Technische Hochschule **Rosenheim**

Modul - IT Systems (IT)

Bachelor Programme AAI

09 - Lecture: Internet Technology

Prof. Dr. Marcel Tilly

Faculty of Computer Science, Cloud Computing

Vers. 12.12.2021

Plan X-Mas break

- Mock exam available on 23.12 in the LC
- Discussion in the exercise on 14.01.2022
- First regular lecture after the vacations on 11.01.2022

Agenda

The menu for today:

- OSI model revisited: Transport Layer
- TCP/ UDP
- Shell commands for networks
- Distributed systems/ -architectures

Bon appetit!

learning objectives

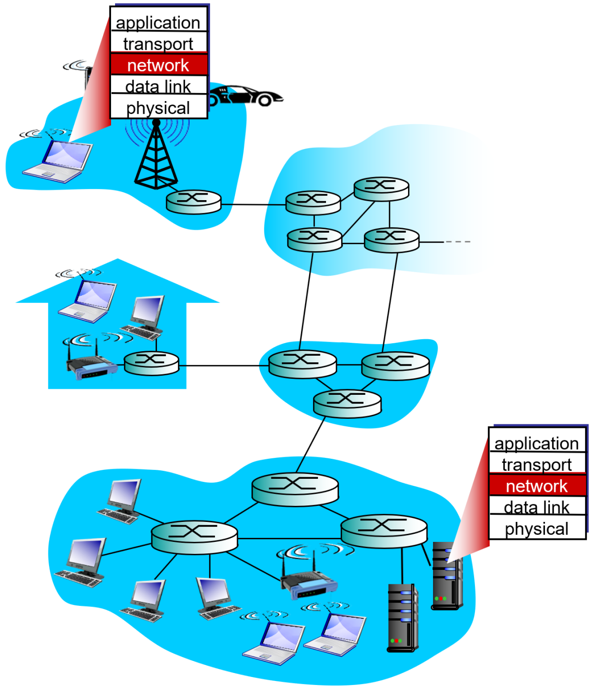
Students will be able to ...

- ... explain TCP/UDP
- ... implement simple client/server communication via UDP/TCP in Python
- ... use shell commands for networking (e.g. IP determination)
- ... explain different concepts of distributed systems

Last week ...

Network Layer

- **end-to-end** connection between sender and receiver
- Sender
  - Packing a Transport Layer segment into datagrams
- Receiver
  - Delivering the datagram to Transport Layer
- Router
  - not interested in layer 4/5
  - only care about forwarding to destination host.



IP Addresses

Where does my IP address come from?

To build a network via IP it is necessary to make an IP configuration at each host. For an IP network the following settings must be made at each host:

- Assigning a unique IP address
- Assigning a subnet mask (subnetmask)
- Assigning the responsible default or standard gateway
- Assignment of the responsible DNS server

DHCP = Dynamic Host Configuration Protocol

- DHCP is a protocol for managing IP addresses in a TCP/IP network and distributing them to the requesting hosts. With DHCP each network participant is able to configure itself automatically.

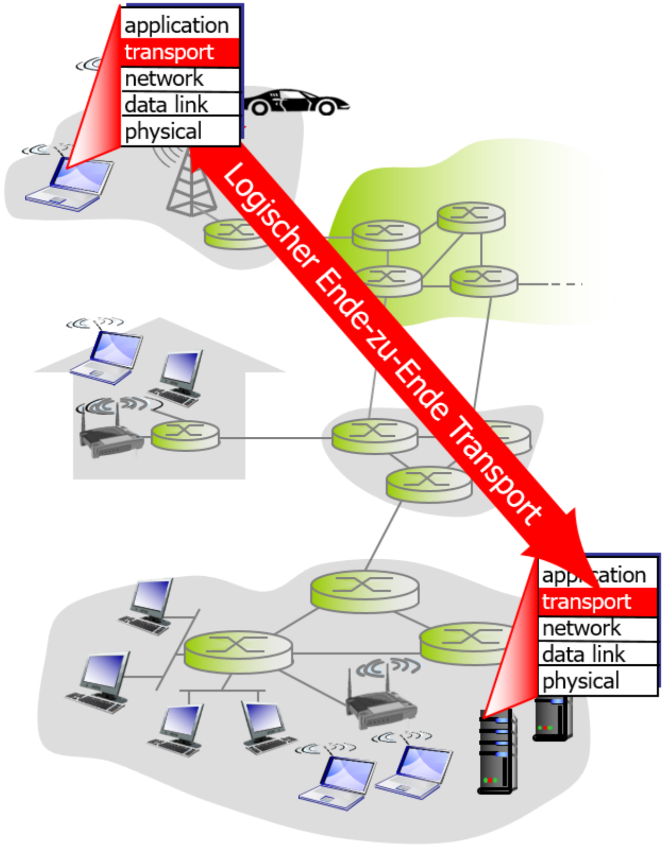
OSI layers

Layers and meaning

#	Layer	Meaning	Units
7	Application Layer	Application Layer	
6	Presentation Layer	Data	
5	Session Layer	Communication Layer	
4	Transport Layer	Transport Layer	TCP, UDP
3	Network Layer	Network Layer	Packets
2	Data Link Layer	Link Layer	Frame
1	Physical Layer	Physical Layer	Bits, Symbols, Packets

Transport Layer

- Communication between processes on sender receiver side.
    - For network layer communication between hosts.
- Concerns hosts, not routers!
    - Sender: Divides application message into segments, forwards to the network layer.
    - Receiver: Assembles messages from segments, forwards them to the application.
- Transport Layer is part of the **operating system**.



Transport Layer

- The transport layer is responsible for the end-to-end transmission of data from one device to another, regardless of the type of data or how it is transmitted.
- The applications that send or receive the data are uniquely identified by so-called port numbers.
- Before data can be exchanged, an application must reserve a port number.
- Two protocols are mainly used in this layer:
    - the *Transmission Control Protocol* (TCP, defined in IETF RFC 793)
    - the *User Datagram Protocol* (UDP, defined in IETF RFC 768)
    - **TCP** is a connection-oriented, reliable protocol.
    - **UDP**, on the other hand, only ensures that packets are sent to the correct application

Transport layer

Technische Hochschule **Rosenheim**

UDP

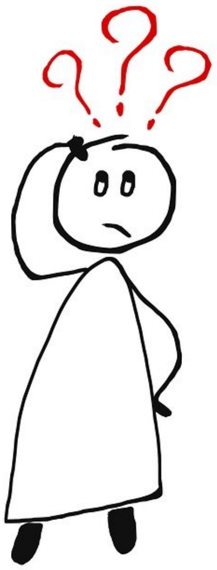
- **UDP** is a *connectionless* protocol, i.e. no connection is established, data is simply sent.
- No other guarantees are made, i.e. you have the same problems here as with *IP datagrams*:
  - The order of the arriving packets can change.
  - Packets can arrive multiple times or not at all.
  - Applications, which use UDP, must handle such problems
  - Pro: data can be sent faster (the connection structure is dropped completely).
- For example, the following applications use UDP: **DNS**, **DHCP**, **SNMP**.

Task

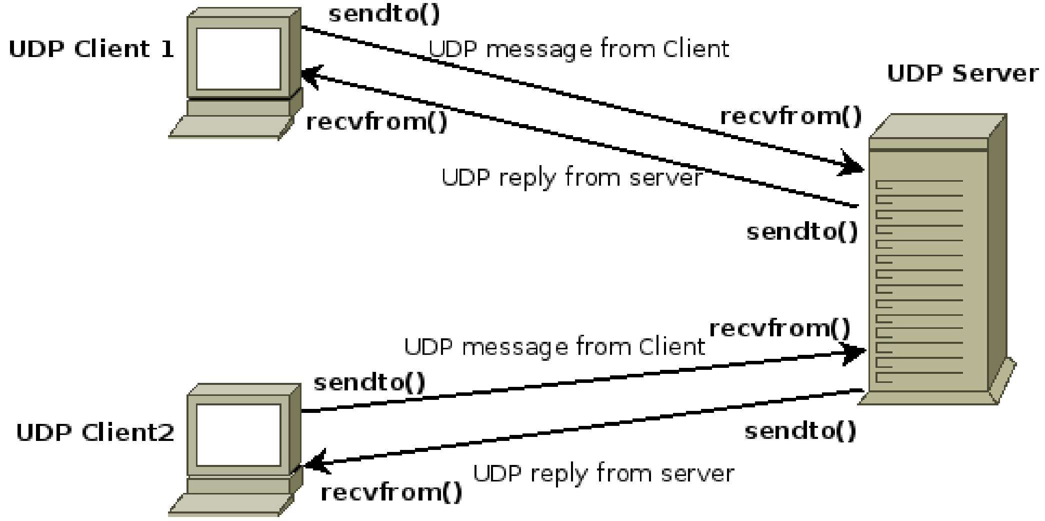
Find out what

- DNS
- DHCP
- SNMP

stands for.

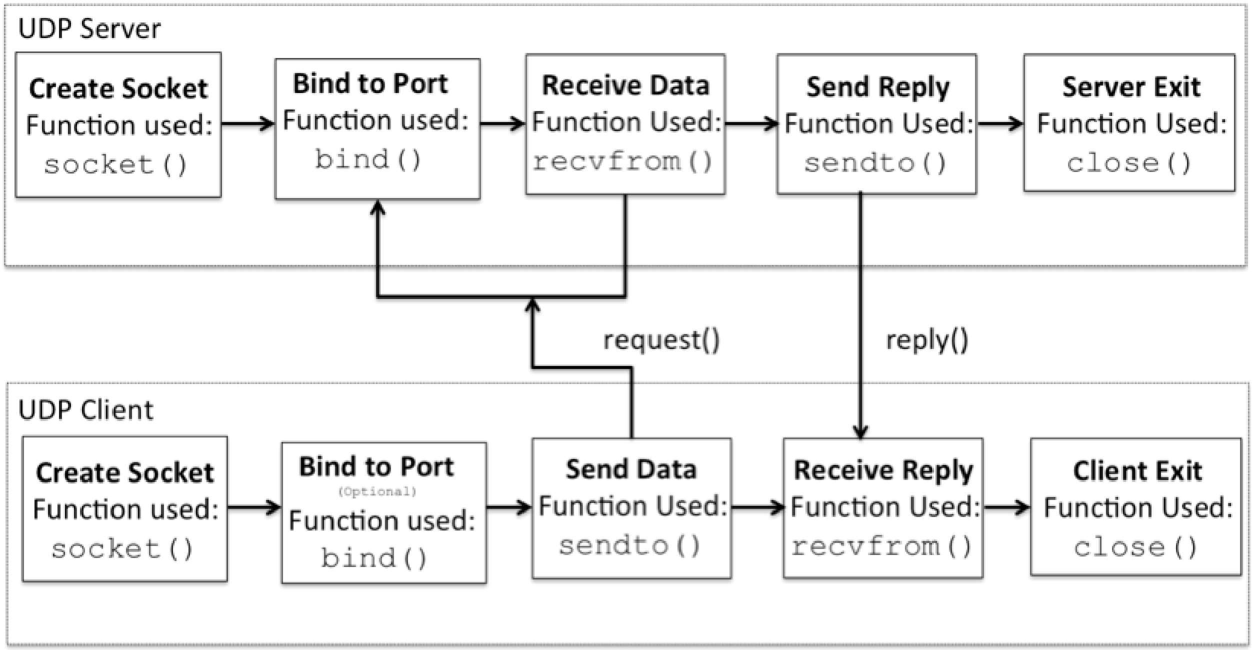


UDP in Python



taken from https://pythontic.com/modules/socket/udp-client-server-example

UDP Client-Server



taken from https://pythontic.com/modules/socket/udp-client-server-example

UDP in Python

Server

```
import socket

localIP     = "127.0.0.1"
localPort   = 7002
bufferSize  = 1024

msgFromServer = "Hello UDP Client"
bytesToSend   = str.encode(msgFromServer)

# Create a datagram socket
UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

# Bind to address and ip
UDPServerSocket.bind((localIP, localPort))
print("UDP server up and listening")

# Listen for incoming datagrams
while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)
    message = bytesAddressPair[0]
    address = bytesAddressPair[1]
    clientMsg = "Message from Client:{}".format(message)
    clientIP  = "Client IP Address:{}".format(address)
    print(clientMsg)
    print(clientIP)

    # Sending a reply to client
    UDPServerSocket.sendto(bytesToSend, address
```

UDP in Python

Client

```python
import socket

msgFromClient       = "Hello UDP Server"
bytesToSend         = str.encode(msgFromClient)
serverAddressPort   = ("127.0.0.1", 20001)
bufferSize          = 1024

# Create a UDP socket at client side
UDPClientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

# Send to server using created UDP socket
UDPClientSocket.sendto(bytesToSend, serverAddressPort)
msgFromServer = UDPClientSocket.recvfrom(bufferSize)
msg = "Message from Server {}".format(msgFromServer[0])
print(msg)
```

Transport layer

TCP

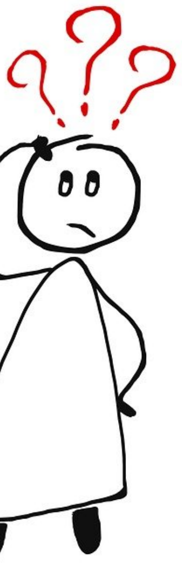
- **TCP** is a key element for the entire model (TCP/IP model) in addition to *IP*.
- The protocol allows the reliable exchange of data (**Reliable**).
  - Protection against transmission errors
  - Protection against packet loss
  - Protection against change of sequence
- **Connection oriented**: Establish connection before data transmission
- The following applications, for example, use TCP: **HTTP**, **FTP**, **SSH**

Task

Find out what

- HTTP
- FTP
- SSH

stands for.



Transport layer

TCP connections

- In order for two applications to communicate with each other, so-called socket pairs consisting of IP address and port numbers must first be created.
- A TCP connection can be uniquely identified by a pair of server and client sockets.
- Before data can now be exchanged, the connection must be established using the TCP handshake:
  - The client sends a SYN packet to the server.
  - The server responds with a SYN/ACK packet to the client.
  - The client completes the connection establishment with an ACK packet to the server.
  - After the TCP handshake, data can be exchanged.
- The connection remains established even if no data is exchanged.
- Only the ordered connection termination (similar to the connection establishment) terminates a connection.

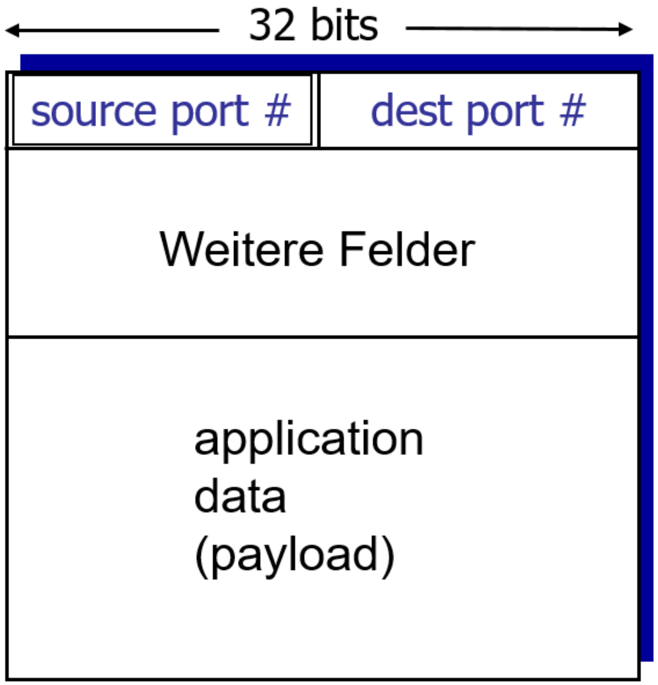
Multiplexing Protocol

Sender

- Inserts process or socket address of sender and receiver into segment.
    - Socket address == port number (16 bit: 1..65535)
    - Destination port of the server must be known
    - Example HTTP(S): Port 80 or 443
    - Source Port: Mostly arbitrary selectable.

Receiver

- Necessary: Received packets are forwarded to correct process based on destination port.



Format eines TCP/UDP Segments

Multiplexing

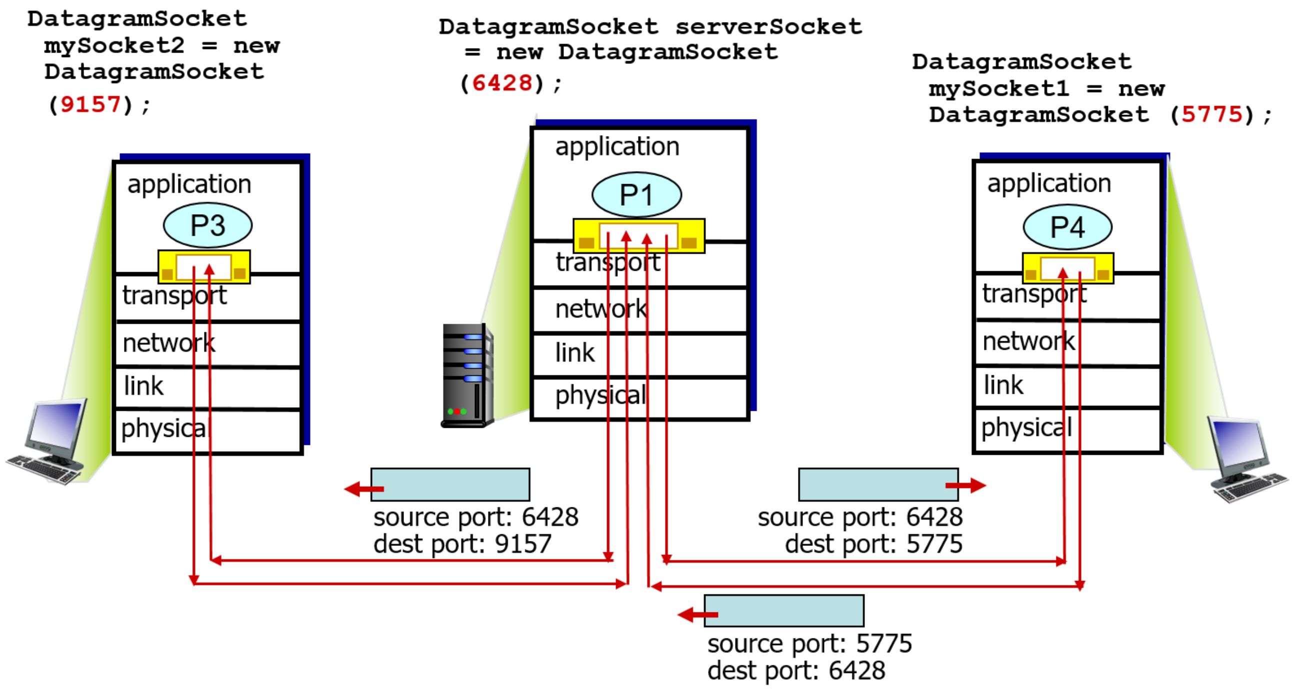
Technische Hochschule **Rosenheim**

TCP vs. UDP

- UDP is **connectionless**.
  - UDP socket specified by: *Dst IP/Port*.
  - Segments with same *Dst IP/Port* are forwarded to same socket.
  - *Src IP* and *Src Port* do not matter
- TCP is **connection oriented**.
  - TCP socket defined by: *Src IP/Port*, *Dst IP/Port*.
  - Only if all 4 values are the same → forward to same socket.
- **Example**: Web server waits for requests on port 80 -> bind()
  - Hosts generate parallel web requests, but with different *Src port* and *Src IP*.
  - server creates the actual socket only on each *accept*

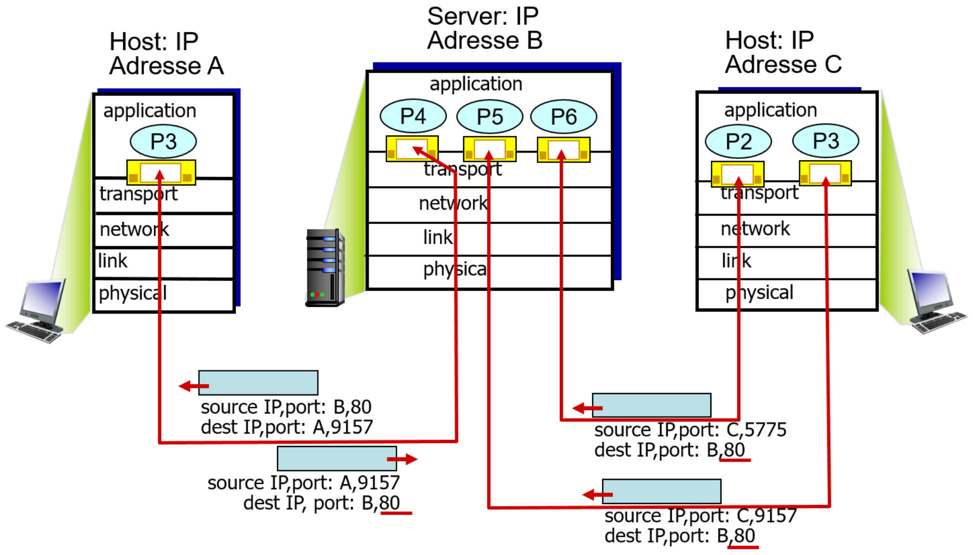
UDP

Multiplexing



TCP

Multiplexing



- The server "runs" 3 sockets, all with the same *dst/IP* and *dst/port*.
- *Src IP/src port* differ, however.

TCP in Python

- via sockets and wit threads

```python
import threading
import socketserver

class ThreadedTCPRequestHandler(socketserver.BaseRequestHandler):

    def handle(self):
        data = str(self.request.recv(1024), 'ascii')
        cur_thread = threading.current_thread()
        print("Current thread " , cur_thread.name , " is handling data:" , data)
        response = bytes("{}: {}".format(cur_thread.name, data), 'ascii')
        self.request.sendall(response)

class ThreadedTCPServer(socketserver.ThreadingMixIn, socketserver.TCPServer):
    pass

if __name__ == "__main__":
    # Port 0 means to select an arbitrary unused port
    HOST, PORT = "localhost", 9999

    server = ThreadedTCPServer((HOST, PORT), ThreadedTCPRequestHandler)
    with server:
        ip, port = server.server_address

        # Start a thread with the server -- that thread will then start one
        # more thread for each request
        server_thread = threading.Thread(target=server.serve_forever)
        # Exit the server thread when the main thread terminates
        server_thread.daemon = True
        server_thread.start()
        print("Server loop running")

        server_thread.join()
        server.shutdown()
```

TCP in Python

Client code

```python
import socket

def client(ip, port, message):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        sock.connect((ip, port))
        sock.sendall(bytes(message, 'ascii'))
        response = str(sock.recv(1024), 'ascii')
        print("Received: {}".format(response))

if __name__ == "__main__":
    HOST, PORT = "localhost", 9999

    client(HOST, PORT, "Hello World 1")
    client(HOST, PORT, "Hello World 2")
    client(HOST, PORT, "Hello World 3")
```

Domain Name System (DNS)

Serves as *"phone book"*: Converts a domain name into an IP address!

- Translation Hostname -> IP Address
  - Name: th-rosenheim.de (easy to remember)
  - IP address: 141.60.160.196 (readable for machines)

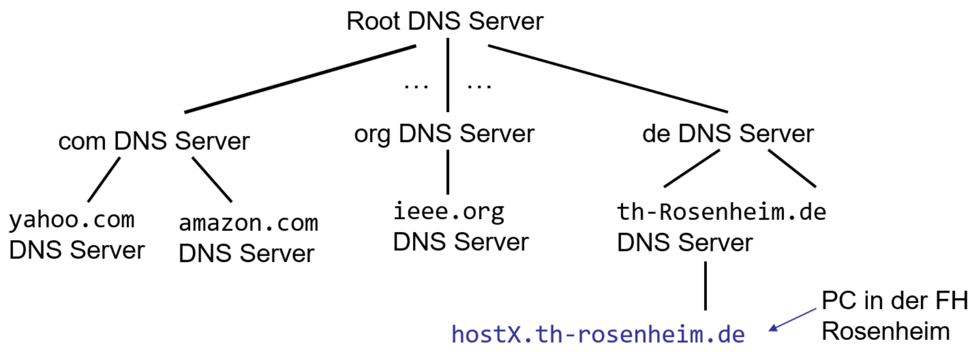
Further tasks

- Host Aliasing: Host can have multiple names, translation
- Canonical name: relay1.west-coast.enterprise.com
- Alias name: www.enterprise.com
- Load Balancing
  - Replicated Web Servers: Many IP addresses have the same name
  - DNS server response determines which physical server / IP is used

Components of DNS

- Distributed directory
  - Hierarchy of name servers
- Application Layer Protocol
  - Hosts and DNS servers communicate with each other.
  - Important Internet function was implemented in the Application Layer.
  - Principle: "Complexity at the edge of the Internet".
- Why no centralized DNS?
  - Single point of failure
  - Traffic volume too high.
  - Name server possibly very far away from requesting host -> high round trip time for DNS requests.

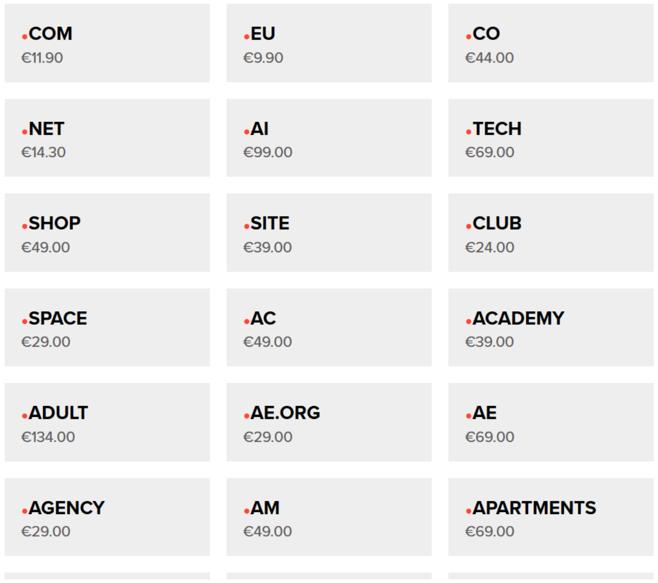
DNS: Hierarchical namespace



- Request to root server
  - DNS server from .com?
- Request to .de DNS server
  - DNS Server of TH Rosenheim?
- Request to DNS server of TH Rosenheim
  - IP address from www.th-rosenheim.de?

Classification of nameservers

- root
  - Knows IPs of all TLD nameservers
  - de, .org, .net, .com, .edu
- Top-Level Domain (TLD)
  - e.g. **.de** TLD server knows nameservers for domain th-rosenheim.de
- Authoritative
  - DNS server of a domain
  - Knows IP addresses of hosts
    - Example: eagle.th-rosenheim.de



.COM €11.90	.EU €9.90	.CO €44.00
.NET €14.30	.AI €99.00	.TECH €69.00
.SHOP €49.00	.SITE €39.00	.CLUB €24.00
.SPACE €29.00	.AC €49.00	.ACADEMY €39.00
.ADULT €134.00	.AE.ORG €29.00	.AE €69.00
.AGENCY €29.00	.AM €49.00	.APARTMENTS €69.00

- Resolver
  - Makes DNS requests to root/TLD/authoritative
  - Does not store any authoritative information itself, only caching!

Commands

ifconfig (Linux)/ ipconfig (Win)

- Configures a network interface.
- Used to (de)activate network interfaces and to assign
- IP address configurations or to display the **current configuration**.
- Option -a displays all interfaces (including disabled ones).

```
$ # configuration of eth0
$ ifconfig eth0 up 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
$ ifconfig wifi
wifi0 link encap:Ethernet HWaddr d4:3b:04:7b:f4:7b
          inet addr:141.60.47.235 Bcast:141.60.47.255 Mask:255.255.240.0
          inet6 addr: fe80::447c:6b2a:facf:1c43/64 Scope:Unknown
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Commands

route (Linux/ Win)

- Shows/changes the routing table.
- Usually used after ifconfig to create a static route for a host/network (e.g. to enter the default gateway).
- Options:
  - n -> no name resolution of IP addresses,
  - net -> create a static route for a network host.

```
$ # create a network route
$ route add -net 10.0.2.0 netmask 255.255.255.0 dev eth0
$ route add default gw 10.0.2.3
$ route
destination router genmask flags metric ref
default 10.0.2.3 0.0.0 UG 0 0
10.0.2.0 * 255.255.255.0 U 0 0
192.168.56.0 * 255.255.255.0 U 0 0
```

Commands

Technische Hochschule Rosenheim

ip (Linux only)

- Displays and modifies network interfaces, routing tables, and more.
- The programs **ifconfig**, **route** and **arp** are still used
- The ip program (part of the iproute2 package) combines these functions

```
$ ip link set eth0 up
$ ip addr add 10.0.2.15/24 broadcast 10.0.2.255 dev eth0
$ ip route add 10.0.2.0/24 dev eth0
$ ip route add default via 10.0.2.3 dev eth0
$ ip neigh add 10.0.2.3 lladdr 52:54:00:12:35:03 dev eth0
```

Commands

ping (Linux/Win)

- Sends a ping (ICMP echo request) to a host.
    - ICMP = *Internet Control Message Protocol*.
- Used to **test** whether a particular host is reachable.
- Caution: Some hosts are configured not to respond for security reasons!

```
$ ping -c 2 www.heise.de

PING www.heise.de (193.99.144.85) 56(84) bytes of data.
64 bytes from www.heise.de (193.99.144.85): icmp_seq=1 ttl=246 time=17.9 ms
64 bytes from www.heise.de (193.99.144.85): icmp_seq=2 ttl=246 time=20.2 ms
```

Question

Which IP addresses have ...

- TH Rosenheim
- ARD
- ZDF
- Technical University Munich

... and to which IPv4 Class do they belong?

Commands

traceroute (Linux) / tracert (Win)

- Shows the route of a packet to a host.
- Can use other methods besides ICMP echo requests (e.g. TCP and UDP).
- Options:
  - I : ICMP method
  - T : TCP method
  - U : UDP method

```
$ traceroute www.ard.de
```

Commands

nmap (Linux)

Nmap ("Network Mapper") is a free and open source (license) utility for network discovery and security auditing. .[...] Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics."_

Scans all ports on `scanme.nmap.org`.

```
nmap -v scanme.nmap.org
```

"Stealth scan against all 256 IPs on the Class C network of `scanme`. Try to determine the OS (-O) additionally

```
nmap -sS -O scanme.nmap.org/24
```

Commands

netcat (nc)

- If *nmap* is not installed.
- Scan port with many options, e.g. -z finds "open ports' without instantiating connection

```
## syntax ##
nc -z -v {host-name-here} {port-range-here}
nc -z -v host-name-here ssh
nc -z -v www.woimmer.de 22

## scan ports ##
nc -zv localhost 6667-7010
```

Commands

netstat

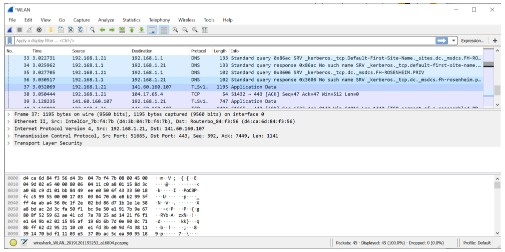
- Displays network information.
- Very versatile program with many options to display routing tables, open connections, statistics, ...
- Display of:
    - *no specification*: open sockets
    - r : routing tables
    - i : network interfaces
    - s : statistics
- options:
    - n : no name resolution
    - l : show only sockets in LISTEN state
    - a : show all sockets
    - t : show TCP sockets
    - u : show UDP sockets

Commands

Network monitoring

There are several programs on Linux for monitoring a network or examining network packets:

- **tcpdump**: CLI;
  - Packets, filtered by conditions, are output.
- **wireshark** (https://www.wireshark.org): GUI;
  - Similar to *tcpdump*, but simpler. Can use tcpdump output.


*WLAN
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.   Time      Source          Destination     Protocol  Length Info
   33 3.022731  192.168.1.21    192.168.1.1     DNS        133 Standard query 0x86ac SRV _kerberos._tcp.Default-First-Site-Name._sites.dc._msdcs.FH-RO…
   34 3.025962  192.168.1.1     192.168.1.21    DNS        133 Standard query response 0x86ac No such name SRV _kerberos._tcp.default-first-site-name.…
   35 3.027705  192.168.1.21    192.168.1.1     DNS        102 Standard query 0x3606 SRV _kerberos._tcp.dc._msdcs.FH-ROSENHEIM.PRIV
   36 3.030517  192.168.1.1     192.168.1.21    DNS        102 Standard query response 0x3606 No such name SRV _kerberos._tcp.dc._msdcs.fh-rosenheim.p…
   37 3.032069  192.168.1.21    141.60.160.107  TLSv1…    1195 Application Data
   38 3.050444  192.168.1.21    104.17.65.4     TCP         54 51432 → 443 [ACK] Seq=47 Ack=47 Win=512 Len=0
   39 3.128235  141.60.160.107  192.168.1.21    TLSv1…     747 Application Data

> Frame 37: 1195 bytes on wire (9560 bits), 1195 bytes captured (9560 bits) on interface 0
> Ethernet II, Src: IntelCor_7b:f4:7b (d4:3b:04:7b:f4:7b), Dst: Routerbo_84:f3:56 (d4:ca:6d:84:f3:56)
> Internet Protocol Version 4, Src: 192.168.1.21, Dst: 141.60.160.107
> Transmission Control Protocol, Src Port: 51665, Dst Port: 443, Seq: 392, Ack: 7449, Len: 1141
> Transport Layer Security

0000  d4 ca 6d 84 f3 56 d4 3b  04 7b f4 7b 08 00 45 00   ··m··V·;·{·{··E·
0010  04 9d 02 e5 40 00 80 06  04 11 c0 a8 01 15 8d 3c   ····@···  ·······<
0020  a0 6b c9 d1 01 bb 84 49  ee e0 50 6f 43 33 50 18   ·k·····I ··PoC3P·
0030  fc c5 99 55 00 00 17 03  03 04 70 d6 e8 b2 99 5f   ···U····  ··p····_
0040  ff 4e ab a4 56 0c 1f 2e  02 bd 86 d7 1b 1e 1e 58   ·N··V··.  ·······X
0050  a8 bd ac 2d 3c fa 50 f1  bc 9e 50 e1 91 7b 9e 67   ···-<·P·  ··P··{·g
0060  80 8f 52 59 62 ae 41 cd  7a 78 25 ad 14 21 f6 f1   ··RYb·A·  zx%··!··
0070  e1 64 9b e2 02 15 95 af  19 6b 6b 7d 0e 90 0c 71   ·d······  ·kk}···q
0080  8b ff 62 d2 95 21 10 c0  e1 fd 3b e0 9d f4 38 11   ··b··!··  ··;···8·
0090  39 14 70 bd f1 11 03 e5  37 0b ac 5c ea 90 95 18   9·p·····  7··\····

wireshark_WLAN_20191201195253_a16804.pcapng   Packets: 45 · Displayed: 45 (100.0%) · Dropped: 0 (0.0%)   Profile: Default
```

OSI layers

Layers and meaning

#	layer	meaning	units
7	Application Layer	Application Layer	
6	Presentation Layer	Data	
5	Session Layer	Communication Layer	
4	Transport Layer	TCP, UDP	
3	Network Layer	Network Layer	Packets
2	Data Link Layer	Link Layer	Frame
1	Physical Layer	Physical Layer	Bits, Symbols, Packets

Question?

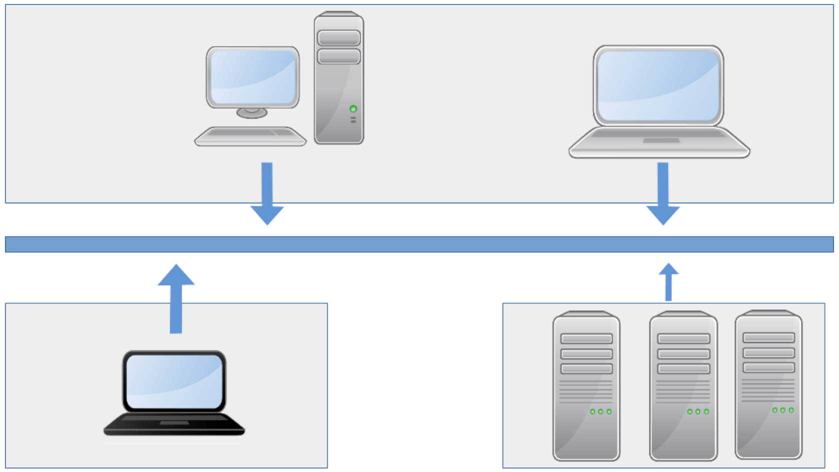
What constitutes a distributed system?

What is part of it?

Distributed system

A practical definition:

- A **distributed system**
  - consists of a set of *autonomous computers*
  - which are interconnected by a *computer network* and which are
  - are equipped with *software for coordination*.



Distributed system

A more general definition:

- A **distributed system** (DS) is a system in which...
  - hardware and software components,
  - which are located on networked computers,
  - communicate with each other and coordinate their actions
  - by exchanging *messages*.
- A **distributed application** is an application,
  - that uses a distributed system to solve an application problem.
  - It consists of various components that communicates with the components of the DS as well as the users.
- Important aspect here: transparency, i.e., the distributed nature is hidden from the user.

Why distributed systems?

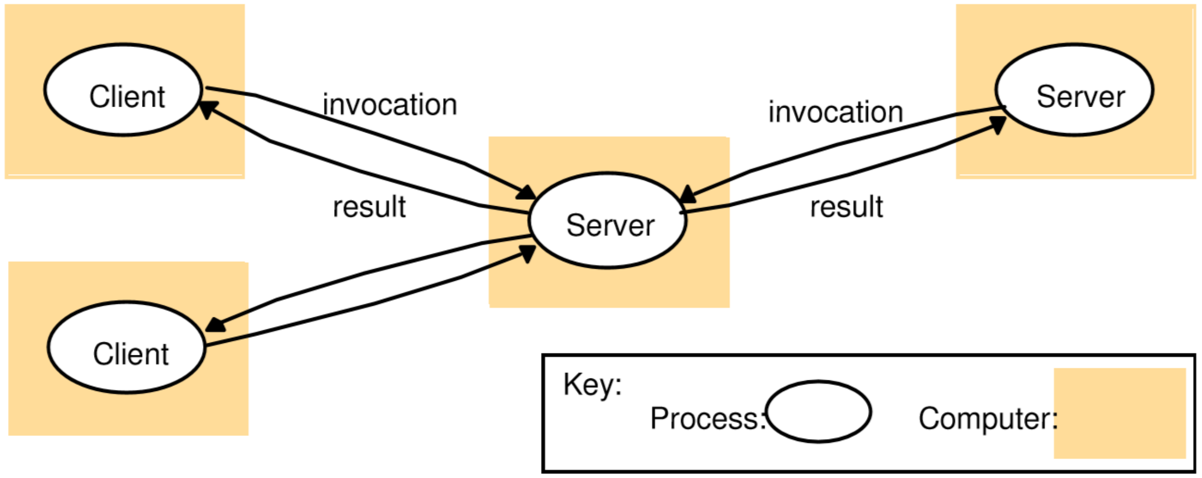
- By the distribution of system components the total system usually becomes clearly more complex.
    - This means that the effort required to create a DS is significantly greater than that of a centralized system.
- So why DS?
    - Solution: Advantages of a computer network

Interaction models of DS

- There are many different ways in which the software components on the computers interact with each other to solve an application problem.
- In the last years some basic models (architectures) have emerged for this purpose:
    - Client-Server (also Multi-Server)
    - Message-oriented architectures
    - Service-Oriented Architecture (SOA)
    - Multi-Tiered
    - peer-to-peer
    - grid computing
    - Cloud Computing
    - Microservices

Client - Server

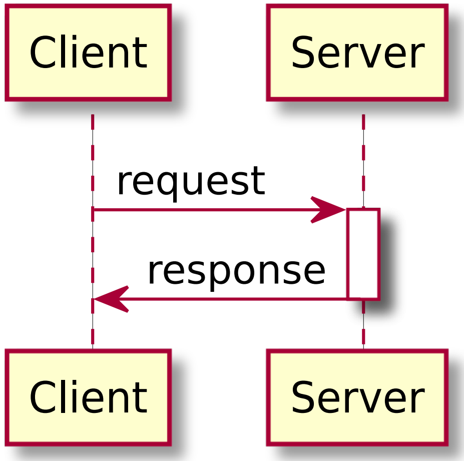
- Client requests a specific service
- Server receives the request, processes it and returns a result



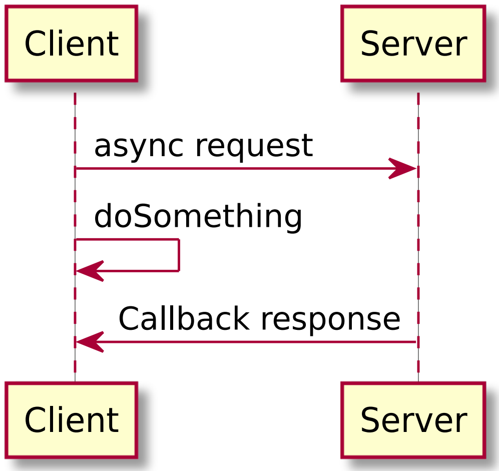
Client - Server

Interaction can be either synchronous or asynchronous

synchronous



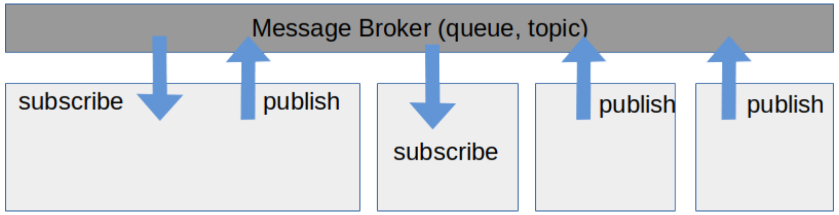
asynchronous



Message-oriented

MO supports three different communication protocols

- Message Passing (Direct communication between applications)
- Message Queueing (Indirect communication via a queue)
- Publish & Subscribe (Publisher provides messages to subscriber)



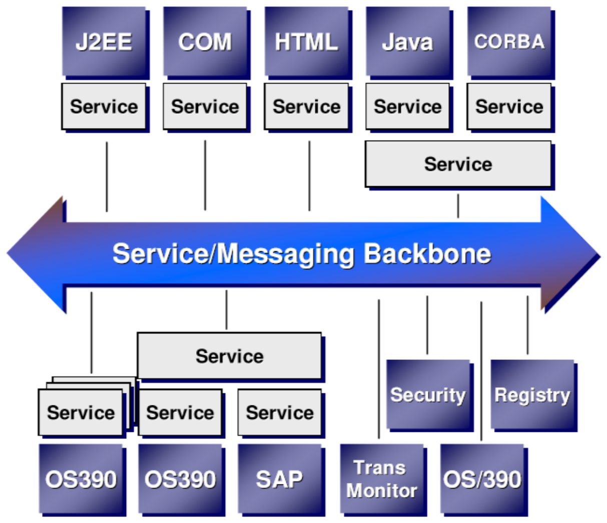
Advantages

- Asynchronous/synchronous communication
- Server/service does not have to be available immediately
- Loose coupling of server/clients
- Parallel processing of messages possible

SOA

SOA = service-oriented architecture

- Consistent continuation of the client-server principle
- service
    - A software component with a formally described interface
    - Gives access to application logic and/or components
    - Communicates by request and response (synchronous and asynchronous)
    - Web services are an important basis of SOAs

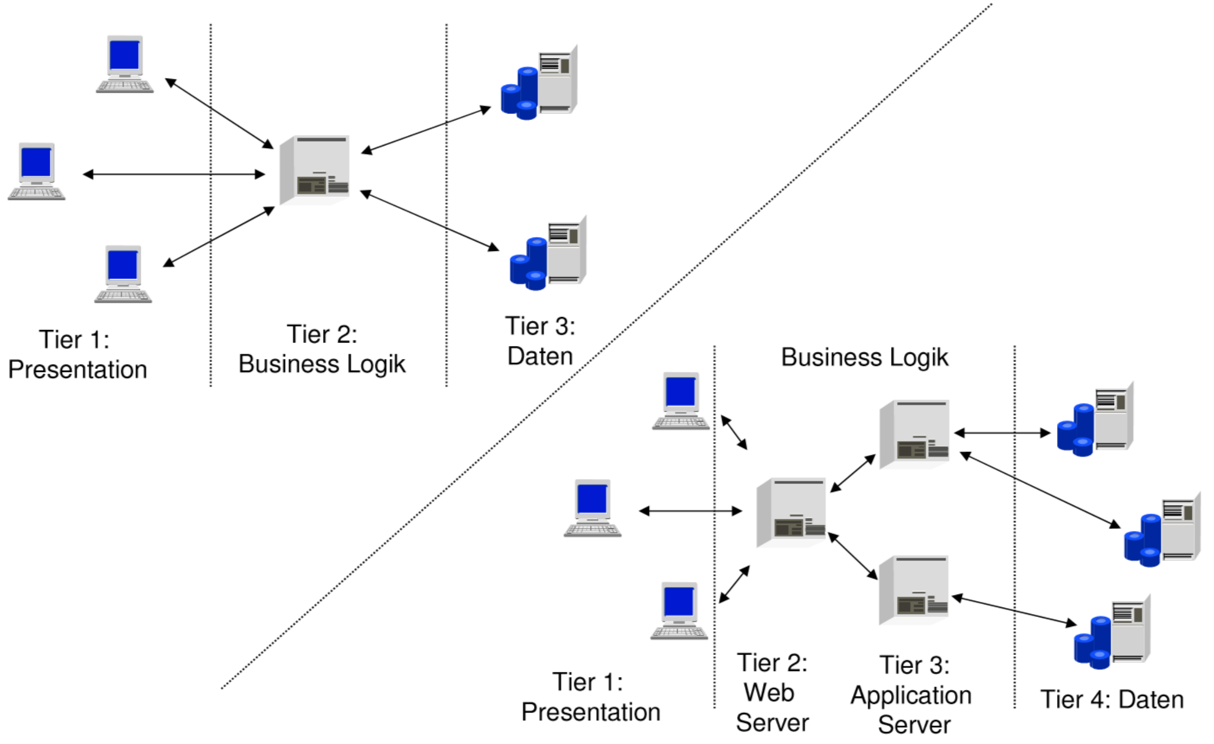


Multi-Tier Architectures

Distributed *(web)* applications today are often developed as **multi-tier applications**.

- Each "tier" (layer, level) has its own task
- Advantages
  - less complexity of the individual parts
  - distribution of implementation tasks
  - flexibility in the distribution of the individual tasks (thinclient)
  - easier maintainability (no client software, exchange of versions)
  - scalability, security
  - In principle a continuation of the client server principle

3-tier / 4-tier


Tier 1:
Presentation
Tier 2:
Business Logik
Tier 3:
Daten
Business Logik
Tier 1:
Presentation
Tier 2:
Web
Server
Tier 3:
Application
Server
Tier 4: Daten

Summary

Lessons learned today:

- TCP/ UDP
- Shell commands (ip, netstat, nv, nmap, ...)
- Wireshark/tcpdump
- Distributed systems
- architectures