObject();
398         json.put("type", "request-user");
399         JSONObject data = new JSONObject();
400         data.put("username", input);
401         data.put("sender_id", id);
402         json.put("data", data);
403         sendCommand(json.toString());
404     }
405
406     public void joinChat(int chatID) {
407         JSONObject json = new JSONObject();
408         json.put("type", "join-chat");
409         JSONObject data = new JSONObject();
410         data.put("chatroom_id", chatID);
411         data.put("sender_id", id);
412         json.put("data", data);
413         sendCommand(json.toString());
414     }
415
416     public void inviteUser(String username, int chatID) {
417         JSONObject json = new JSONObject();
418         json.put("type", "invite-user");
419         JSONObject data = new JSONObject();
420         data.put("username", username);
421         data.put("chat_id", chatID);
422         data.put("sender_id", id);
423         json.put("data", data);
424         sendCommand(json.toString());
425     }
426
427     public void deleteChat(Chat chat) {
428         chats.remove(chat);
429         dataReader.deleteMessagesByChat(chat.id());
430         JSONObject json = new JSONObject();
431         json.put("type", "delete-chat");
432         JSONObject data = new JSONObject();
433         data.put("chat_id", chat.id());
434         data.put("sender_id", id);
435         json.put("data", data);
436         sendCommand(json.toString());
```

```
437     }
438
439     public void acceptFriend(int id) {
440         JSONObject json = new JSONObject();
441         json.put("type", "accept-friend");
442         JSONObject data = new JSONObject();
443         //data.put("username", username);
444         data.put("friend_id", id);
445         data.put("sender_id", this.id);
446         json.put("data", data);
447         sendCommand(json.toString());
448     }
449
450     public void unfriendUser(Friend friend) {
451         friends.remove(friend);
452         dataReader.deleteMessagesByChat(friend.chatID());
453         JSONObject json = new JSONObject();
454         json.put("type", "unfriend-user");
455         JSONObject data = new JSONObject();
456         data.put("friend_id", friend.id());
457         data.put("sender_id", id);
458         data.put("chat_id", friend.chatID());
459         json.put("data", data);
460         sendCommand(json.toString());
461     }
462 }
```

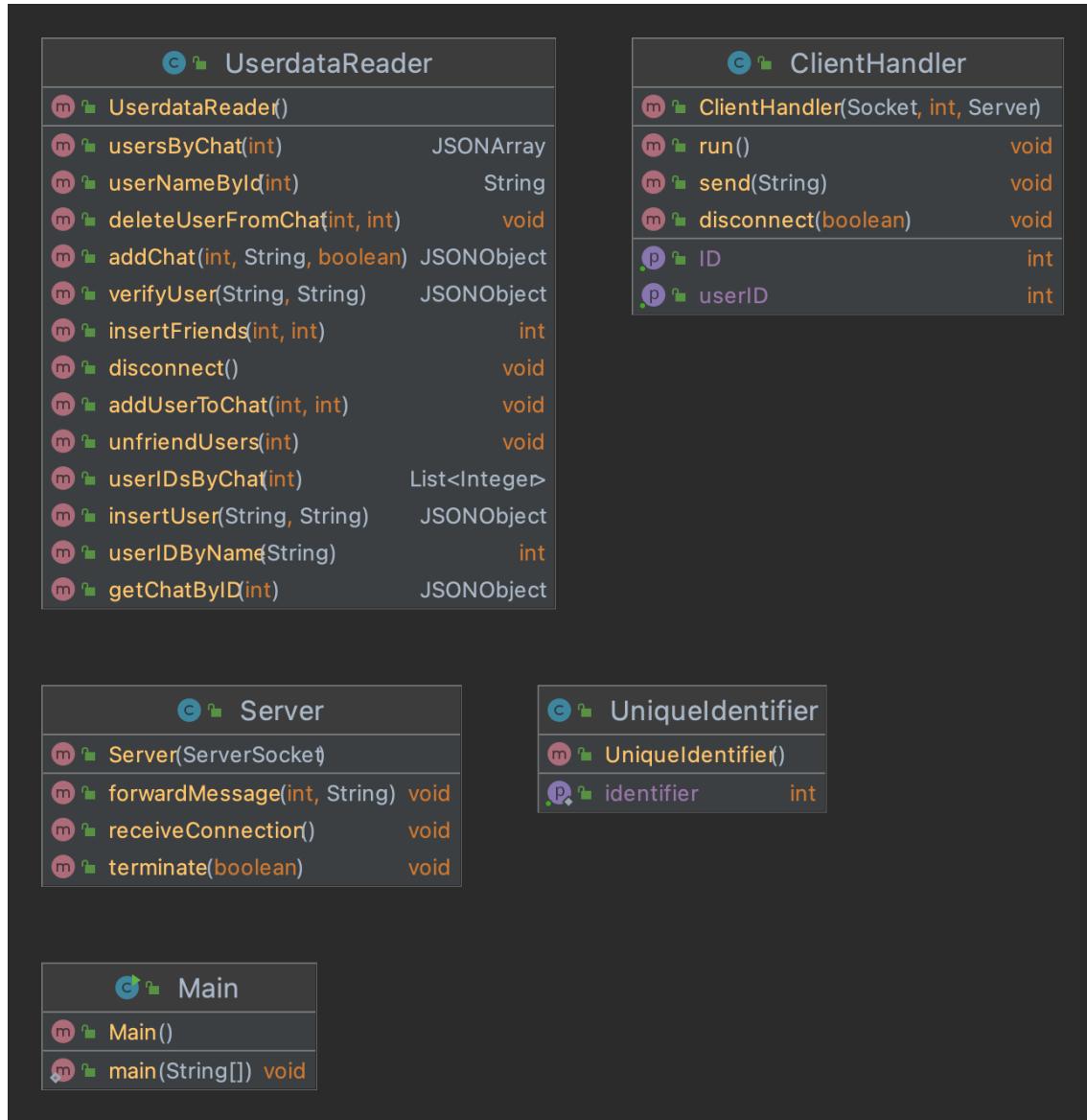
VIII – Quap Class Diagramm



IX – Client Class Diagramm



X – Server Class Diagramm



```
Thread(ThreadGroup gruppe, String name) oder
Thread(ThreadGroup gruppe, Runnable objekt, String name)
```

einer Gruppe zugeordnet werden.

7.4 Priorität

Mit

```
public final void setPriority (int prioritaet)
```

kann die Priorität eines Threads gesetzt werden. Je höher die Zahl, desto höher ist die Priorität. Sie ist einstellbar von `Thread.MIN_PRIORITY` bis `Thread.MAX_PRIORITY`. Ein Thread wird defaultmäßig mit der Priorität des ihn anlegenden Prozesses ausgestattet. Default ist `Thread.NORM_PRIORITY`.

7.5 Synchronisation

Java garantiert, dass die meisten primitiven Operationen atomar (unteilbar) durchgeführt werden. Das sind z.B. Operationen, zu denen der Zugriff auf die Standardtypen (außer `long` und `double`) und auch die Referenztypen gehört (gemeint sind z.B. Zuweisungen an einen Referenztyp selbst).

Zum Schutz von "kritischen Bereichen" stellt Java ein Monitorkonzept zur Verfügung. Jedes Objekt mit kritischem Bereich bekommt zur Laufzeit einen Monitor. Kritische Bereiche sind eine oder mehrere Methoden oder Anweisungsböcke, die von nicht mehr als einem Thread gleichzeitig durchlaufen werden dürfen.

Das Grundprinzip ist, dass individuelle Code-Blöcke über den Konstrukt

```
synchronized (irgendeinObjekt) { irgendeinCode }
```

synchronisiert werden können.

Eine `synchronized` Methode ist eine Methode, deren gesamte Implementation als ein `synchronized` Block

```
synchronized (this) { Methodencode }
```

zu verstehen ist. Das Monitor-Objekt ist das Objekt, wofür die Methode aufgerufen wird. Bei `static` Methoden wird das zur Klasse gehörige Klassenobjekt verwendet:

```
synchronized (this.getClass ()) { staticMethodencode }
```

Synchronisation ist so implementiert, dass für jedes Objekt (inklusive Klassenobjekten) ein nicht zugängliches Lock verwaltet wird, das im wesentlichen ein Zähler ist. Wenn der Zähler beim Betreten eines über das Objekt synchronisierten Blocks oder einer synchronisierten Methode *nicht* Null ist, dann wird der Thread blockiert (suspended). Er wird in eine Warteschlange für das Objekt gestellt (**Entry Set**), bis der Zähler Null ist. Wenn ein synchronisierter Bereich betreten wird, dann wird der Zähler um Eins erhöht. Wenn ein synchronisierter Bereich verlassen wird, dann wird der Zähler um Eins erniedrigt, (auch wenn der Bereich über eine Exception verlassen wird.)

Ein Thread, der ein Lock auf ein Objekt hat, kann mehrere kritische Bereiche oder Methoden desselben Objekts benutzen, wobei der Zähler beim Betreten des Bereichs hochgesetzt, beim Verlassen heruntergesetzt wird.

Nicht-synchronisierte Methoden oder Bereiche können von anderen Threads verwendet werden, auch wenn ein Thread ein Lock auf das Objekt hat. Insbesondere können alle Daten eines eventuell gelockten Objekts benutzt werden, sofern sie zugänglich (sichtbar) sind.

Der Modifikator `synchronized` wird nicht automatisch vererbt. Wenn also eine geerbte Methode auch in der abgeleiteten Klasse `synchronized` sein soll, so muss das angegeben werden. Bei Schnittstellen kann `synchronized` nicht angegeben werden, aber die Methoden können `synchronized` implementiert werden.

7.5.1 synchronized

Syntaktisch gibt es folgende Möglichkeiten:

1. `synchronized` als Modifikator einer Methode: Bevor die Methode von einem Thread für ein Objekt abgearbeitet wird, wird das Objekt von dem betreffenden Thread zum ausschließlichen Gebrauch von synchronisiertem Code des Objekts reserviert. Dabei wird der Thread eventuell so lange blockiert, bis der synchronisierte Code des Objekts nicht mehr von anderen Threads benötigt wird.
2. `synchronized (Ausdruck) Block`: Der Ausdruck muss in einem Objekt oder Feld resultieren. Bevor ein Thread den Block betrifft, wird versucht, ein Lock auf das Resultat des Ausdrucks zum ausschließlichen Gebrauch synchronisierten Codes zu bekommen. Dabei wird der Thread eventuell so lange blockiert, bis der synchronisierte Code nicht mehr von anderen Threads benötigt wird.
3. `synchronized (Ausdruck.getClass ()) Block`: Der Ausdruck muss in einem Objekt resultieren, dessen Klasse ermittelt wird. Damit werden `synchronized` Klassenmethoden geschützt. Bevor ein Thread den Block betrifft, werden alle `synchronized` Klassenmethoden der betreffenden Klasse oder über die Klasse synchronisierte Code-Stücke zum ausschließlichen Gebrauch reserviert.
4. Konstruktoren können nicht `synchronized` definiert werden. Das ergibt einen Compilezeit-Fehler.

Mit `synchronized` werden Codestücke definiert, die *atomar* oder *unteilbar* bezüglich eines Objekts durchzuführen sind. Das Problem des gegenseitigen Ausschlusses ist damit schon gelöst und wesentlich eleganter gelöst als etwa mit Semaphoren.

 **git** --distributed-is-the-new-centralized

Search entire site...

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient **staging areas**, and **multiple workflows**.



About
The advantages of Git compared to other source control systems.

Documentation
Command reference pages, Pro Git book content, videos and other material.

Downloads
GUI clients and binary releases for all major platforms.

Community
Get involved! Bug reporting, mailing list, chat, development and more.

Latest source Release
2.35.1
[Release Notes \(2022-01-29\)](#)

[Download for Mac](#)



Mac GUIs **Tarballs**

Windows Build **Source Code**


Pro Git by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

Companies & Projects Using Git

Google FACEBOOK Microsoft  LinkedIn   PostgreSQL

</> About this site
Patches, suggestions, and comments are welcome.

Git is a member of Software Freedom Conservancy

XIII

THOMAS KRENN | Thomas-Krenn-Wiki durchsuchen | Onlineshop DE -

wiki Lesen Quelltext Versionsgeschichte Benutzerkonto erstellen

Git Grundbegriffe

Hauptseite > Themenübersicht > Git

Git ist ein verteiltes Versionierungssystem welches frei als Open-Source zur Verfügung gestellt wird. Einige bekannte Namen wie der Linux-Kernel, Eclipse oder Android verwenden Git für die Verwaltung ihrer Projekte.^[1] Ähnlich zu Subversion (SVN) wird Git für die Versionskontrolle (stetige Protokollierung von Änderungen) von Dateien eingesetzt. Im Gegensatz zu SVN gibt es jedoch einige grundlegende Unterschiede, andere Begriffe und Schemata können über auch direkt umgelegt werden.^[2] Dieser Artikel erläutert grundlegende Begriffe die immer wieder in Zusammenhang mit Git auftauchen, außerdem werden die Prinzipien eines verteilten Versionierungssystems erläutert.

Inhaltsverzeichnis [Verbergen]
Links auf diese Seite Änderungen an verlinkten Seiten Spenderkennung Druckversion Permanenter Link Seiteninformationen
In anderen Sprachen English

Was ist Git?

Git wird für die Versionierung von Dateien eingesetzt. Vor allem im Programmierbereich dient es dazu die eigenen Änderungen zu überwachen, sie rückgängig zu machen, sie oderne über sogenannte "Repositories" (Repos) zur Verfügung zu stellen oder Aktualisierungen von anderen einzuholen. Im Gegensatz zu SVN besitzt Git kein zentrales Repository sondern tritt als verteiltes System auf - aus der Sicht eines Bearbeiters ist die eigene Arbeitskopie (Working Copy) ein eigenes Repo für sich. Bei Bedarf können Änderungen von öffentlich zugänglichen Repos in die eigene Working Copy eingelesen werden oder die eigenen Änderungen z.B. über einen Patch den anderen zugänglich gemacht werden.

Dadurch ergibt sich um Repo für Bearbeiter folgende Möglichkeiten:^[3]

- Die eigene Arbeit kann einfach wieder in die zentrale Basis integriert werden.
- Es kann zeitgleich weiterentwickelt werden, z.B. jeder von verschiedenen Features.
- Die Versionierung verhindert, dass bereits getätigten Arbeiten verloren gehen bzw. überschrieben werden.
- Bei Bedarf kann zu führen Versionen zurückgekehrt werden oder simultan an verschiedenen Versionen gearbeitet werden (auch "Branches" genannt).

Git bietet über noch viel mehr, welche Eigenschaften es anderen Systemen überlegen macht z.B. unter Why Git is better than X⁴ nachgelesen werden.

Wichtige Begriffe

Repository und Working Copy: In einem Repository bzw. einem Repo befinden sich alle Dateien inklusive aller vorangegangenen Versionen. Dadurch stehen stets alle Änderungen zur Verfügung, die von einer Datei ins Repo gespielt wurden und es kann nachvollzogen werden Wer, Wann, welche Änderungen durchgeführt hat. Das besondere an Git ist, dass jede lokale Working Copy eines Users (ein "Klon" - via "git clone") wieder ein vollständiges, eigenes lokales Repo darstellt. Es existieren somit mehrere Kopien der Repos, der, der einen Klon besitzt, kann dann abrufen - inklusive kompletter History, auch offline und ohne Abhängigkeit von einem zentralen Server.^[5] Die Änderungen aus dem eigenen Repodaten Working Copy können dann auf einen Punkt oder Schritt für Schritt, wenn sie als public angeschaut werden, wieder in das Remote-Repo "gepusht", wenn sie "git push".

Branch: Beim Einsatz von Git eilen - branch - sich verzweigen darf, um einen separaten Arbeitsweg zu erstellen. Dieser kann dann auch als neuer Kontext gesehen werden, in dem gearbeitet wird. So kann z.B. die Programmierung eines Sicherheits-Patches in einer eigenen Branch erfolgen (im Kontext des Patches), der bei Fertigstellung des Testen und Implementieren in die zentrale Basis integriert werden kann.

Verzweigung: Sobald an einer Working Copy gearbeitet wird produziert Git alle getätigten Änderungen mit. Mittels commit können die Änderungen zu dem Repository hinzugefügt werden, eine neue Version der Datei(n) befinden sich dann im Repo. Anschließend können verschiedene Versionen miteinander verglichen, Änderungen rückgängig oder zu einer früheren Version zurückgekehrt werden. Die Log-Informationen, die von Git mit aufgezeichnet werden können mit "git log" ausgegeben werden, "git status" listet die noch nicht ins Repo gespielten Änderungen der Working Copy auf.

Einzelnachweise

1. [T Git-Übersicht #](#) (Git Projekt-Website)

2. [T Git Crash Course for SVN users](#) (kernel.org)

3. [T Git for designers #](#) (hotch.emp.com)

4. [T Why Git? #](#) (issaris.blogspot.com)

f t e Druckversion

Das könnte Sie auch interessieren

Git Commits mit rebase zusammenführen Zum Artikel

Git diff mit odt-Dateien Zum Artikel

Git Grundbefehle Zum Artikel

Kategorie: Git

Thomas-Krenn steht für Server Made in Germany.
Wir assemblingen und liefern europaweit innerhalb von 24 Stunden.
Unter www.thomas-krenn.com können Sie Ihre Server individuell konfigurieren.

Jetzt Thomas-Krenn Newsletter abonnieren
email address OK

Datenschutz | Über Thomas-Krenn-Wiki | Haftungsausschluss

Facebook Twitter LinkedIn YouTube Instagram



Hello World

Follow this Hello World exercise to get started with GitHub.

In this article

- [Introduction](#)
- [Creating a repository](#)
- [Creating a branch](#)
 - [Create a branch](#)
- [Making and committing changes](#)
- [Opening a pull request](#)
- [Merging your pull request](#)
- [Next steps](#)

Introduction

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

This tutorial teaches you GitHub essentials like repositories, branches, commits, and pull requests. You'll create your own Hello World repository and learn GitHub's pull request workflow, a popular way to create and review code.

In this quickstart guide, you will:

- Create and use a repository

- Start and manage a new branch
- Make changes to a file and push them to GitHub as commits
- Open and merge a pull request

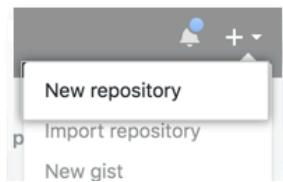
To complete this tutorial, you need a [GitHub account](#) and Internet access. You don't need to know how to code, use the command line, or install Git (the version control software that GitHub is built on). If you have a question about any of the expressions used in this guide, head over to the [glossary](#) to find out more about our terminology.

Creating a repository

A repository is usually used to organize a single project. Repositories can contain folders and files, images, videos, spreadsheets, and data sets -- anything your project needs. Often, repositories include a *README* file, a file with information about your project. *README* files are written in the plain text Markdown language. You can use this [cheat sheet](#) to get started with Markdown syntax. GitHub lets you add a *README* file at the same time you create your new repository. GitHub also offers other common options such as a license file, but you do not have to select any of them now.

Your `hello-world` repository can be a place where you store ideas, resources, or even share and discuss things with others.

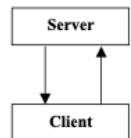
- ➊ In the upper-right corner of any page, use the + drop-down menu, and select **New repository**.



16.2 Server/Client-Programmierung

Rechner im Netz bieten anderen Rechnern Dienste an bzw. nutzen angebotene Dienste. So ist es möglich, Aufgaben als Dienste auf wenigen Rechnern zu installieren und mehreren Rechnern gleichzeitig zur Verfügung zu stellen. Die Wartung und Pflege solcher Dienste wird durch die Konzentration auf wenige Rechner vereinfacht.

Client - Server - Architektur



Server (Diener): Ein Programm auf einem Rechner (**Server-Rechner**, **Server-Host**), welches einen *Service* (Dienst) über ein Netzwerk anbietet.

Client (Kunde): Ein Programm auf einem Rechner (**Client-Rechner**, **Client-Host**), welches einen *Service* (Dienst) eines anderer Rechner über ein Netzwerk in Anspruch nimmt.

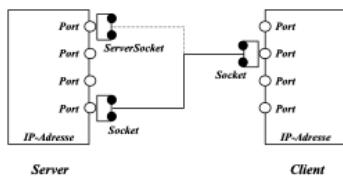
Ein Rechner kann sowohl Server- als auch Client-Rechner sein.

16.2.1 Aufbau einer Server/Client-Verbindung

Server

Verbindungsauflauf

1. Ein portgebundener Serversocket wird erzeugt. Dieser wird für Clientanmeldungen verwendet: IP-Adresse des Servers und Portnummer muss als Zugang für die Dienstleistung dem Client bekannt sein.
2. Der Server wartet auf Anmeldungen von Clients.
3. Hat sich ein Client angemeldet, so wird *serverseitig* ein weiterer Socket zur Abwicklung der Kommunikation mit dem Client eingerichtet.



XVI

C | X-1 Nichtsequentialität 1 Kausalitätsprinzip

Systemprogrammierung

Prozesssynchronisation: Nichtsequentialität

Wolfgang Schröder-Preikschat

Lehrstuhl Informatik 4

6. November 2013

©woesch (Lehrstuhl Informatik 4) Systemprogrammierung SP2 # WS2013/14 1 / 28

C | X-1 Nichtsequentialität 1 Kausalitätsprinzip 1.1 Parallelisierbarkeit

Nebenläufige Programmabschnitte

Sequentielles \mapsto Nichtsequentialles Programm

Nebenläufigkeit (engl. *concurrency*) bezeichnet das Verhältnis von nicht kausal abhängigen Ereignissen, die sich also nicht beeinflussen

- Aktionen können nebenläufig ausgeführt werden, wenn keine das Resultat des anderen benötigt

```
1: foo = 4711;
2: bar = 42;
3: foobar = foo + bar;
4: barfoo = bar + foo;
5: hal = foobar + barfoo;
```

- Zeile 1 kann nebenläufig zu Zeile 2 ausgeführt werden
- Zeile 3 kann nebenläufig zu Zeile 4 ausgeführt werden

Kausalität (lat. *causa*: Ursache) ist die Beziehung zwischen Ursache und Wirkung, d.h., die ursächliche Verbindung zweier Ereignisse

- Ereignisse sind nebenläufig, wenn keines Ursache des anderen ist

©woesch (Lehrstuhl Informatik 4) Systemprogrammierung SP2 # WS2013/14 3 / 28

Gliederung

- ① Kausalitätsprinzip
 - Parallelisierbarkeit
 - Kausalordnung
- ② Konkurrenz und Koordination
 - Koordinierung
 - Konkurrenz
- ③ Verfahrensweisen
 - Einordnung
 - Kritischer Abschnitt
 - Lebendigkeit
- ④ Zusammenfassung

C | X-1 Nichtsequentialität 1 Kausalitätsprinzip 1.2 Kausalordnung

Ursache und Wirkung

Nebenläufigkeit als relativistischer Begriff von Gleichzeitigkeit

ist Ursache von e_i

ist Wirkung von e_i

ist nebenläufig zu e_i

• ein Ereignis ist nebenläufig zu einem anderen (e_j), wenn es im Anderswo des anderen Ereignisses (e_i) liegt

- d.h., weder in der Zukunft noch in der Vergangenheit des anderen

• das Ereignis ist nicht Ursache/Wirkung des anderen Ereignisses (e_j)

- ggf. aber Ursache/Wirkung anderer (von e_i verschiedener) Ereignisse

©woesch (Lehrstuhl Informatik 4) Systemprogrammierung SP2 # WS2013/14 4 / 28

Rangfolge aus Gründen von Daten- und Zeitabhängigkeit

Aktionen können — um selbst ein korrektes Ergebnis zu produzieren — nebenläufig stattfinden, sofern:

allgemein keine das Resultat der anderen benötigt (S. 3)

- **Datenabhängigkeiten** gleichzeitiger Prozesse beachten

speziell (zusätzlich im Echtzeitbetrieb) keine die **Zeitbedingungen** der anderen verletzt

- Zeitpunkte dürfen nicht/nur selten verpasst werden
- Zeitintervalle dürfen nicht/nur begrenzt gedehnt werden

Umgang mit Ereignissen bzw. Aktionen gleichzeitiger Prozesse

„ist Ursache von“
 „ist Wirkung von“
 „ist nebenläufig zu“ } ~ Koordinierung (vor/zur Laufzeit)
 } = Parallelität (implizit)

Serialisierung gleichzeitiger Aktivitäten

Maßnahme zum korrekten Zugriff gleichzeitiger Prozesse auf gemeinsame aber unteilbare Betriebsmittel, mit sehr unterschiedlicher Auswirkung:

- | | | |
|-------------------------|--|---|
| blockierend | • pessimistische Annahme einer Überlappung | ⊕ |
| nichtblockierend | • optimistische Annahme keiner Überlappung | ⊖ |

Synchronisation (gr. *sýn*: zusammen, *chrónos*: Zeit) bezeichnet allg. das Herstellen von Gleichzeitigkeit

- (a) Koordination der Kooperation und Konkurrenz zwischen Prozessen X
- (b) Abgleich von Echtzeituhren (oder Daten) in verteilten Systemen
- (c) Sequentialisierung von Ereignissen entlang einer Kausalordnung

Herangehensweisen

- | | |
|--------------------|--|
| analytisch | • Prozesseinplanung: implizite Synchronisation |
| konstruktiv | • Programmietechniken: explizite Synchronisation X |

Gliederung

1 Kausalitätsprinzip

- Parallelisierbarkeit
- Kausalordnung

2 Konkurrenz und Koordination

- Koordinierung
- Konkurrenz

3 Verfahrensweisen

- Einordnung
- Kritischer Abschnitt
- Lebendigkeit

4 Zusammenfassung

Betriebsmittel und Betriebsmittelarten



Hardware

- CPU, Speicher
- Geräte (Peripherie)
- *Signale*
- Dateien, E/A-Puffer
- Seitenrahmen
- Deskriptoren, ...
- *Signale, Nachrichten*

Software

Wettbewerb um Betriebsmittel (engl. resource contention)

Anzahl und Art der Betriebsmittel bedingen problemspezifische Verfahren:

- | | |
|-----------------------------|------------------------------------|
| einseitige Synchronisation | • konsumierbare Betriebsmittel |
| mehrseitige Synchronisation | • wiederverwendbare Betriebsmittel |

Unteilbarkeit

Duden¹: Einheit, Ganzheit, Zusammengehörigkeit

(Wirtschaft) Umstand, der die Verteilung der Produktion auf mehrere Unternehmen oder Arbeitskräfte verhindert.

Sinnbild: Übertragung auf nichtsequentielle Programme

(Informatik, Betriebssysteme) Umstand, der die Verteilung der Betriebsmittel auf mehrere Prozessoren oder Prozesse verhindert.

- unteilbar ist ein Betriebsmittel, wenn es zu einem Zeitpunkt von nur genau einem Prozessor/Prozess genutzt werden darf
 - Zugriffsoperationen darauf können/dürfen nicht zeitlich zerteilt werden
 - sie müssen atomar, d.h., als Elementaroperation ausgeführt werden
- teilbar ist ein Betriebsmittel, wenn mehrere Prozessoren/Prozesse es gleichzeitig benutzen dürfen
 - es dem einem entzogen und einem anderen gegeben werden darf

¹<http://www.duden.de/rechtschreibung/Unteilbarkeit>

Wechselseitiger Ausschluss

Serialisierung von Prozessen mit begrenzten/unteilbaren Betriebsmitteln

Sequenzialisierung der Zugriffe gleichzeitiger Prozesse per Protokoll:

Vergabe → das Betriebsmittel sperren und dem Prozess zuteilen

- beim Versuch, ein gesperrtes Betriebsmittel erneut zu belegen, wird der anfordernde Prozess blockiert
- der blockierende Prozess erwartet das Ereignis/Signal zur Freigabe des gesperrten Betriebsmittels, ihm wird die CPU entzogen

Freigabe → das Betriebsmittel dem Prozess wieder entziehen

- sollten Prozesse die Freigabe dieses Betriebsmittels erwarten, wird es sofort der Wiedervergabe zugeführt; das bedeutet:
 - (a) das Betriebsmittel entsperren und alle Prozesse deblockieren, die sich dann wiederholen um die Vergabe zu bemühen haben oder
 - (b) einen Prozess auswählen und ihm das Betriebsmittel zuteilen
- nur der das Betriebsmittel „besitzende“ Prozess kann es freigeben

Konfliktfall bei der Benutzung von Betriebsmitteln

Prozesse befinden sich untereinander im **Konflikt**, wenn:

- nur eine begrenzte Anzahl gemeinsamer Betriebsmitteln vorrätig ist
 - diese Betriebsmittel unteilbar und von derselben Art sind
 - ein Zugriff darauf gleichzeitig geschieht: **gleichzeitige Prozesse** [1]
- Prozesse sind im **Wettstreit** (engl. contention) um ein Betriebsmittel, wenn einer das Betriebsmittel anfordert, das ein anderer bereits besitzt
- der anfordernde Prozess blockiert und wartet auf die Freigabe des Betriebsmittels durch den Prozess, der das Betriebsmittel belegt
 - der das Betriebsmittel belegende Prozess löst den auf die Freigabe des Betriebsmittels wartenden Prozess aus, deblockiert ihn wieder

Wechselseitiger Ausschluss

- n-fach mehrseitige, für $n - 1$ blockierend wirkende Synchronisation
- eine **Option** zum Schutz eines kritischen Abschnitts

Serialisierung gleichzeitiger Prozesse: Koordinierung

Synchronisation ≡ Koordination der Kooperation und Konkurrenz zwischen Prozessen

ko-or-di'nie-ren beordnen; in ein Gefüge einbauen; aufeinander abstimmen; nebeneinanderstellen; Termine ~.

- sich überlappen könnennde Aktivitäten der Reihe nach ausführen
 - gleichzeitige Prozesse im kritischen Abschnitt koordinieren
- „der Reihe nach“ ~ **Verzögerung gleichzeitiger Prozesse erzwingen**
 - (a) potentiell überlappende Prozesse verzögern, der Konflikt ist ungewiss
 - (b) den überlappten Prozess verzögern, der Konflikt ist gewiss

Synchronisationsverfahren...

- wirken einseitig oder mehrseitig
 - unterdrückend, blockierend, nichtblockierend
 - behinderungs-, sperr-, wortfrei



XVII

Sie sind hier:
Bildungsplan 2016: Informatik, Klasse 7 » Rechner und Netze » Hintergrund » Hintergrund: Kommunikation in Rechnernetzen » Bestandteile eines Netzwerks

Bildungsplan 2016: Informatik, Klassenstufe 7	
Daten und Codierung	
Algorithmen	
Rechner und Netze	
Hintergrund	
Stoffverteilung	
Unterrichtsverlauf	
Hintergrund: Kommunikation in Rechnernetzen	
Bestandteile eines Netzwerks	
Adressierung	
Client-Server-Prinzip	
Das Internet	
Internetdienste	
Rollenspiel Netzwerk	
Hintergrund: Die „Cloud“	
Hintergrund: Sicherheit von mobilen Geräten	
Hintergrund: Kryptographie	
Kopiertvorlagen	
Vorlagen Tauschordner	
Lösungen	
Präsentationen	
Software	
Alle Dateien herunterladen	
Informationen zur Fortbildung	
Regionale Fortbildungen	
Autorenteam	

Bestandteile eines Netzwerks

Wikipedia definiert Rechnernetze so:

„Ein Rechnernetz ist ein Zusammenschluss von verschiedenen technischen, primär selbstständigen elektronischen Systemen (insbesondere Computern, aber auch Sensoren, Aktoren, Funktechnologischen Komponenten usw.), der die Kommunikation der einzelnen Systeme untereinander ermöglicht.“
→ Siehe Seite „Rechnernetz“, URL:

<http://de.wikipedia.org/w/index.php?title=Rechnernetz&oldid=80697890> (abgerufen: 4. November 2017)



Bild „Lokales Netz“, Schaller. Erstellt mit Filus-Netzwerk simulation, URL:

<http://www.lernsoftware-filus.de> (November 2016)

Computernetze bestehen also üblicherweise aus Geräten (PCs, Laptops, Drucker, usw.), die mit Kabeln über Switches (=Gerb mit vielen Netzwerkanschlüssen, das Daten an die angeschlossenen Geräte innerhalb eines Netzwerks weiterleiten kann) verbunden sind, können aber auch andere Komponenten enthalten. Natürlich können statt Kabel auch WLAN-Funkverbindungen zum Einsatz kommen. Die in einem Netz verbündeten Rechner heißen **Netzketten** oder kurz **Knoten**. Ein derartiges Netzwerk (z.B. von einer Schule oder einer Firma) wird als Local Area Network (LAN) bezeichnet.

Computernetze bestehen also üblicherweise aus Geräten (PCs, Laptops, Drucker, usw.), die mit Kabeln über Switches (=Gerb mit vielen Netzwerkanschlüssen, das Daten an die angeschlossenen Geräte innerhalb eines Netzwerks weiterleiten kann) verbunden sind, können aber auch andere Komponenten enthalten. Natürlich können statt Kabel auch WLAN-Funkverbindungen zum Einsatz kommen. Die in einem Netz verbündeten Rechner heißen **Netzketten** oder kurz **Knoten**. Ein derartiges Netzwerk (z.B. von einer Schule oder einer Firma) wird als Local Area Network (LAN) bezeichnet.

Ein spezieller Computer, der als Server fungiert, ist dabei nicht notwendig. Genau genommen kann jeder Computer spezielle Dienste bereitstellen (z.B. Dateien für die anderen Computer speichern). Diese Dienste werden als Server-Programme bezeichnet. Leider wird oft auch der Computer, der viele Server-Dienste bereitstellt, als Server bezeichnet. Dies trägt zur Verwirrung bei.

Topologie des Netzes

Wenn viele Geräte miteinander verbunden werden sollen, stellt sich die Frage, wie diese miteinander zu verbinden sind. Es geht hier nicht darum, ob mit oder ohne Kabel verbunden wird, sondern um einen rein topologischen (=die Anordnung im Raum betreffenden) Sachverhalt.

Es gibt folgende Netzwerk-Topologien:

Stern-Topologie

In der Stern-Topologie unterhält eine zentrale Station die Verbindungen zu allen anderen Stationen. Jede Station ist über eine eigene physikalische Leitung an die zentrale Station angebunden. Es handelt sich im Regelfall um einen Switch, der die Verteilerfunktion für die Datenpakete übernimmt.



Bildquelle: NetworkTopologies.png von Foobaz [PD] via [Wikimedia Commons](http://commons.wikimedia.org) (abgerufen: 4.1.2017)

In vielen Haushalten steht ein Router, der vom Provider geliefert wird. Dieser erfüllt neben der Anbindung an das Internet die Funktion des zentralen Switches, an den die Endgeräte per Kabel oder WLAN angeschlossen sind.

Baum-Topologie

Die Baum-Topologie ist eine erweiterte Stern-Topologie. Größere Netze nehmen eine solche Struktur an. An der Wurzel und jeder Verästelung befinden sich Switches. Dabei ist die Netzwerklast an der Wurzel des Baumes am höchsten. Dort muss ein leistungsfähiger Switch verwendet werden.

In den meisten Schulen wird diese Topologie verwendet. Der zentrale Server ist dabei direkt an der Wurzel des Baums angebunden. Von dort gehen Leitungen in die einzelnen Gebäude / Gebäudeteile. Innerhalb dieser Gebäude werden die einzelnen Netzwerkanschlüsse über weitere Switches angeschlossen.

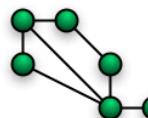


Bildquelle: NetworkTopologies.png von Foobaz [PD] via [Wikimedia Commons](#) (abgerufen: 4.1.2017)

Vermaschte Topologie

In der vermaschten Topologie sind die einzelnen Knoten über mehrere Wege miteinander verbunden. Dadurch bleibt dieses Netz auch bei Ausfall einzelner Leitungen oder Knoten funktionsfähig.

Das Internet ist in weiten Teilen ein vermaschtes Netz. Es ist daher gegen den Ausfall einzelner Komponenten gesichert. Trotzdem gibt es "Hauptverkehrsadern" (die Backbone-Leitungen). Die Router an diesen Backbone-Leitungen sind für Abhöraktionen besonders interessant. Die einzelnen Knoten stellen dabei die Schnittstellen (Router) zu den lokalen Netzwerken dar, die in einer eigenen Topologie realisiert sind.



Bildquelle: NetworkTopologies.png von Foobaz [PD] via [Wikimedia Commons](#) (abgerufen: 4.1.2017)

Geschwindigkeit der Datenübertragung

Bei der Planung eines Netzwerks spielt der erforderliche Datendurchsatz eine große Rolle. Neben der Topologie des Netzes ist dabei die Datenübertragungsgeschwindigkeit entscheidend. In vielen Werbebroschüren der Netzwerkprovider wird vor allem mit hohen Übertragungsgeschwindigkeiten geworben.

Normale Netzwerkkabel und Switches sind heute auf einen Datendurchsatz von 100 MBit/s (Megabit pro Sekunde) oder 1 GBit/s (Gigabit pro Sekunde) ausgelegt. Das entspricht 12,5 Megabyte/s, bzw. 125 Megabyte/s. Mit 100 MBit lässt sich ein Digitalbild hoher Auflösung in 0,25s oder ein Film (1,8 Gbyte) in 2,5-min übertragen, wenn die volle Bandbreite zur Verfügung steht.

VDSL wird zurzeit (Oktober 2015) mit einer Geschwindigkeit von max. 100 Mbit/s angeboten. Auch Funknetze bieten mit LTE (4G) heutzutage bis zu 100 MBit/s. Internet Backbones arbeiten mit 10 GBit/s bis max. 1 TBit/s. (vgl. Topologie des Baden-Württembergischen Forschungsnetz Belwue, an das die meisten Schulen angeschlossen sind¹).

Anschluss an das Internet mittels Router

Im Gegensatz zu den Local Area Networks (LAN) ist das Internet ein globales, räumlich nicht beschränktes Netzwerk (WAN – Wide Area Network). Es entsteht durch die Verbindung vieler lokaler Netze. Um ein LAN an das Internet anzuschließen, ist ein Router erforderlich.

Dieser hat mehrere Netzwerkkarten. Jede dieser Netzwerkkarte ist dann Teil eines lokalen Netzwerks. Die Router sorgen dafür, dass der Datenaustausch zwischen diesen lokalen Netzwerken funktioniert. Die Router für Heimnetzwerke erfüllen meistens aber noch weitere Funktionen, die die eigentliche Routing-Funktion verschleieren:

- **VDSL-Modem:** Die Daten werden per VDSL an den Provider geschickt. Der private Router übernimmt das Versenden der Daten im VDSL-Format.
- **Switch:** Die meisten Router haben mehrere Netzwerkanschlüsse, um mehrere private Computer anschließen zu können. Oft ist auch noch eine WLAN-Funktionalität eingebaut. Dadurch spart man sich den Kauf eines separaten Switches.
- **DHCP-Server:** Der Router übernimmt die Konfiguration der Netzwerkkarten der privaten PCs, damit die User sich nicht darum kümmern müssen (die würden es in vielen Fällen auch gar nicht können...).
- **Firewall:** In der Regel ist eine Firewall in den Router integriert, die den Datenverkehr aus dem Internet in das private Netzwerk kontrolliert.
- **Network Adress Translation (NAT):** Der Internetprovider gibt uns für eine beschränkte Zeit eine IP-Adresse. Trotzdem können wir mit mehreren PCs gleichzeitig ins Internet gehen, da der Router die IP-Adresse des PCs bei Anfragen ins Internet gegen seine eigene ersetzt und dafür sorgt, dass die Antwort aus dem Internet an den richtigen PC weitergegeben wird.

¹ Siehe Seite „Topologie des Baden-Württembergischen Forschungsnetz“. URL: <http://www.belwue.de/topology/> (abgerufen: Januar 2017)

Hintergrund: Kommunikation in Rechnernetzen: [!\[\]\(801e99145fad6ede5611e1f5d0ccd0ef_img.jpg\) Herunterladen](#) [odt][181 KB]

Hintergrund: Kommunikation in Rechnernetzen: [!\[\]\(7510f031844ad4008a55bb76ecaf4be0_img.jpg\) Herunterladen](#) [pdf][358 KB]

Weiter zu [► Adressierung](#)

**Elektronik
Kompendium**

[Startseite](#) [Themen ▾](#) [News](#) [Forum](#) [Online-Shop](#)

Grundlagen Netzwerktechnik

Ein Netzwerk ist die physikalische und logische Verbindung von mehreren Computersystemen. Jedes Netzwerk basiert auf Übertragungstechniken, Protokollen und Systemen, die eine Kommunikation zwischen den Teilnehmern eines Netzwerks ermöglichen. Ein einfaches Netzwerk besteht aus zwei Computersystemen. Sie sind über ein Kabel miteinander verbunden und somit in der Lage ihre Ressourcen gemeinsam zu nutzen. Wie zum Beispiel Rechenleistung, Speicher, Programme, Daten, Drucker und andere Peripherie-Geräte. Ein netzwerkfähiges Betriebssystem stellt den Benutzern auf der Anwendungsebene diese Ressourcen zur Verfügung.

Daraus ergeben sich einige Vorteile gegenüber unvernetzten Computern:

- Nutzen gemeinsamer Datenbestände
- Nutzen verfügbarer Ressourcen
- Teilen von Rechenleistung und Speicherkapazität
- zentrales Steuern von Programmen und Daten
- Durchsetzen von Berechtigungen und Zuständigkeiten
- Durchsetzen Datenschutz und Datensicherheit

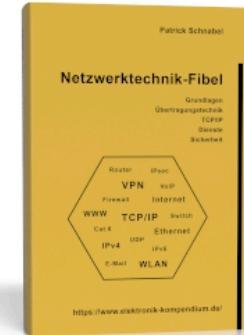
Als es die ersten Computer gab, waren Peripherie-Geräte und Speicher sehr teuer. Die erste Möglichkeit, Peripherie-Geräte gemeinsam zu nutzen, waren manuelle Umschaltboxen. So konnte man zum Beispiel von mehreren Computern aus einen Drucker nutzen. Mit welchem Computer der Drucker verbunden war, wurde über die Umschaltbox bestimmt. Leider haben Umschaltboxen den Nachteil, dass Computer und Peripherie relativ nahe beieinander stehen müssen, weil die Kabellänge physikalisch bedingt begrenzt ist. Der Bedarf zwischen mehreren Computern Daten auszutauschen und Ressourcen zu teilen führte dazu, Computer miteinander zu verbinden bzw. zu vernetzen.

Folge uns



Das Buch zu dieser Webseite

Netzwerktechnik-Fibel

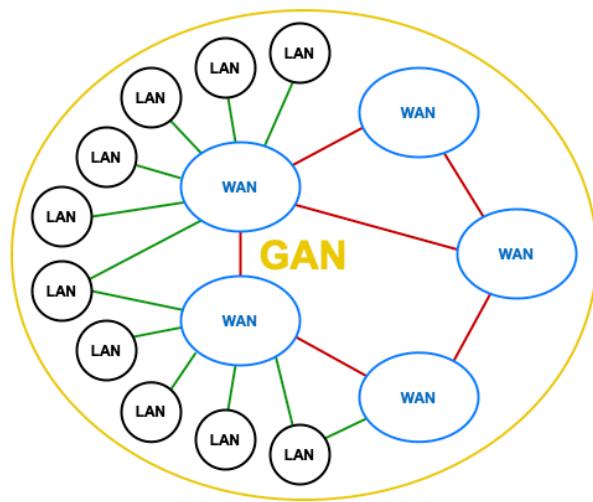


Kundenmeinung:



"Da Bücher, die einfach zu verstehen sind, schwer zu finden

Netzwerk-Dimensionen: LAN, WAN und GAN



Bestimmte Netzwerktechniken unterliegen Beschränkungen, die insbesondere die Reichweite und geografische Ausdehnung des Netzwerks begrenzt. Hierbei unterscheidet man zwischen verschiedenen Netzwerk-Dimensionen für die es unterschiedliche Netzwerktechniken gibt.

- **PAN** - Personal Area Network: personenbezogenes Netz, z. B. Bluetooth
- **LAN** - Local Area Network: lokales Netz, z. B. Ethernet und WLAN
- **MAN** - Metropolitan Area Network: regionales Netz
- **WAN** - Wide Area Network: öffentliches Netz, z. B. DSL und Mobilfunk
- **GAN** - Global Area Network: globales Netz, z. B. das Internet

In der Regel findet ein Austausch zwischen den Netzen statt. Das heißt, dass Netzwerk-Teilnehmer eines LANs auch ein Teilnehmer eines WANs oder eines GANs ist.

Eine 100%ig klare Abgrenzung zwischen diesen Dimensionen ist nicht immer möglich, weshalb man meist nur eine grobe Einteilung vornimmt. So unterscheidet man in der Regel zwischen LAN und WAN, wobei es auch Techniken und Protokolle gibt, die sowohl im LAN, als auch im WAN zum Einsatz kommen.

- Warum vernetzen wir Computersysteme zu einem **LAN**? Um **gemeinsame Ressourcen** zu nutzen.
Zum Beispiel Internet-Zugang, Drucker, Speicher, Rechenleistung, Dienste und Anwendungen.
- Warum verbinden wir uns mit einem **WAN**? Um **Dienste** von anderen zu nutzen oder um eigene Dienste anderen anzubieten. Zum Beispiel Datenverarbeitung und Informationsaustausch.
- Warum verbinden wir uns mit einem **GAN**? Um an einer weltweit verfügbaren **Kommunikationsanwendung** teilzunehmen. Zum Beispiel Internet, Telefonie, Messaging und E-Mail.



Kundenmeinung:



"Die Kommunikationstechnik-Fibel ist sehr informativ und verständlich. Genau das habe ich schon seit langem gesucht. Endlich mal ein Buch, das kurz und bündig die moderne Informationstechnik beleuchtet."

[Kommunikationstechnik-Fibel jetzt bestellen!](#)

Protokolle in der Netzwerktechnik

In der Netzwerktechnik bestimmen Protokolle den Ablauf der Kommunikation zwischen den Systemen. Netzwerk-Protokolle sind eine Sammlung von Regeln, die den Ablauf einer Kommunikation zwischen zwei oder mehr Systemen festlegen. Ein Netzwerk-Protokoll definiert, wie die Kommunikation aufgebaut wird, wie und über was sich die Systeme austauschen und wie die Kommunikation wieder beendet wird. Während einer Kommunikation werden also nicht nur Informationen oder Daten ausgetauscht, sondern zusätzlich Protokoll-Informationen, die beim Empfänger verarbeitet werden.

Typischerweise ist nicht nur ein Netzwerk-Protokoll für die Kommunikation verantwortlich, sondern mehrere, die ganz bestimmte Teilaufgaben innerhalb der Kommunikation übernehmen. Die Einteilung erfolgt anhand eines Schichtenmodells. Ein Protokoll ist in der Regel einer bestimmten Schicht zugeordnet.

- [Protokolle in der Netzwerktechnik](#)

Schichtenmodelle

Weil ein Netzwerk möglichst universell sein soll, also von mehreren Teilnehmern und mehreren Anwendungen gleichzeitig genutzt werden soll, ist die Netzwerk-Kommunikation in Schichten aufgeteilt. Jede Schicht hat ihre Aufgabe und löst dabei ein bestimmtes Teilproblem einer Kommunikation. Sender und Empfänger müssen dabei mit dem gleichen Schichtenmodell arbeiten. Es gibt verschiedene Schichtenmodelle, die sich in der Anzahl der Schichten und somit der Verdichtung der Aufgaben unterscheiden.

- [Schichtenmodelle](#)
- [DoD-Schichtenmodell](#)
- [ISO/OSI-7-Schichtenmodell](#)

Datenübertragung im Netzwerk

Die Kommunikation kann grundsätzlich auf zwei Arten erfolgen. Entweder verbindungsorientiert oder verbindungslos.

Bei der verbindungsorientierten Datenübertragung wird vor dem Austausch der Daten erst eine logische Verbindung hergestellt. Während der Übertragung bleibt die Verbindung zwischen den Kommunikationspartnern aufrechterhalten. Die logische Verbindung bleibt solange bestehen, bis sie durch einen Verbindungsabbau beendet wird.

Bei der verbindungslosen Kommunikation wird keine logische Verbindung und damit auch keine dauerhafte Verbindung aufgebaut. Die Daten werden in kleine Einheiten geteilt. Die Übertragung jeder Einheit wird auf den meisten Protokoll-Schichten als abgeschlossener Vorgang behandelt. Je nach Technik werden die einzelnen Übertragungseinheiten allgemein als Paket oder Datenpaket bezeichnet. Protokolle bzw. die OSI-Schichten haben meist eigene Begriffe, die meist für das gleiche stehen.

OSI-Schicht	Typ (Deutsch)	Typ (Englisch)
Alle Schichten	Paket/Datenpaket	Packet
Anwendung	Nachricht	Message
Transport	Segment	Segment
Vermittlung	Datagramm	Datagramm
Sicherung	Rahmen	Frame
Bitübertragung	Bitfolge / Bitstrom	Bitstream

Ganz allgemein spricht man von Datenpaketen oder nur Paketen. Nimmt man es ganz genau, dann spricht man bei IPv4 von Datagrammen (RFC 891) und bei IPv6 von Paketen (RFC 2460) und bei TCP ist es das Segment.

Netzwerk-Adressen

Innerhalb eines Netzwerks werden spezielle Netzwerk-Adressen verwendet, um Sender und Empfänger eines Datenpakets oder einer Nachricht zu adressieren. Dabei unterscheiden sich Netzwerk-Adressen anhand ihrer Funktion, Anwendungen und Protokoll-Schicht. Das bedeutet, dass eine Kommunikation auf jeder OSI-Schicht mit eigenen Adressen arbeitet.

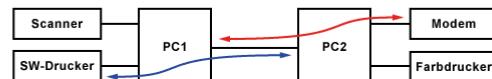
Protokoll	Adresse
Anwendung	URL, Domain, E-Mail-Adresse, ...
Transport	Port
Vermittlung	IPv4-Adresse, IPv6-Adresse
Netzzugang	MAC-Adresse (IEEE)

Internet-Adressen ist ein Überbegriff für Netzwerk-Adressen, die im Internet verwendet werden.

Der Aufbau von Netzwerk-Adressen ist meistens hierarchisch. Das bedeutet, für die einzelnen Bestandteile einer Netzwerk-Adresse gibt es verschiedene verantwortliche Stellen. So stellt man sicher, dass Adressen eindeutig sind und deren Zuteilung nicht zu sehr zentralistisch ist.

- [Netzwerk-Adressen](#)

Ein einfaches Netzwerk: Peer-to-Peer



In einem Peer-to-Peer-Netzwerk ist jeder angeschlossene Computer zu den anderen gleichberechtigt. Jeder Computer stellt den anderen Computern seine Ressourcen zur Verfügung. Ein Peer-to-Peer-Netzwerk eignet sich für bis zu 10 Stationen. Bei mehr Stationen wird es schnell unübersichtlich. Diese Art von Netzwerk ist relativ schnell und kostengünstig aufgebaut. Die Teilnehmer sollten möglichst dicht beieinander stehen.

Einen Netzwerk-Verwalter gibt es nicht. Jeder Netzwerk-Teilnehmer ist für seinen Computer selber verantwortlich. Deshalb muss jeder Netzwerk-Teilnehmer selber bestimmen, welche Ressourcen er freigeben will. Auch die Datensicherung muss von jedem Netzwerk-Teilnehmer selber vorgenommen werden.

Sicherheit in der Netzwerktechnik

Die globale, wie auch lokale, weltweite Vernetzung hat zu einer großen Bedeutung für die Computer- und Netzwerksicherheit geführt. Wo früher vereinzelt kleine Netze ohne Verbindungen nach außen für sich alleine standen, ist heute jedes noch so kleine Netzwerk mit dem Internet verbunden. So ist es möglich, dass aus allen Teilen der Welt unbekannte Personen, ob mit guter oder böser Absicht, eine Verbindung zu jedem Netzwerk herstellen können.

- [Grundlagen der Netzwerk-Sicherheit](#)
- [Sicherheitsrisiken und Sicherheitslücken in der Netzwerktechnik](#)
- [Wie sicher ist ...?](#)
- [Sicherheitskonzepte in der Informations- und Netzwerktechnik](#)
- [Kryptografie](#)
- [Authentifizierung im Netzwerk](#)
- [Verschlüsselung](#)

TCP/IP

TCP/IP ist eine Protokoll-Familie für die Vermittlung und den Transport von Datenpaketen in einem dezentral organisierten Netzwerk. Es wird im LAN (Local Area Network) und im WAN (Wide Area Network) verwendet. Die zentrale Aufgabe von TCP/IP ist dafür Sorgen zu tragen, dass Datenpakete innerhalb eines dezentralen Netzwerks beim Empfänger ankommen. Dafür stellt TCP/IP die folgenden zentralen Funktionen bereit.

- [TCP/IP](#)
- [IPv4 - Internet Protocol Version 4](#)
- [IPv6 - Internet Protocol Version 6](#)
- [TCP - Transmission Control Protocol](#)

Ethernet und WLAN

Ethernet ist eine Familie von Netzwerktechniken, die vorwiegend in lokalen Netzwerken (LAN), aber auch zum Verbinden großer Netzwerke zum Einsatz kommt (WAN). Ethernet wird in der Regel als Synonym für ein lokales Netzwerk verstanden. Im Prinzip werden heute fast alle Vernetzungen im LAN und WAN mit Ethernet-Technik realisiert.

Wireless LAN, kurz WLAN, ist die allgemeine Bezeichnung für ein schnurloses lokales Netzwerk (Wireless Local Area Network). IEEE 802.11 ist ein Standard für eine technische Lösung, die den Aufbau eines Wireless LAN ermöglicht. IEEE 802.11 fand schnell Akzeptanz bei Herstellern und Konsumenten. Deshalb ist es die am weitesten verbreitete drahtlose Technik für ein WLAN.

- [IEEE 802.3 / Ethernet](#)
- [IEEE 802.11 / WLAN](#)

Virtualisierung im Netzwerk

Typischerweise versteht man unter Virtualisierung den Parallelbetrieb von Betriebssystemen auf einer Hardware. Doch auch in der Netzwerktechnik wird virtualisiert. So lassen sich auf einer Netzwerk-Infrastruktur, bestehend aus Verteilkomponenten und Übertragungswegen, mehrere logisch voneinander getrennte Netzwerke betreiben.

- [Virtualisierung im Netzwerk](#)
- [NFV - Network Functions Virtualization](#)

Cloud Computing

Cloud Computing oder Cloud IT umfasst Anwendungen, Daten, Speicherplatz und Rechenleistung aus einem virtuellen Rechenzentrum, das auch Cloud (= Wolke) genannt wird. Die Bezeichnung Cloud wird deshalb verwendet, weil das virtuelle Rechenzentrum aus zusammengeschalteten Computern (Grid) besteht und die Ressource von keinem spezifischen Computer bereitgestellt wird. Die Ressource befindet sich irgendwo in dieser Wolke aus vielen Computern. Eine Anwendung ist keinem Server mehr fest zugeordnet. Die Ressourcen sind dynamisch und bedarfswise abrufbar.

- [Cloud Computing](#)
- [Serverless Computing](#)
- [FaaS - Function as a Service](#)

XIX

IP-Adressen

Jeder Computer im Netzwerk oder im Internet besitzt eine IP-Adresse. Es dient dazu, einen Computer im Netzwerk eindeutig zu identifizieren und ist wie eine Art Telefonnummer, mit der man den Computer anwählen kann. Unterschieden wird dabei zwischen IPv4- und IPv6-Adressen, die neueste Version. Der Grund für die Entwicklung war, dass bei IPv4 lediglich ca. 4,3 Milliarden IP-Adressen vergeben werden können. Durch den rasanten Ausbau des Internets, sind diese mittlerweile aufgebraucht. Das Internet wächst nach wie vor und für die Zukunft ist geplant, weitere Geräte mit IP-Adressen auszustatten, z.B. Kühlschränke die automatisch im Internet bestellen. Daher stand man vor dem Problem, dass man ein neues Internet-Protokoll benötigte, mit dem man wesentlich mehr IP-Adressen vergeben kann. Als Ergebnis wurde IPv6 entwickelt, womit man im Grunde jeder Kaffeemaschine eine eigene IP-Adresse vergeben kann.

Aufbau von IP-Adressen mit IPv4

In dieser Version setzen sich IP-Adressen aus 4 Zahlenblöcken zusammen, z.B. 192.168.178.1 (dezimale Schreibweise). Jeder Zahlenblock kann dabei innerhalb des Bereichs 0 bis 255 sein. Eine IP-Adresse wie beispielsweise 192.168.178.469 gibt es daher nicht, denn der Zahlenblock 469 wäre außerhalb dieses Bereichs. Der Grund dafür ist, dass bei der IP-Adressierung in der Version IPv4 jeder Zahlenblock 8 Bits belegt. 8 Bits sind 1 Byte und damit kann man genau 256 unterschiedliche Zahlen darstellen. Da die 0 mitgezählt wird, ist der erlaubte Bereich im Bereich von 0 bis 255.

IP-Adresse nach IPv4



Adressaufbau in IPv4

In binärer Schreibweise wäre der Bereich zwischen 0 und 11111111. Intern wird im Computer natürlich mit der binären Schreibweise gearbeitet, die dezimale Schreibweise dient den Menschen, z.B. für die Konfiguration. Die einzelnen Zahlenblöcke werden durch Punkte getrennt. Das erleichtert die Betrachtung der IP-Adresse und die Trennung der einzelnen Zahlenblöcke. Intern wird ohne Punkte gearbeitet bzw. sie werden vom Computer nicht beachtet. Insgesamt gerechnet bedeutet das, dass man $256 \times 256 \times 256 \times 256$ Adressen vergeben kann und das wären genau 4,294967296 Milliarden Adressen.

Lokale IP-Adressen

Bei der Vergabe von IP-Adressen wird zwischen internen und externen (Internet) IP-Adressen unterschieden. Hat man z.B. ein Firmennetzwerk, kommuniziert man im Netzwerk über die interne IP-Adresse. Im Internet kommuniziert man dagegen mit der externen IP-Adresse. Dieser wird vom Internet-Provider automatisch vergeben, sobald eine Verbindung ins Internet hergestellt wird. Folgende Bereiche sind für interne IP-Adressen reserviert und können nur in eigenen Netzwerken bzw. Firmennetzwerken genutzt werden.

- 10.0.0.0 bis 10.255.255.255
- 172.16.0.0 bis 172.31.255.255
- 192.168.0.0 bis 192.168.255.255

Adressbereiche für lokale Netzwerke

10.0.0.0 bis 10.255.255.255

172.16.0.0 bis 172.31.255.255

192.168.0.0 bis 192.168.255.255

Lokale Adressbereiche

IP-Adressen innerhalb dieser Bereiche werden im Internet nicht geroutet. Sie dienen zum Aufbau lokaler Netzwerke. Dadurch ist es möglich, dass jede Firma oder Privatperson ein lokales Netzwerk einrichten und den PC's eine IP-Adresse vergeben kann, ohne dass man IP-Adressen vom ohnehin knappen Pool nehmen muss.

Unterteilung lokaler Netzwerke mit der Subnetzmaske

Ein lokales Netzwerk kann in viele kleinere Netzwerke aufgeteilt werden. So kann man einzelne Netzwerkbereiche klar voneinander abgrenzen. Das kann aus verschiedenen Gründen notwendig sein, z.B. um einzelne Abteilungen aus Sicherheitsgründen voneinander zu trennen oder um den Netzwerkverkehr zu beschleunigen. Eine Segmentierung eines Netzwerks erfolgt mit der **Subnetzmaske**, auch Netzmaske, Netzwerkmaske oder auf Englisch subnet mask genannt. Die Subnetzmaske ist ähnlich wie die IP-Adresse. Es besteht aus vier Zahlenblöcken, die mit einem Punkt getrennt sind. Als Werte können sie entweder 0 oder 255 enthalten, z.B. 255.255.255.0.

Zugehörigkeit zum Netzwerk

IP-Adresse 192.168.178.1

Subnetzmaske 255.255.255.0

Definieren, zu welchem Netzwerk der Computer gehört

Zusammensetzung einer IP-Adresse

Zusammen mit der Subnetzmaske bildet die IP-Adresse eine Einheit. Denn, über die Subnetzmaske wird bestimmt, zu welchem Netzwerk der Computer gehört. Ein Beispiel. Einem Computer wurde folgende IP-Adresse und Subnetzmaske zugewiesen:

- IP-Adresse: 192.168.178.125
- Subnetzmaske: 255.255.255.0

Ermitteln der Netzwerkkennung

Die ersten 3 Zahlenblöcke der Subnetzmaske sind mit 255 angegeben. Das würde bedeuten, man nimmt die ersten 3 Zahlenblöcke der IP-Adresse und füllt den Rest mit 0. Der Computer gehört zum Netzwerk 192.168.178.0, auch **Netzwerkkennung** genannt.

Ermitteln der Netzwerk-ID und Host-ID

Die IP-Adresse teilt man auf in **Netzwerkanteil**, das wären in diesem Fall die ersten 3 Zahlenblöcke, und den **Hostanteil**, das wäre der letzte Zahlenblock. Alle Rechner im Netzwerk, bei denen die ersten 3 Zahlenblöcke der IP-Adresse identisch sind, würden zum selben Netzwerk gehören, also innerhalb des Bereichs 192.168.178.1 - 192.168.178.254. Ist die IP-Adresse eines Rechners innerhalb der ersten 3 Zahlenblöcke nicht identisch, zählt der Rechner in diesem Fall nicht zum selben Netzwerk. Ein Rechner mit der IP-Adresse 192.168.128.1 würde z.B. nicht zum selben Netzwerk gehören. Möchte man mit dem Rechner kommunizieren, benötigt man dafür die Hilfe eines [Routers](#), der die Anfrage an das andere Netzwerk weiterleitet.

Subnetzmaske 255.255.255.0



Subnetzmaske: Die ersten 3 Zahlenblöcke

Netzwerkkennung: 192.168.178.0

Netzwerk-ID: 192.168.178.

Host-ID: 125

Zum Netzwerk gehören: 192.168.178.1 – 192.168.178.254

Netzwerkanteil und Hostanteil

Regeln für die Vergabe des Hostanteils

Für die Vergabe der Host-ID gibt es einige Regeln, die es zu beachten gilt. Die Host-ID darf nicht 0 und auch nicht mit dem Wert 255 gefüllt werden. Daher ist in diesem Beispiel auch bewusst geschrieben, alle Rechner mit der IP-Adresse innerhalb des Bereichs 192.168.178.1 - 192.168.178.254 gehören zum selben Netzwerk. Die 0 und 255 fehlen, denn sie dürfen nicht vergeben werden. Das hat folgende Gründe.

Die Netzwerkkennung wird ermittelt, indem man aus der IP-Adresse die Anzahl der Zahlenblöcke nimmt, die auch in der Subnetzmaske mit 255 angegeben wurden und füllt den Rest mit 0. Es ist nicht möglich, auch der Host-ID die 0 zu vergeben. Denn sonst hätte der Rechner dieselbe IP-Adresse wie die Netzwerkkennung.

In einem Netzwerk ist es möglich, alle Rechner gleichzeitig anzusprechen. Diesen Vorgang nennt man Broadcast, was so viel wie Rundsendung bedeutet. Hierfür gibt es die Broadcastadresse, wofür die Netzwerk-ID genommen wird und die restlichen Zahlenblöcke mit 255 gefüllt werden. Diese Adressen sind speziell für die Broadcastadressen reserviert und können nicht gleichzeitig den Rechnern im Netzwerk vergeben werden.

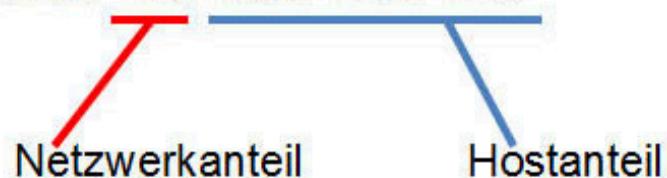
Die Host-ID darf übrigens innerhalb des eigenen Netzwerks nicht identisch mit einer anderen Host-ID sein, da in einem Netzwerk eine IP-Adresse einzigartig sein muss. Sonst erhält man eine eingeschränkte Konnektivität. Das ist eine häufige Fehlerquelle, insbesondere wenn man die [IP-Adresse ändert](#) und manuell vergibt. Daher sollte man insbesondere bei großen Netzwerken [DHCP-Server](#) einsetzen, die die Verwaltung der IP-Adressen automatisch übernehmen.

Adressklassen in Netzwerken

Damit man IP-Adressen strukturieren kann, wurden sogenannte Adressklassen geschaffen. Unterschieden wird dabei zwischen Klasse A - Klasse E. Somit kann man, je nach Größe des Unternehmens, unterschiedliche Netzwerke aufbauen. Welche Zahl der erste Zahlenblock hat und welche Subnetzmaske verwendet wird, entscheidet über die Adressklasse des Netzwerks.

Subnetzmaske 255.0.0.0

IP-Adresse 10.168.178.125



Klasse A Netzwerk

Subnetzmaske: Der erste Zahlenblock

Netzwerk kennung: 10.0.0.0

Netzwerk-ID: 10.

Host-ID: 168.178.125

Zum Netzwerk gehören: 10.0.0.1 – 10.255.255.254

IP-Adresse eines Klasse A Netzwerks

Die IP-Adresse 10.168.178.125 mit der Subnetzmaske 255.0.0.0 gehört zu einem Klasse A Netzwerk. 128.168.178.125 mit der Subnetzmaske 255.255.0.0 würde zu einem Klasse B Netzwerk gehören. 192.168.178.125 mit der Subnetzmaske 255.255.255.0 gehört zu einem Klasse C Netzwerk. 224.168.178.125 mit der Subnetzmaske 255.255.255.255 wäre ein Klasse D Netzwerk, 240.168.178.125 mit der Subnetzmaske 255.255.255.255 wäre ein Klasse E Netzwerk. Wie die genauen Regeln definiert sind, wird unter [Adressklassen_in_Netzwerken](#) beschrieben.

XX

Package java.net

Provides the classes for implementing networking applications.

See: [Description](#)

Interface Summary	
Interface	Description
ContentHandlerFactory	This interface defines a factory for content handlers.
CookiePolicy	CookiePolicy implementations decide which cookies should be accepted and which should be rejected.
CookieStore	A CookieStore object represents a storage for cookie.
DatagramSocketImplFactory	This interface defines a factory for datagram socket implementations.
FileNameMap	A simple interface which provides a mechanism to map between a file name and a MIME type string.
ProtocolFamily	Represents a family of communication protocols.
SocketImplFactory	This interface defines a factory for socket implementations.
SocketOption<T>	A socket option associated with a socket.
SocketOptions	Interface of methods to get/set socket options.
URLStreamHandlerFactory	This interface defines a factory for URL stream protocol handlers.

Class Summary	
Class	Description
Authenticator	The class Authenticator represents an object that knows how to obtain authentication for a network connection.
CacheRequest	Represents channels for storing resources in the ResponseCache.
CacheResponse	Represent channels for retrieving resources from the ResponseCache.
ContentHandler	The abstract class ContentHandler is the superclass of all classes that read an Object from a URLConnection.
CookieHandler	A CookieHandler object provides a callback mechanism to hook up a HTTP state management policy implementation into the HTTP protocol handler.
CookieManager	CookieManager provides a concrete implementation of CookieHandler , which separates the storage of cookies from the policy surrounding accepting and rejecting cookies.
DatagramPacket	This class represents a datagram packet.
DatagramSocket	This class represents a socket for sending and receiving datagram packets.
DatagramSocketImpl	Abstract datagram and multicast socket implementation base class.
HttpCookie	An HttpCookie object represents an http cookie, which carries state information between server and user agent.
URLConnection	A URLConnection with support for HTTP-specific features.
IDN	Provides methods to convert Internationalized domain names (IDNs) between a normal Unicode representation and an ASCII Compatible Encoding (ACE) representation.
Inet4Address	This class represents an Internet Protocol version 4 (IPv4) address.
Inet6Address	This class represents an Internet Protocol version 6 (IPv6) address.
InetAddress	This class represents an Internet Protocol (IP) address.
InetSocketAddress	This class implements an IP Socket Address (IP address + port number) It can also be a pair (hostname + port number), in which case an attempt will be made to resolve the hostname.
InterfaceAddress	This class represents a Network Interface address.
JarURLConnection	A URL Connection to a Java ARChive (JAR) file or an entry in a JAR file.
MulticastSocket	The multicast datagram socket class is useful for sending and receiving IP multicast packets.
NetPermission	This class is for various network permissions.
NetworkInterface	This class represents a Network Interface made up of a name, and a list of IP addresses assigned to this interface.

PasswordAuthentication	The class PasswordAuthentication is a data holder that is used by Authenticator.
Proxy	This class represents a proxy setting, typically a type (http, socks) and a socket address.
ProxySelector	Selects the proxy server to use, if any, when connecting to the network resource referenced by a URL.
ResponseCache	Represents implementations of URLConnection caches.
SecureCacheResponse	Represents a cache response originally retrieved through secure means, such as TLS.
ServerSocket	This class implements server sockets.
Socket	This class implements client sockets (also called just "sockets").
SocketAddress	This class represents a Socket Address with no protocol attachment.
SocketImpl	The abstract class <code>SocketImpl</code> is a common superclass of all classes that actually implement sockets.
SocketPermission	This class represents access to a network via sockets.
StandardSocketOptions	Defines the <i>standard</i> socket options.
URI	Represents a Uniform Resource Identifier (URI) reference.
URL	Class URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web.
URLConnection	This class loader is used to load classes and resources from a search path of URLs referring to both JAR files and directories.
URLConnection	The abstract class <code>URLConnection</code> is the superclass of all classes that represent a communications link between the application and a URL.
URLEncoder	Utility class for HTML form decoding.
URLEncoder	Utility class for HTML form encoding.
URLStreamHandler	The abstract class <code>URLStreamHandler</code> is the common superclass for all stream protocol handlers.

Enum Summary	
Enum	Description
Authenticator.RequestorType	The type of the entity requesting authentication.
Proxy.Type	Represents the proxy type.
StandardProtocolFamily	Defines the standard families of communication protocols.

Exception Summary	
Exception	Description
BindException	Signals that an error occurred while attempting to bind a socket to a local address and port.
ConnectException	Signals that an error occurred while attempting to connect a socket to a remote address and port.
HttpRetryException	Thrown to indicate that a HTTP request needs to be retried but cannot be retried automatically, due to streaming mode being enabled.
MalformedURLException	Thrown to indicate that a malformed URL has occurred.
NoRouteToHostException	Signals that an error occurred while attempting to connect a socket to a remote address and port.
PortUnreachableException	Signals that an ICMP Port Unreachable message has been received on a connected datagram.
ProtocolException	Thrown to indicate that there is an error in the underlying protocol, such as a TCP error.
SocketException	Thrown to indicate that there is an error creating or accessing a Socket.
SocketTimeoutException	Signals that a timeout has occurred on a socket read or accept.
UnknownHostException	Thrown to indicate that the IP address of a host could not be determined.
UnknownServiceException	Thrown to indicate that an unknown service exception has occurred.
URISyntaxException	Checked exception thrown to indicate that a string could not be parsed as a URI reference.

Package java.net Description

Provides the classes for implementing networking applications.

The java.net package can be roughly divided in two sections:

- A *Low Level API*, which deals with the following abstractions:
 - Addresses, which are networking identifiers, like IP addresses.
 - Sockets, which are basic bidirectional data communication mechanisms.
 - Interfaces, which describe network interfaces.
- A *High Level API*, which deals with the following abstractions:
 - URIs, which represent Universal Resource Identifiers.
 - URLs, which represent Universal Resource Locators.
 - Connections, which represents connections to the resource pointed to by URLs.

Addresses

Addresses are used throughout the java.net APIs as either host identifiers, or socket endpoint identifiers.

The `InetAddress` class is the abstraction representing an IP (Internet Protocol) address. It has two subclasses:

- `Inet4Address` for IPv4 addresses.
- `Inet6Address` for IPv6 addresses.

But, in most cases, there is no need to deal directly with the subclasses, as the `InetAddress` abstraction should cover most of the needed functionality.

About IPv6

Not all systems have support for the IPv6 protocol, and while the Java networking stack will attempt to detect it and use it transparently when available, it is also possible to disable its use with a system property. In the case where IPv6 is not available, or explicitly disabled, `Inet6Address` are not valid arguments for most networking operations any more. While methods like `InetAddress.getByName(java.lang.String)` are guaranteed not to return an `Inet6Address` when looking up host names, it is possible, by passing literals, to create such an object. In which case, most methods, when called with an `Inet6Address` will throw an Exception.

Sockets

Sockets are means to establish a communication link between machines over the network. The java.net package provides 4 kinds of Sockets:

- `Socket` is a TCP client API, and will typically be used to connect to a remote host.
- `ServerSocket` is a TCP server API, and will typically accept connections from client sockets.
- `DatagramSocket` is a UDP endpoint API and is used to send and receive datagram packets.
- `MulticastSocket` is a subclass of `DatagramSocket` used when dealing with multicast groups.

Sending and receiving with TCP sockets is done through `InputStreams` and `OutputStreams` which can be obtained via the `Socket.getInputStream()` and `Socket.getOutputStream()` methods.

Interfaces

The `NetworkInterface` class provides APIs to browse and query all the networking interfaces (e.g. ethernet connection or PPP endpoint) of the local machine. It is through that class that you can check if any of the local interfaces is configured to support IPv6.

High level API

A number of classes in the java.net package do provide for a much higher level of abstraction and allow for easy access to resources on the network. The classes are:

- `URI` is the class representing a Universal Resource Identifier, as specified in RFC 2396. As the name indicates, this is just an Identifier and doesn't provide directly the means to access the resource.
- `URL` is the class representing a Universal Resource Locator, which is both an older concept for URIs and a means to access the resources.
- `URLConnection` is created from a URL, and is the communication link used to access the resource pointed by the URL. This abstract class will delegate most of the work to the underlying protocol handlers like http or ftp.
- `HttpURLConnection` is a subclass of `URLConnection` and provides some additional functionalities specific to the HTTP protocol.

The recommended usage is to use `URI` to identify resources, then convert it into a `URL` when it is time to access the resource. From that URL, you can either get the `URLConnection` for fine control, or get directly the `InputStream`.

Here is an example:

```
URI uri = new URI("http://java.sun.com/");
URL url = uri.toURL();
InputStream in = url.openStream();
```

Protocol Handlers

As mentioned, `URL` and `URLConnection` rely on protocol handlers which must be present, otherwise an Exception is thrown. This is the major difference with `URIs` which only identify resources, and therefore don't need to have access to the protocol handler. So, while it is possible to create an `URI` with any kind of protocol scheme (e.g. `myproto://myhost.mydomain/resource/`), a similar `URL` will try to instantiate the handler for the specified protocol; if it doesn't exist an exception will be thrown.

By default the protocol handlers are loaded dynamically from the default location. It is, however, possible to add to the search path by setting the `java.protocol.handler.pkgs` system property. For instance if it is set to `myapp.protocols`, then the `URL` code will try, in the case of http, first to load `myapp.protocols.http.Handler`, then, if this fails, `http.Handler` from the default location.

Note that the `Handler` class **has to** be a subclass of the abstract class `URLStreamHandler`.

Kommunikation in Netzwerken

Das Internet ist eine erstaunliche Erfindung mit weitreichenden Folgen – doch obwohl wir es ständig benutzen, versteht fast niemand, wie es eigentlich funktioniert. Also: Wie wird der weltweite Austausch von Informationen zwischen Digitalgeräten ermöglicht?

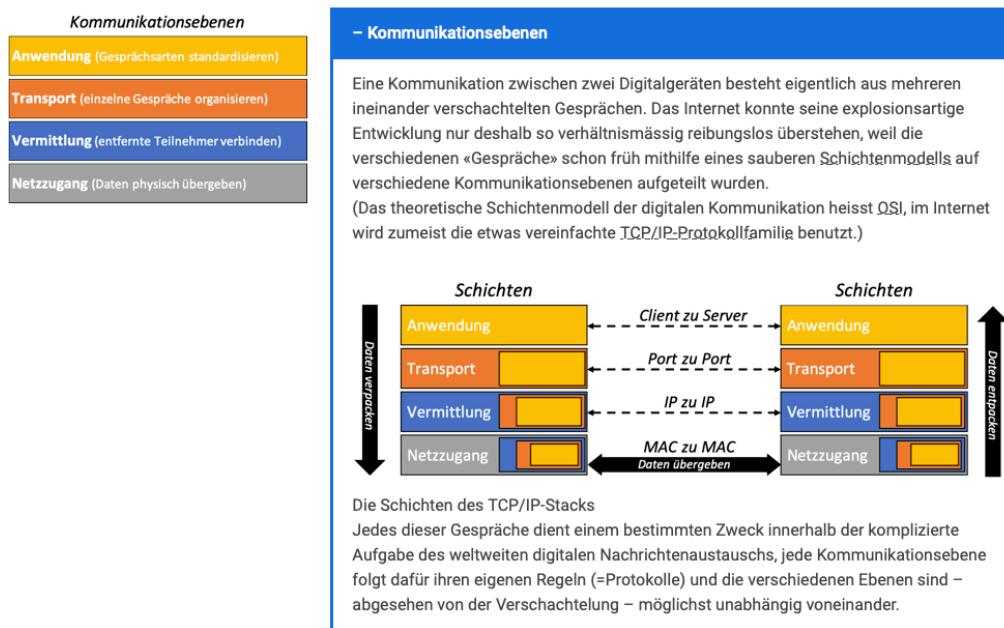
Statt von Mensch zu Mensch miteinander zu sprechen, kommunizieren in Netzwerken Geräte miteinander. Damit das reibungslos funktioniert, gibt es verschiedenen Schichten dieser Kommunikation, die unterschiedliche Aufgaben haben.

MATERIALIEN

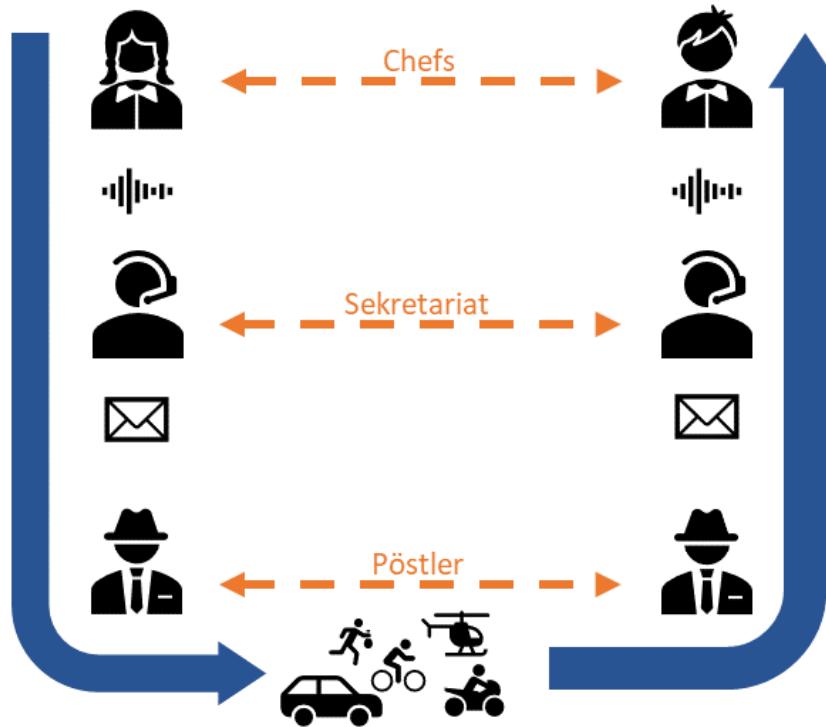
- ❖ Filius
- ❖ IPs und Netzmasken
- ❖ Netzwerke-mit-Filius-simuliert
- ❖ Subnetze.docx
- ❖ Subnetze.docx.pdf
- ❖ Vernetzung.pptx.pdf

Kommunikation unter Geräten

Die Kommunikation zwischen Digitalgeräten ist auf so allgemeine Art geregelt, dass es für die beteiligten Endgeräte eigentlich keinen Unterschied macht, ob sie direkt miteinander verbunden sind, oder ob sie ihre Nachrichten über ein weltweites Netzwerk (=Internet) austauschen. Die Grundlage für beides ist ein Schichtenmodell, das die komplexe Aufgabe der Nachrichtenübermittlung auf verschiedenen Ebenen, bzw. Schichten verteilt:



Analogie



Zwei Direktoren wollen miteinander ein Geschäft machen – der Ablauf könnte in etwa so aussehen, wie auf der Skizze in Form eines Schichtenmodells dargestellt: Auf der obersten Ebene geht es um den Inhalt, auf der mittleren Ebene geht es um die Form und auf der untersten Ebene geht es um die Übermittlung. An diesem Beispiel zeigt sich auch der grosse Vorteil von Schichtenmodellen: Für den Inhalt macht es keinen Unterschied, ob der Brief letztendlich mit dem Auto oder mit dem Fahrrad transportiert wird.

Aus dieser Regelung für die Kommunikation zwischen Digitalgeräten ergibt sich auch die Form der übermittelten Nachrichten:

tarife.at

Top Tarifvergleiche
unabhängig & kostenlos Hilfe benötigt?
Frag uns was! Speedtest
Jetzt prüfen

Handytarife Handys Internet für Zuhause Internet Cube Mobiles Internet Internet + TV Fernsehen Weiteres ▾

Protokoll

Protokolle, auch Netzwerkprotokolle genannt, sind spezielle Regeln nach denen die Kommunikation zwischen Geräten funktioniert. Durch ein Protokoll wird beispielsweise bereits definiert, wie der Aufbau von Nachrichten auszusehen hat oder wie sie übermittelt werden können. Wichtige Protokolle sind zum Beispiel HTTP für Webseiten, oder NFS für Netzwerkf freigaben.

Sie können ganz unterschiedlichen Nutzen haben: Etwa dass Rechner sich verbinden können, Datenpakete übertragen werden, das Dateiformat festgelegt wird, die Übertragung vollständig und in der richtigen Reihenfolge passiert, Unbefugte keinen Zugriff erhalten oder ähnliches.

Protokollgruppen

Es gibt mehrere Gruppen, in die Protokolle unterschieden werden:

Protokolle nach Nutzeranzahl

Es gibt Protokolle, die Daten an einzelne Empfänger (**Unicast**-Protokolle), mehrere Empfänger (**Multicast**-Protokolle) oder alle Stationen (**Broadcast**-Protokolle) übermitteln.

Protokolle nach Datenstrom

Bei **Simplex**-Protokollen ist eine Station nur Sender und die andere Station nur Empfänger. **Voll duplex**-Protokolle hingegen ermöglichen den gleichzeitigen Datenaustausch in beide Richtungen.



Inhaltsverzeichnis

Protokollgruppen

- Protokolle nach Nutzeranzahl
- Protokolle nach Datenstrom
- Weitere Protokolle
- Protokollformen
- Strenge Protokoll

Weitere Protokolle

Es gibt viele weitere Unterschiedsmöglichkeiten, etwa nach Hierarchie, wie bei der **Server-Client**-Kommunikation oder der gleichrangigen **Peer-to-Peer**-Kommunikation. Oder ob sie verbindungslos oder verbindungsorientiert sind. Auch ob die Daten synchron oder asynchron übertragen werden.

Protokollformen

Bekannte Protokolle sind:

- **IP:** Das Internet Protokol ordnet Datenpakete der richtigen Adresse zu.
- **TCP:** Das Transmission Control Protocol definiert die Art der Übertragung.
- **HTTP:** Das Hypertext Transfer Protocol ist ein Dateiübertragungsprotokoll, das hauptsätzlich dafür eingesetzt wird um Webseiten zu laden.
- **FTP:** Das File Transfer Protocol ist ein Dateiübertragungsprotokoll, das hauptsätzlich dafür eingesetzt wird, um Daten zwischen Servern oder Server und Client zu übertragen.
- **DHCP:** Das Dynamic **Host** Configuration Protokoll ist ein Kommunikationsprotokoll, das die Zuweisung der Netzwerkkonfiguration an Clients durch einen Server ermöglicht.
- **BGP:** Das Border Gateway Protocol ist ein Routingprotokoll, das autonome Systeme miteinander verbindet.

Strenge Protokoll

Das Wort Protokoll wird nicht nur in der Computertechnik verwendet. Es wird auch in der Forschung, der Medizin, in verschiedenen Technikbereichen (wie zum Beispiel der Automobilindustrie), in der Rechtssprechung und ganz allgemein für Niederschrift verwendet.

Dabei beschreibt es überall eine Aufzeichnung, zu welchem Zeitpunkt oder in welcher Reihenfolge welcher Vorgang durch wen oder durch was veranlasst wurde oder wird.

Protokoll stammt aus dem Griechischen von protos – erster und kolla – Leim, weil es ursprünglich das Blatt war, das amtlichen Papyrusrollen vorgeleimt wurde. Dieses enthielt die bibliographischen Daten und erleichterte die Zuordnung, ähnlich einem heutigen Aktendeckel.

XXIII



Startseite Themen News Forum Online-Shop

TCP - Transmission Control Protocol

Das Transmission Control Protocol, kurz TCP, ist Teil der Protokollfamilie TCP/IP. TCP ist ein verbindungsorientiertes Protokoll und soll maßgeblich Datenverluste verhindern, Dateien und Datenströme aufteilen und Datenpakete den Anwendungen zuordnen können.

Videokonferenz-Server im eigenen Netzwerk betreiben



- Eigener Videokonferenz-Server auf Basis von Jitsi und WebRTC
- Sicherer und Datenschutz-konformer durch den **Eigenbetrieb**
- **Unabhängig** von externen Diensten
- Externe Teilnehmer einladen
- In jedem **Webbrowser** einfach zu bedienen
- Mit Jitsi Meet ist eine **Smartphone-App** verfügbar

Bestellen Sie Ihre TrutzBox mit integriertem Videokonferenz-Server jetzt mit dem Gutschein-Code "elko50" und sparen Sie dabei **50 Euro**.

[Mehr über den Videokonferenz-Server](#)

Das Transmission Control Protocol (TCP) im TCP/IP-Protokollstapel

Schicht	Dienste / Protokolle / Anwendungen
Anwendung	HTTP IMAP DNS SNMP
Transport	TCP UDP
Internet	IP (IPv4 / IPv6)
Netzzugang	Ethernet, ...

Für die Anwendungen ist TCP transparent. Die Anwendungen übergeben ihren Datenstrom an den TCP/IP-Stack und nehmen ihn von dort auch wieder entgegen. Mit der für die Übertragung nötige TCP-Paketstruktur sowie die Parameter der ausgehandelten Verbindung haben die Anwendungen nichts zu tun.

Aufgaben und Funktionen von TCP

- Segmentierung (Data Segmenting): Dateien oder Datenstrom in Segmente teilen, Reihenfolge der Segmente wieder herstellen und zu Dateien oder einem Datenstrom zusammensetzen
- Verbindungsmanagement (Connection Establishment and Termination): Verbindungsaufbau und Verbindungsabbau
- Fehlerbehandlung (Error Detection): Bestätigung von Datenpaketen und Zeitüberwachung
- Flusssteuerung (Flow Control): Dynamische Auslastung der Übertragungsstrecke
- Anwendungsunterstützung (Application Support): Adressierung spezifischer Anwendungen und Verbindungen durch Port-Nummern

Folge uns



Das Buch zu dieser Webseite

Netzwerkechnik-Fibel



Kundenmeinung:


"Da Bücher, die einfach zu verstehen sind, schwer zu finden sind, habe ich die Investition in die Netzwerkechnik-Fibel nicht bereut."

[Netzwerkechnik-Fibel jetzt bestellen!](#)

Das Buch zu dieser Webseite

Kommunikationstechnik-Fibel



Segmentierung (Data Segmenting)

Eine Funktion von TCP besteht darin, den von den Anwendungen kommenden Datenstrom in Datenpakete bzw. Segmente aufzuteilen (Segmentierung) und beim Empfang wieder zusammenzusetzen. Die Segmente werden mit einem Header versehen, in dem Steuer- und Kontrollinformationen enthalten sind. Danach werden die Segmente an das Internet Protocol (IP) übergeben. Da beim IP-Routing die Datenpakete unterschiedliche Wege gehen können, entstehen unter Umständen zeitliche Verzögerungen, die dazu führen, dass die Datenpakete beim Empfänger in einer anderen Reihenfolge eingehen, als sie ursprünglich hatten. Deshalb werden die Segmente beim Empfänger auch wieder in die richtige Reihenfolge gebracht und erst dann an die adressierte Anwendung übergeben. Dazu werden die Segmente mit einer fortlaufenden Sequenznummer versehen (Sequenzierung).

Verbindungsmanagement (Connection Establishment and Termination)

Als verbindungsorientiertes Protokoll ist TCP für den Verbindungsaufbau und Verbindungsabbau zwischen zwei Stationen einer Ende-zu-Ende-Kommunikation zuständig. Durch TCP stehen Sender und Empfänger ständig in Kontakt zueinander.

- [TCP-Kommunikation \(Verbindungsaufbau/Verbindungsabbau\)](#)

Fehlerbehandlung (Error Detection)

Obwohl es sich eher um eine virtuelle Verbindung handelt, werden während der Datenübertragung ständig Kontrollmeldungen ausgetauscht. Der Empfänger bestätigt dem Sender jedes empfangene Datenpaket. Trifft keine Bestätigung beim Absender ein, wird das Paket noch mal verschickt. Da es bei Übertragungsproblemen zu doppelten Datenpaketen und Quittierungen kommen kann, werden alle TCP-Pakete und TCP-Meldungen mit einer fortlaufenden Sequenznummer gekennzeichnet. So sind Sender und Empfänger in der Lage, die Reihenfolge und Zuordnung der Datenpakete und Meldungen zu erkennen.

- [TCP-Kommunikation \(Fehlerbehandlung\)](#)

Flusssteuerung (Flow Control)

Bei einer paketorientierten Übertragung ohne feste zeitliche Zuordnung und ohne Kenntnis des Übertragungswegs erhält das Transport-Protokoll vom Übertragungssystem keine Information über die verfügbare Bandbreite. Mit der Flusssteuerung werden beliebig langsame oder schnelle Übertragungsstrecken dynamisch auszulasten und auch auf unerwartete Engpässe und Verzögerungen reagiert.

- [TCP-Kommunikation \(Flusssteuerung\)](#)

Anwendungsunterstützung (Application Support)

TCP- und UDP-Ports sind eine Software-Abstraktion, um Kommunikationsverbindungen voneinander unterscheiden zu können. Ähnlich wie IP-Adressen Rechner in Netzwerken adressieren, adressieren Ports spezifische Anwendungen oder Verbindungen, die auf einem Rechner laufen.

- [TCP- und UDP-Ports](#)

Der kleine Bruder: UDP - User Datagram Protocol

Neben dem verbindungsorientierten TCP gibt es auch das verbindungslose und unsichere UDP. Das User Datagram Protocol (UDP) ist ebenso auf der Schicht 4, der Transportschicht, des OSI-Schichtenmodells angeordnet. Es hat die selbe Aufgabe wie TCP, nur das ihm nahezu alle Kontrollfunktionen fehlen und dadurch schlanker daher kommt und einfacher zu verarbeiten ist.

- [Mehr Informationen über UDP](#)

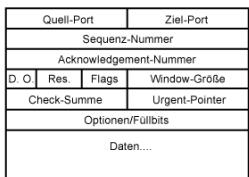
Kundenmeinung:



"Die Kommunikationstechnik-Fibel ist sehr informativ und verständlich. Genau das habe ich schon seit langem gesucht. Endlich mal ein Buch, das kurz und bündig die moderne Informationstechnik beleuchtet."

[Kommunikationstechnik-Fibel jetzt bestellen!](#)

Aufbau des TCP-Headers



Jedem Datenpaket, das TCP verschickt, wird ein Header vorangestellt, der die folgenden Daten enthält:

- Sender-Port
- Empfänger-Port
- Paket-Reihenfolge (Nummer)
- Prüfsumme
- Quittierungsnummer
- Flags

Aufbau des TCP-Headers TCP-Pakete setzen sich aus dem Header-Bereich und dem Daten-Bereich zusammen. Im Header sind alle Informationen enthalten, die für eine gesicherte TCP-Verbindung wichtig sind. Der TCP-Header ist in mehrere 32-Bit-Blöcke aufgeteilt. Mindestens enthält der Header 5 solcher Blöcke. Somit hat ein TCP-Header eine Länge von mindestens 20 Byte.

Bedeutung der Felder im TCP-Header

Feldinhalt	Bit	Beschreibung
Quell-Port (Source-Port)	16	Hier steht der Quell-Port, von der die Anwendung das TCP-Paket verschickt. Bei einer Stellenanzahl von 16 Bit beträgt der höchste Port 65.535.
Ziel-Port (Destination-Port)	16	Hier steht der Ziel-Port, über welchen das TCP-Paket der Anwendung zugestellt wird. Bei einer Stellenanzahl von 16 Bit beträgt der höchste Port 65.535.
Sequenz-Nummer	32	Bei jeder TCP-Verbindung werden Nummern zwischen den Kommunikationspartnern ausgetauscht. Während der Verbindung werden diese Nummern verwendet um die TCP-Pakete eindeutig zu identifizieren.
Acknowledgement-Nummer	32	Alle Datenpakete werden bestätigt. Dazu dient das ACK-Flag und die Acknowledgement-Nummer, die sich aus der Sequenz-Nummer und der Anzahl von empfangenen Bytes errechnet. Damit kann der Sender feststellen, ob die Daten beim Empfänger vollständig angekommen sind.
Data Offset	4	Hier steht die Anzahl der 32-Bit-Blöcke des TCP-Headers. Die Mindestmenge beträgt 5.
Reserviert	6	Dieser Bereich ist auf 000000 gesetzt und für Erweiterungen des TCP-Headers gedacht.
Flags	6	Kennzeichnung bestimmter für die Kommunikation und Weiterverarbeitung der Daten wichtiger Zustände (URG, ACK, PSH, RST, SYN, FIN).
Window-Größe	16	Der Empfänger sendet dem Sender in diesem Feld die Anzahl an Daten, die der Sender senden darf. Dadurch wird das Überlaufen des Empfangspuffers beim Empfänger verhindert. Den Vorgang nennt man Windowing und dient der Datenflusssteuerung.
Check-Summe	16	Dieses Feld dient der Kontrolle von Header- und Datenbereich.
Urgent-Pointer	16	Zusammen mit der Sequenz-Nummer gibt dieser Wert die genaue Position der Daten im Datenstrom an. Der Wert ist nur gültig, wenn das URG-Flag gesetzt ist.
Optionen/Füllbits (Options/Padding), jeweils 32 Bit lang		Dieses Feld beinhaltet optionale Informationen. Um die 32-Bit-Grenze einzuhalten wird das Options-Feld mit Nullen aufgefüllt.

tarife.at

Top Tarifvergleiche
unabhängig & kostenlos

Hilfe benötigt?
Frag uns was!

Speedtest
Jetzt prüfen

Handytarife Handys Internet für Zuhause Internet Cube Mobiles Internet Internet + TV Fernsehen Weiteres ▾

UDP

UDP steht für User Datagram Protocol. Es ist ein Kommunikationsprotokoll, das auf dem Internet Protokoll aufbaut und kurze Datenpakete sendet, die auch Datagramme genannt werden. UDP ist ein verbindungsloses und ungesichertes **Netzwerkprotokoll**. Es dient vor allem dem Transport von Daten als auch dem Multiplexen von Verbindungen.

Nutzen von UDP

UDP ist das ideale Protokoll für Netzwerkanwendungen, bei denen es auf eine möglichst geringe **Latenz** ankommt. Es ermöglicht fehlertolerante Verbindungen bei niedrigem Ping. Das betrifft zum Beispiel Spiele oder **Videotelefonie**, wo ein geringfügiger Datenverlust relativ egal ist.

Außerdem bietet UDP die Option Prüfsummen zu erstellen. Damit lässt sich sicherstellen, dass bestimmte Daten korrekt angekommen sind. Es bietet zusätzlich auch Portnummern, um zwischen verschiedenen Benutzeranfragen oder Anwendungen einfacher unterscheiden zu können.

TCP oder UDP?

UDP ist eine Alternative zu TCP (Transmission Control Protocol). Sowohl UDP als auch TCP basieren auf dem Internet Protocol (IP) und werden auch UDP/IP oder TCP/IP genannt. TCP verzögert beim Versenden der Datenpakete die Latenz, während UDP eine geringere **Bandbreite** benötigt.

JETZT HANDYTARIFE FINDEN!

Datenvolumen 10 GB

Minuten 500

SMS 500

Vergleich Starten

Inhaltsverzeichnis

- Nutzen von UDP**
- TCP oder UDP?
- Wo agiert UDP?

Der Nachteil von UDP ist, dass Pakete verloren gehen können oder in der falschen Reihenfolge eintreffen. Bei Anwendungen, die eine erneute Übertragung fehlender Pakete anfordern, oder die Pakete richtig einordnen können, stellt dies jedoch kein Problem dar.

Wo agiert UDP?

UDP gehört zur Transportschicht, dem OSI Layer 4. OSI bietet verschiedene Schichten und ist ein Standard der Netzwerkarchitektur. Seit 1980 ist UDP im RFC 768 definiert.

Das RFC (Request for Comments, „Bitte um Kommentare“) ist eine Reihe von technischen Dokumenten, die vor allem Internetstandards definiert. Das RFC wird durch die Internet Engineering Task Force (IETF) verwaltet.

UDP arbeitet mit höheren Protokollen zusammen, wie zum Beispiel [DNS](#) - Anfragen, Internet Control Message Protocol (ICMP), Trivial File Transfer Protocol (TFTP), Real Time [Streaming](#) Protocol (RTSP) oder dem Simple Network Protocol (SNP).

(Artikel veröffentlicht am: 13.10.2020 . Letzte Aktualisierung am: 29.10.2020)

XXV

[Definitionen](#) > Was ist das OSI-Modell?

Definition

Was ist das OSI-Modell?

01.08.2018 | Autor / Redakteur: [Dipl.-Ing. \(FH\) Stefan Luber](#) / [Dipl.-Ing. \(FH\) Andreas Donner](#)

Beim OSI-Modell, oft auch als ISO/OSI-Schichtenmodell bezeichnet, handelt es sich um ein Referenzmodell, mit dem sich die Kommunikation zwischen Systemen beschreiben und definieren lässt. Das Referenzmodell besitzt sieben einzelne Schichten (Layer) mit jeweils klar voneinander abgegrenzten Aufgaben.



(© aga7ta - Fotolia)

Die Abkürzung OSI steht für den englischen Fachbegriff Open Systems Interconnection Model. Die Entwicklung des OSI-Modells begann bereits in den 70er Jahren. Die Veröffentlichung erfolgte von Seiten der International Telecommunication Union (ITU) und der International Organization for Standardization (ISO).

Das OSI-Modell wird daher auch gerne als ISO/OSI-Schichtenmodell bezeichnet. Zielsetzung bei der Definition des ISO/OSI-Standards war es, ein Referenzmodell zu schaffen, das die Kommunikation verschiedener technischer Systeme über unterschiedliche Medien und Technologien ermöglicht und Kompatibilitäten bereitstellt. Um dieses Ziel zu erreichen, verwendet das OSI-Modell insgesamt sieben verschiedene Schichten (Layer), die hierarchisch aufeinander aufbauen.

In jeder einzelnen Schicht werden genau definierte Aufgaben ausgeführt. Die Schnittstellen zur jeweils darüber- und darunterliegenden Schicht sind exakt beschrieben. Dadurch lassen sich Zwischenschichten austauschen, ohne dass die anderen Layer davon betroffen sind. Netzwerkprotokolle, Anwendungsprotokolle oder Übertragungsmedien werden dank des Schichtenmodells prinzipiell beliebig ersetzbar. Jede Schicht bietet der direkt über ihr liegenden Schicht Dienste zur Nutzung an. Um diese Dienste zur Verfügung zu stellen, verwendet die Schicht die Dienste des unter ihr liegenden Layers und führt die Aufgaben des eigenen Layers aus.



In den einzelnen Schichten müssen eine Vielzahl verschiedener Aufgaben bewältigt werden, die für die Sicherheit, die Zuverlässigkeit und die Performance der Kommunikationsverbindung sorgen. Kommunizieren zwei Systeme miteinander, werden alle sieben Schichten des OSI-

Modells mindestens zweimal durchlaufen, da sowohl der Sender als auch der Empfänger das Schichtenmodell zu berücksichtigen hat.

XXVI

[Definitionen](#) > Was ist Multithreading?

Gleichzeitige Ausführung von Threads

Was ist Multithreading?

29.06.2021 | Autor / Redakteur: [Dipl.-Ing. \(FH\) Stefan Luber](#) / [Dr. Jürgen Ehneß](#)

Unter den Begriff des Multithreadings fallen Verfahren und Techniken, mit denen sich mehrere Ausführungsstränge von Prozessen oder Anwendungen gleichzeitig oder quasi-gleichzeitig ausführen lassen. Die Verfahren und Techniken können Soft- oder Hardware-basiert arbeiten. Ziel des Multithreadings ist es, die Ausführungsgeschwindigkeit der Anwendungen oder allgemein die Rechengeschwindigkeit zu erhöhen. Sowohl Intel als auch AMD bieten Prozessoren mit Multithreading-Techniken an.



Die wichtigsten IT-Fachbegriffe verständlich erklärt.

(Bild: © aga7ta - Fotolia)

Multithreading ist ein Begriff aus der Informatik. Es handelt sich um Verfahren oder Techniken, mit denen sich mehrere Threads eines Prozesses gleichzeitig oder quasi-gleichzeitig (pseudo-gleichzeitig) ausführen lassen. Als Threads werden einzelne Ausführungsstränge von Prozessen bezeichnet. Die Multithreading-Verfahren oder -Techniken arbeiten Soft- oder Hardware-basiert.

Ziel des Multithreadings ist es, die Ausführungsgeschwindigkeit der Anwendungen zu erhöhen oder allgemein die Leistungsfähigkeit und Rechengeschwindigkeit der Prozessoren zu verbessern. Eine Anwendung, die aus mehreren Prozessen und Threads besteht, arbeitet dank Multithreading und der parallelen Abarbeitung mehrerer Aufgaben schneller. Die Anwendung muss dafür vorbereitet sein und sich in sinnvolle, gleichzeitig zu bearbeitende Threads aufteilen lassen. Zudem ist ein gewisses Zusammenspiel zwischen Hard- und Software notwendig.

Anzeige

Der Stellenmarkt für IT-Profis.

Die großen Prozessorhersteller Intel und AMD setzen Multithreading-Technologien ein. Sie basieren auf dem Simultaneous Multithreading (SMT). Bei SMT verhält sich ein einzelner Prozessor wie mehrere logische Prozessoren. Intel nennt die Technologie „Hyper-Threading“.

Die verschiedenen Arten des Multithreadings

Grundsätzlich kann zwischen den Software-basierten und Hardware-basierten Multithreading-Technologien unterschieden werden. Software-basiertes Multithreading funktioniert mit einem einzigen Prozessorkern ohne besondere Hardware-Unterstützung. Für den Prozessorkern selbst ist die parallele Abarbeitung mehrerer Threads nicht erkennbar. Die Software (die Anwendung oder das Betriebssystem) regelt die scheinbar gleichzeitige Ausführung mehrerer Threads über Multiplexen und eine intelligente Zuteilung der einzelnen Ausführungsstränge an den Prozessor.