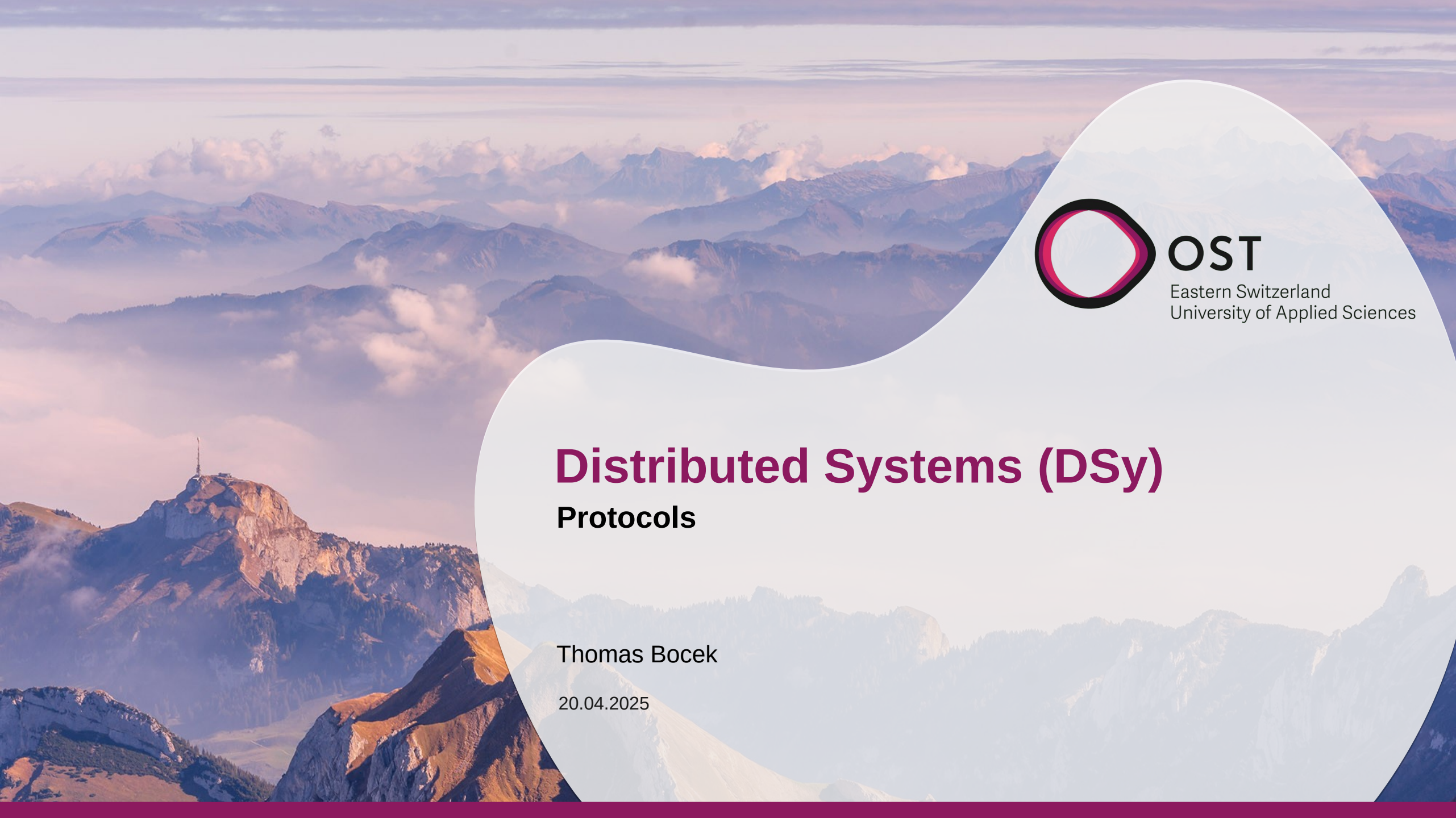


OST
Eastern Switzerland
University of Applied Sciences

Distributed Systems (DSy)

Protocols

Thomas Bocek

20.04.2025

Learning Goals

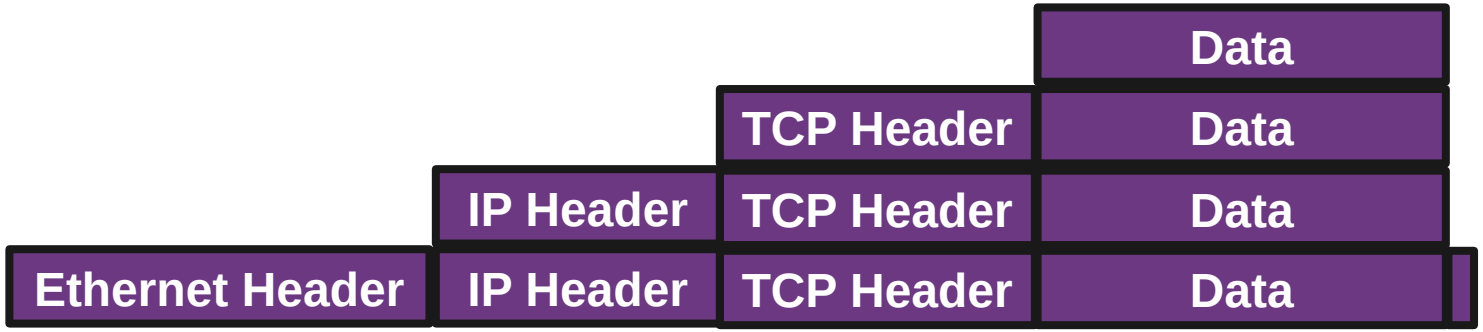
- Lecture 9 (Protocols)
  - How do network layers work?
  - What are the TCP mechanisms?
  - What are the problems of TCP, and how other protocols (HTTP/3) can improve that

Networking: Layers

- Networking: Each vendor had its own proprietary solution - not compatible with another solution
  - IPX/SPX – 1983, AppleTalk 1985, DECnet 1975, XNS 1977
- Nowadays most vendors build compatible networks hardware/software
  - Cisco, Dell, HP, Huawei, Juniper, Lenovo, Netgear, MicroTik, Ubiquiti, etc.

- Goal of layers: interoperability
  - 1984: ISO 7498 - The Basic Reference Model for Open Systems Interconnection

OSI model	"Internet model"
Application	Application
Presentation	
Session	
Transport	Transport
Network	Internet
Data link	Link
Pysical	



OST

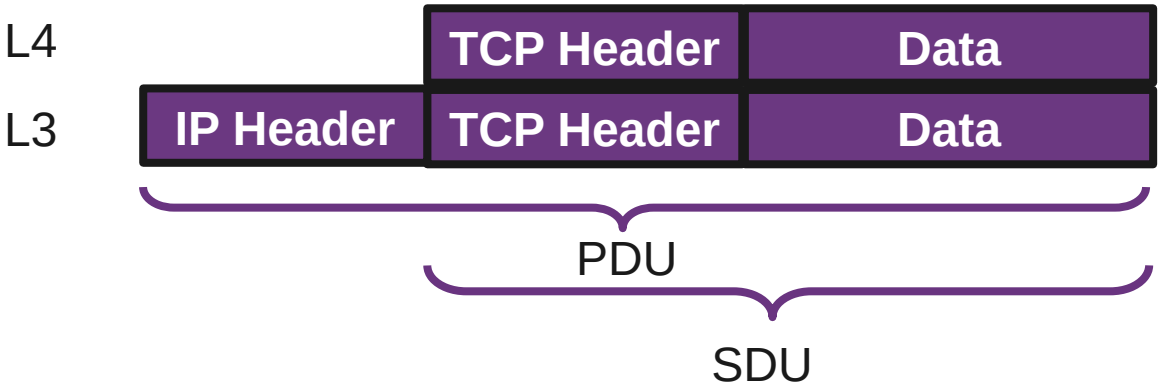
Networking: Definitions

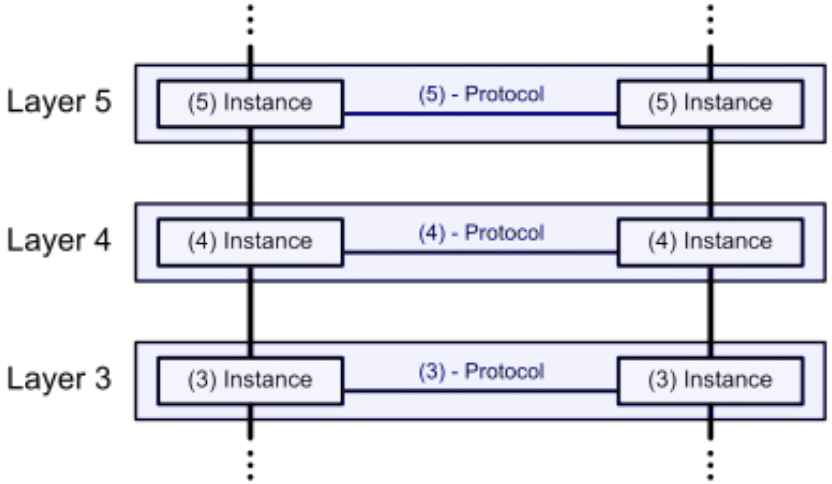
RFC 1122, Internet STD 3 (1989)	OSI model
Four layers	Seven layers
"Internet model"	OSI model
Application	Application
	Presentation
	Session
Transport	Transport
Internet	Network
Link	Data link
	Physical

source: https://en.wikipedia.org/wiki/Internet_protocol_suite

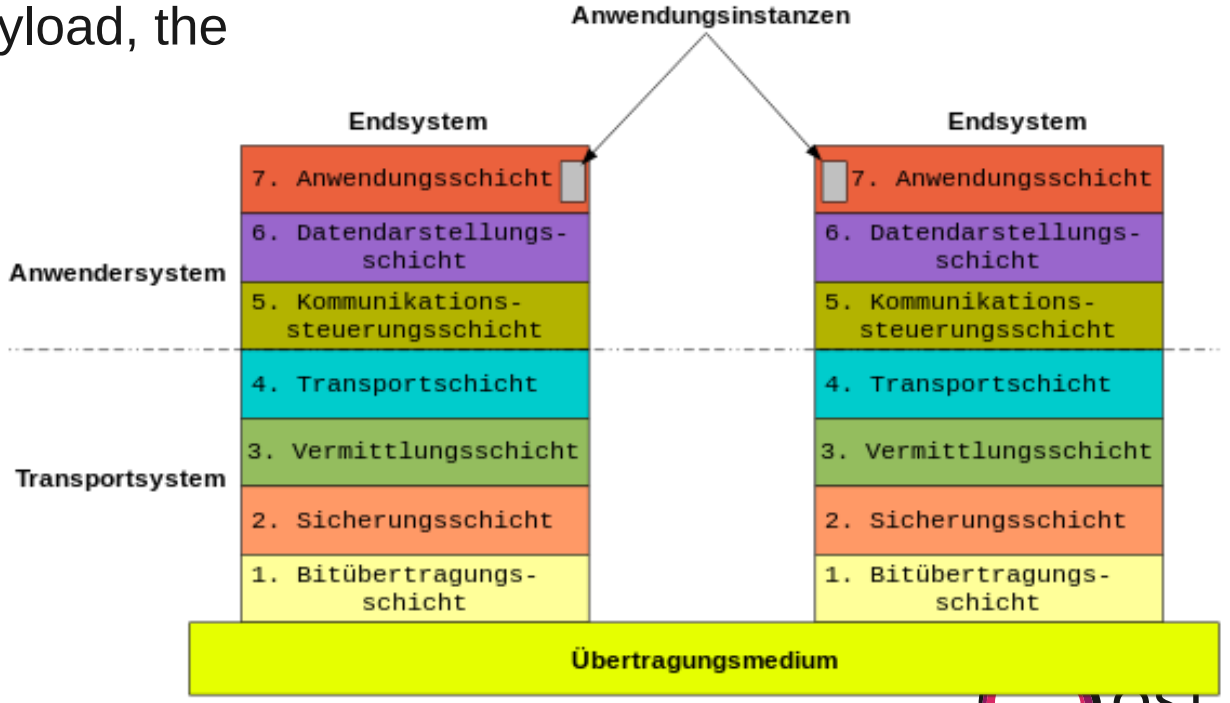
Layer Abstraction

- Protocols enable an entity/instance to interact with an entity/instance at the same layer in another host
- Service definitions: provide functionality to an (N)-layer by an (N-1) layer
- Each PDU contains a protocol header and payload, the service data unit (SDU). E.g. PDU of L3:





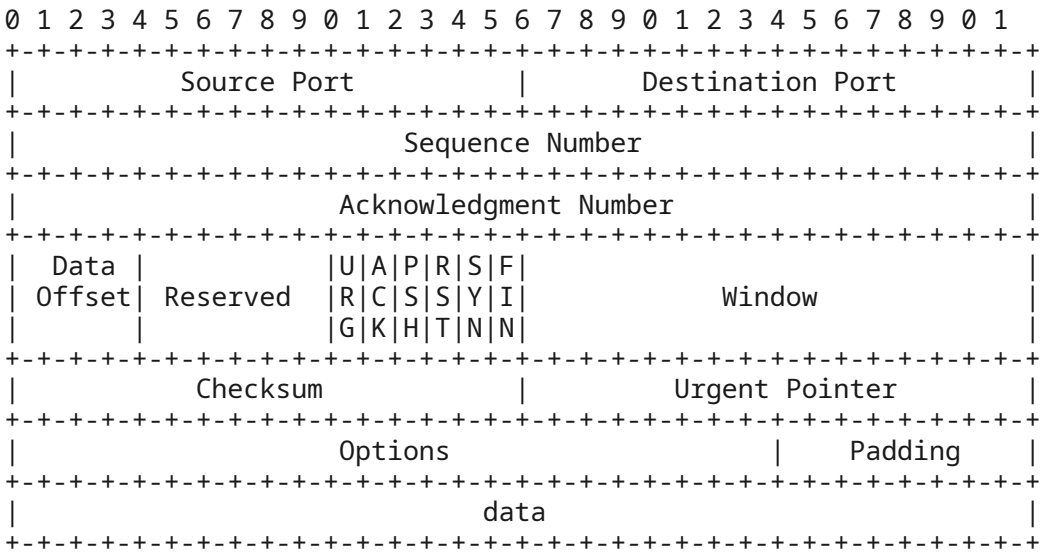
source: https://en.wikipedia.org/wiki/OSI_model



source: https://de.wikipedia.org/wiki/OSI-Modell

Layer 4 - Transport

- TCP (Transmission Control Protocol)
  - Reliable (retransmission)
  - Ordered
  - Window – capacity of receiver
  - Checksum – 16bit (crc16)
  - TCP overhead: 20bytes
    - IP overhead: 20bytes
    - Ethernet frame: 18bytes (crc32)
- TCP tries to correct errors; you don’t need to worry…
  - Sometimes, you need to worry…

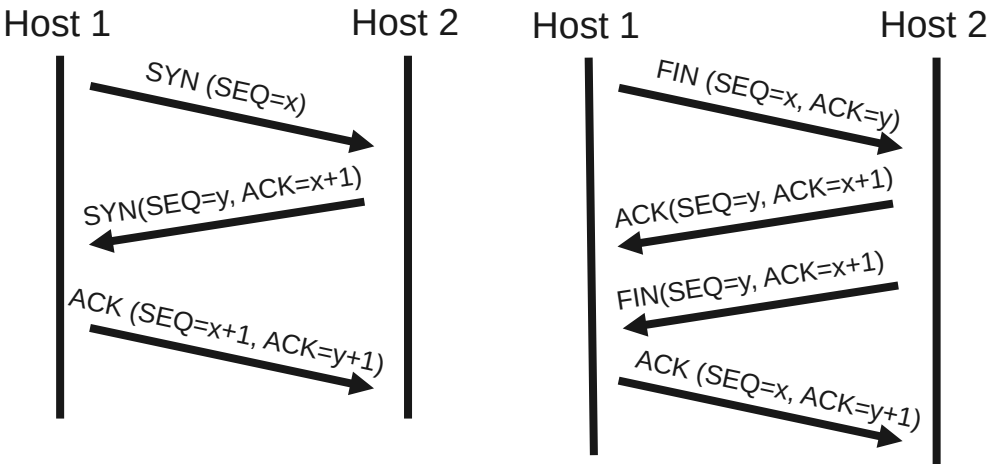


source: https://datatracker.ietf.org/doc/html/rfc9293

OST

Layer 4 - TCP

- Connection establishment
  - SYN, SYN-ACK, ACK (three way)
  - Initiates TCP session: initial sequence number is ~random
- Connection termination
  - FIN, ACK + FIN, ACK (three/four way)
  - 3-way handshake, when host 1 sends a FIN and host 2 replies with a FIN & ACK
- Sequences and ACKs
  - Identification each byte of data
  - Order of the bytes → reconstruction
  - Detecting lost data: RTO, DupACK:

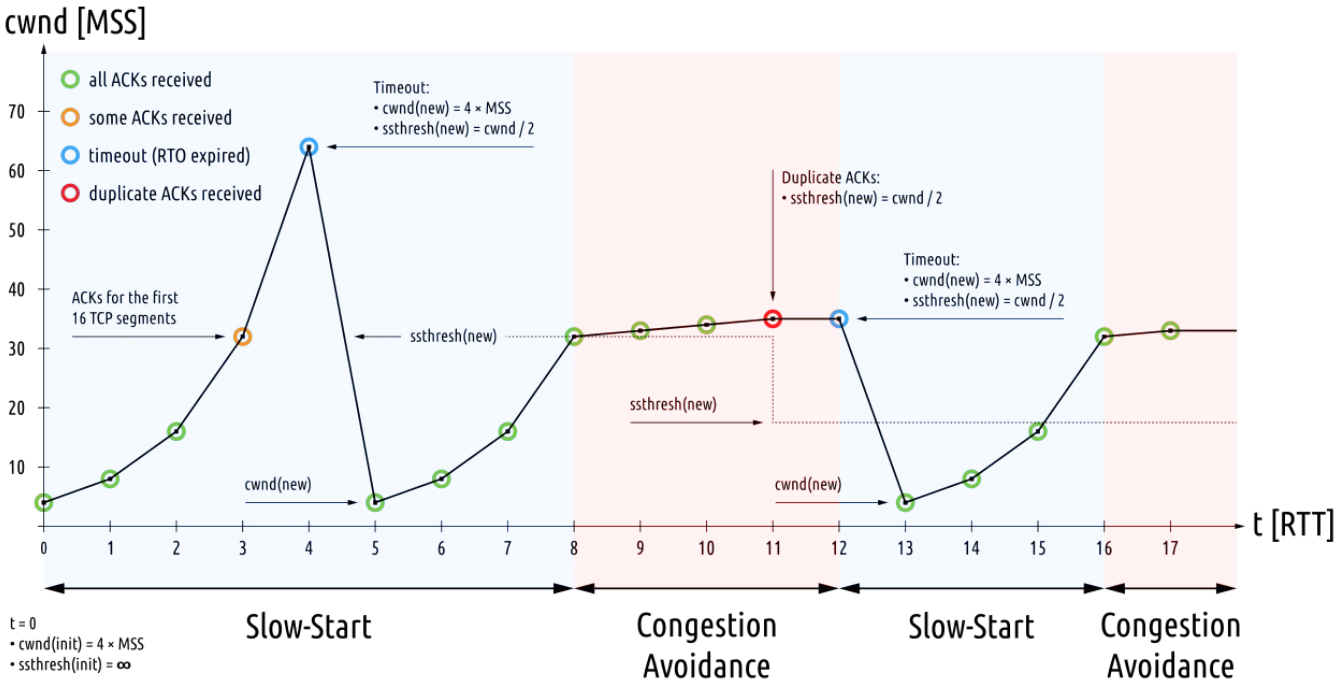


- Retransmission timeout
  - If no ACK is received aftert timout (e.g. 2xRTT), resend.
- Duplicate cumulative acknowledgements, selective ACK [link]
  - ACKs for last consecutive packets
  - 3 times same ACK → retransmit missing packets (fast retransmit)

OST

Layer 4 - TCP

- Flow control
  - Sender is not overwhelming a receiver
  - Back pressure
  - Sliding window:
    - Receiver specifies the amount of additionally received data in bytes that can be buffered
    - Sender up to that amount of data before ACK
- Congestion control
  - slow-start
  - congestion avoidance
- Difference flow/congestion control



source: https://upload.wikimedia.org/wikipedia/commons/thumb/2/24/TCP_Slow-Start_and_Congestion_Avoidance.svg/1280px-TCP_Slow-Start_and_Congestion_Avoidance.svg.png

OST

TCP/IP from an Application Developer View

- Server in golang (repo)
  - git clone https://github.com/tbocek/DSy [link]
  - go run server.go → server
- Listening on TCP port 8081
  - Return string in uppercase
- Node.js version
- Client: nc localhost 8081

```
const net = require('net');
const server = new net.Server();
server.listen(8081, function() {
   console.log('Launching server...');
});

server.on('connection', function(socket) {
   socket.on('data', function(chunk) {
      console.log(`Data received from client: $
{chunk.toString()}`);

socket.write(chunk.toString().toUpperCase() +
"\n");
   });
});
```

```
package main
import ("bufio"
    "fmt"
    "net"
    "strings")
func main() {
    fmt.Println("Launching server...")
    ln, _ := net.Listen("tcp", ":8081") // listen on all
interfaces
    for {
        conn, _ := ln.Accept() // accept connection on
port
        message, _ :=
bufio.NewReader(conn).ReadString('\n') //read line
        fmt.Print("Message Received:", string(message))
        newMessage := strings.ToUpper(message) //change
to upper
        conn.Write([]byte(newMessage + "\n")) //send
upper string back
    }
}
```


OST

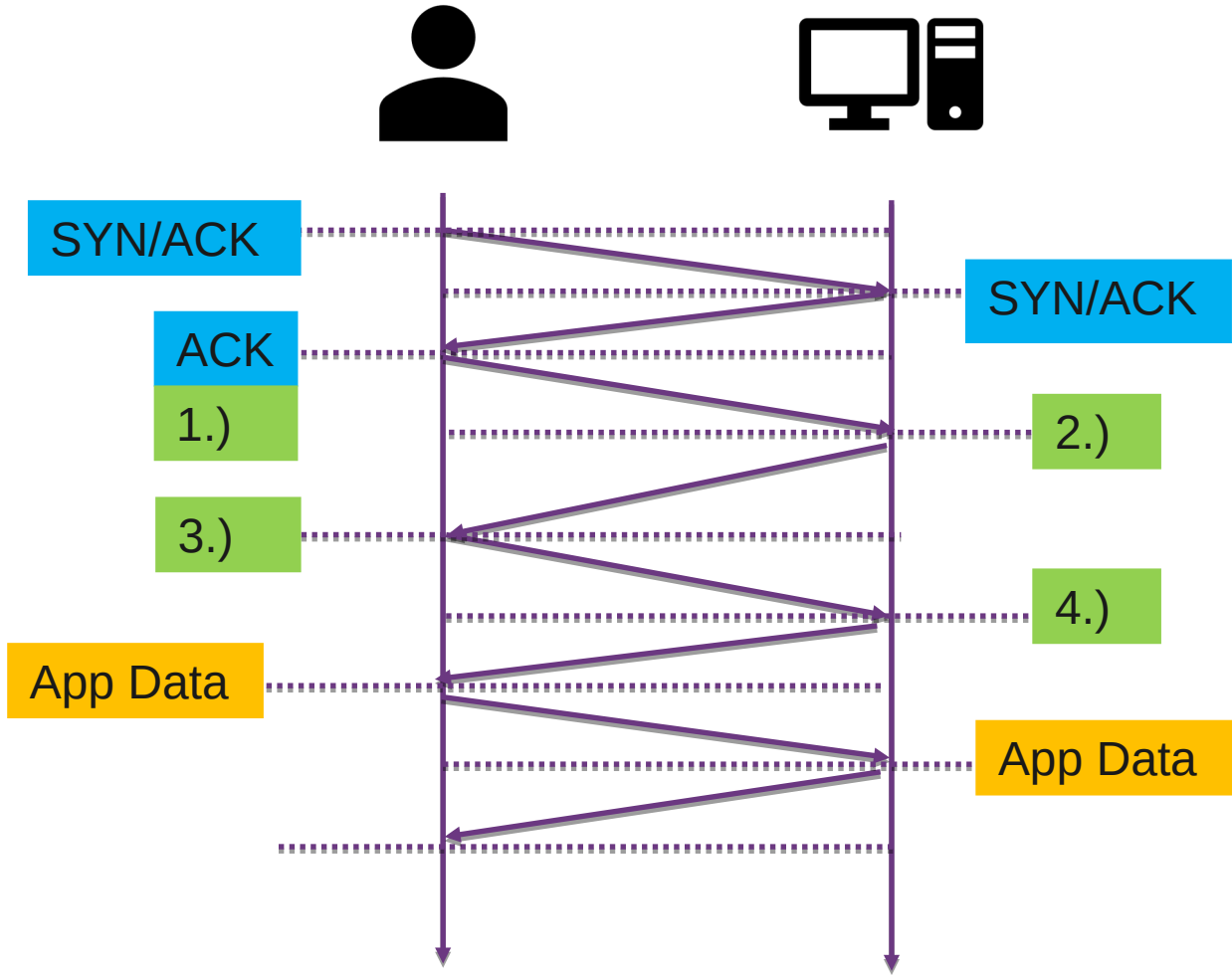
TCP Considerations

- Fallacy 2: Latency is zero
  - Nürnberg data center: 12ms, Australia: 300ms
  - Ping ftp.au.debian.org , Starklink + ~30ms
- Problem: TCP handshake is not flexible
  - You need a handshake (1RT)
    1) If you want to make sure the other side accepts packets (and not drop it) - ensure both sides are ready to transmit and receive data
    2) If you want to exchange public / private keys
  - TCP supports 1) but not 2)
    - Use another security layer for 2), but a security layer needs at least 1 RT
    - TCP + Security = at least 2 RT
      - Nürnberg + Starlink:2 x (12 + 30ms) = 84ms
      - Australian: (2 x 300ms) = 600ms
- TCP + Security at least 2 RT
  - DNS query may be required too: 3 RT
  - Old security protocols add RT: 4RT
- Worst case: Starlink/Australia/DNS/TCP/old sec: 1.4s before data can be sent → new protocols on the way (HTTP/3)

OST

Layer 4 – TCP + TLS

- Security: Transport Layer Security (TLS)
  1. "client hello" lists cryptographic information, TLS version, ciphers/keys
  2. "server hello" chosen cipher, the session ID, random bytes, digital certificate (checked by client), optional: "client certificate request"
  3. Key exchange using random bytes, now server and client can calc secret key
  4. "finished" message, encrypted with the secret key
  - 3 RTT to send first byte, 4RTT to receive first byte



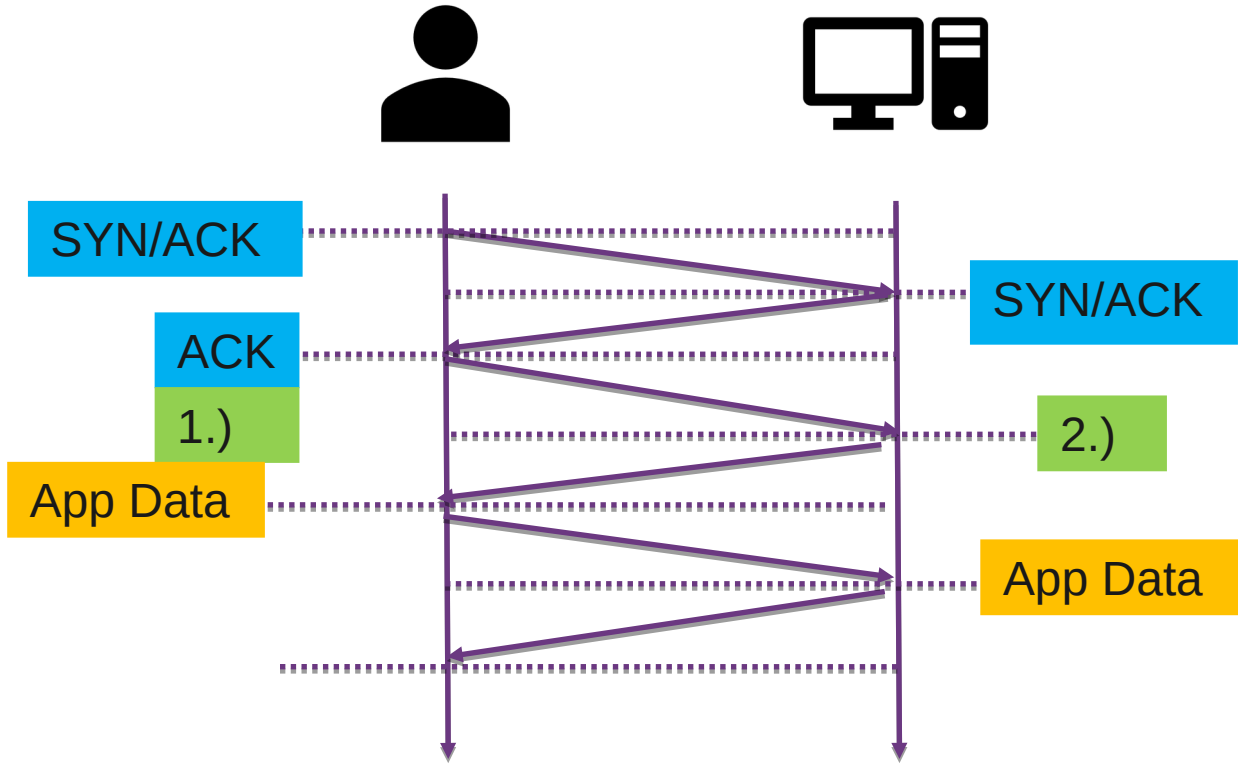
https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_8.0.0/com.ibm.mq.sec.doc/q009930_.htm

OST

```
PING sydney.edu.au (129.78.5.8) 56(84) bytes of data.
64 bytes from scilearn.sydney.edu.au (129.78.5.8): icmp_seq=1 ttl=233 time=307 ms
64 bytes from scilearn.sydney.edu.au (129.78.5.8): icmp_seq=2 ttl=233 time=305 ms
64 bytes from scilearn.sydney.edu.au (129.78.5.8): icmp_seq=3 ttl=233 time=305 ms
```

Layer 4 – TCP + TLS

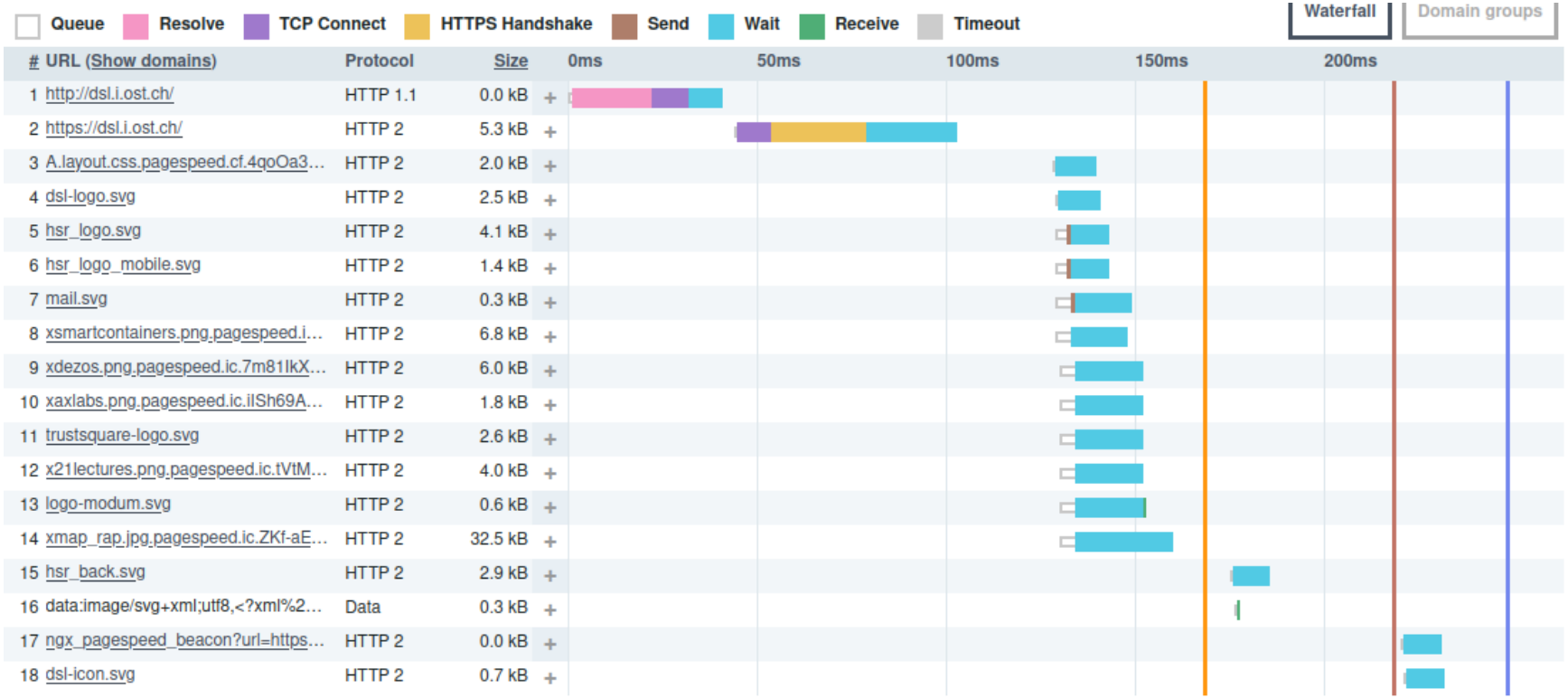
- TCP + TLS handshake:
  - 1RTT - Ping to Australia: 329ms
  - 3RTT = 987ms! No data sent yet
- TLS 1.3, finished Aug 2018
  - 1 RTT instead of 2
    - 1.) Client Hello, Key Share
    - 2.) Server Hello, key Share, Verify Certificate, Finished
  - 0 RTT possible, for previous connections
  - 95% of browsers used already support it



OST

Layer 4 – TCP + TLS

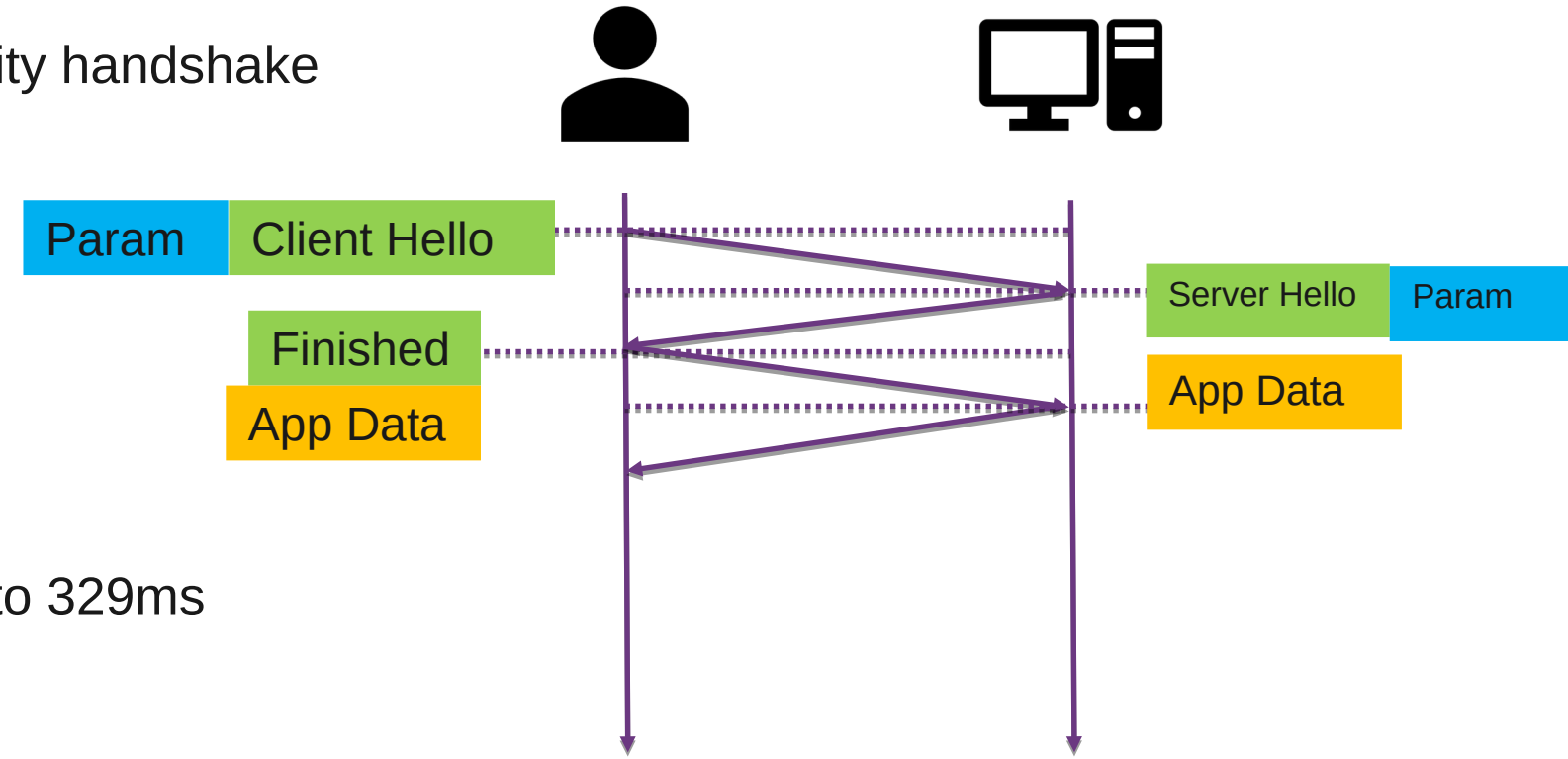
- Website Speed Test [link]
  - Resolve → DNS, TCP Connect → TCP Handshake, HTTPS Handshake → TLS/SSL



OST

QUIC / HTTP3

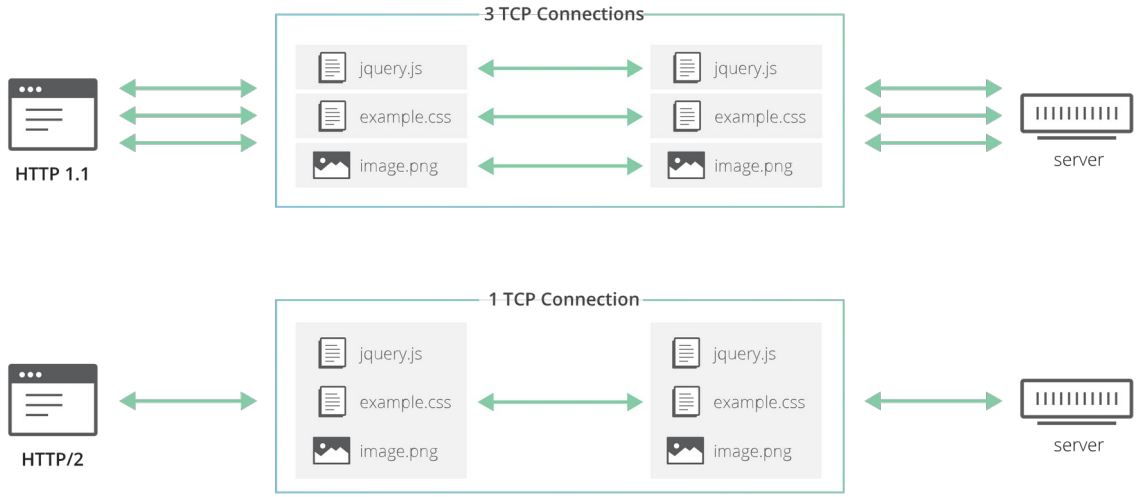
- QUIC: 1RTT connection + security handshake
  - For known connections: 0RTT
  - Built in security
  - Reports
    - Facebook
    - state of HTTP
- Example Australia: from 987ms to 329ms



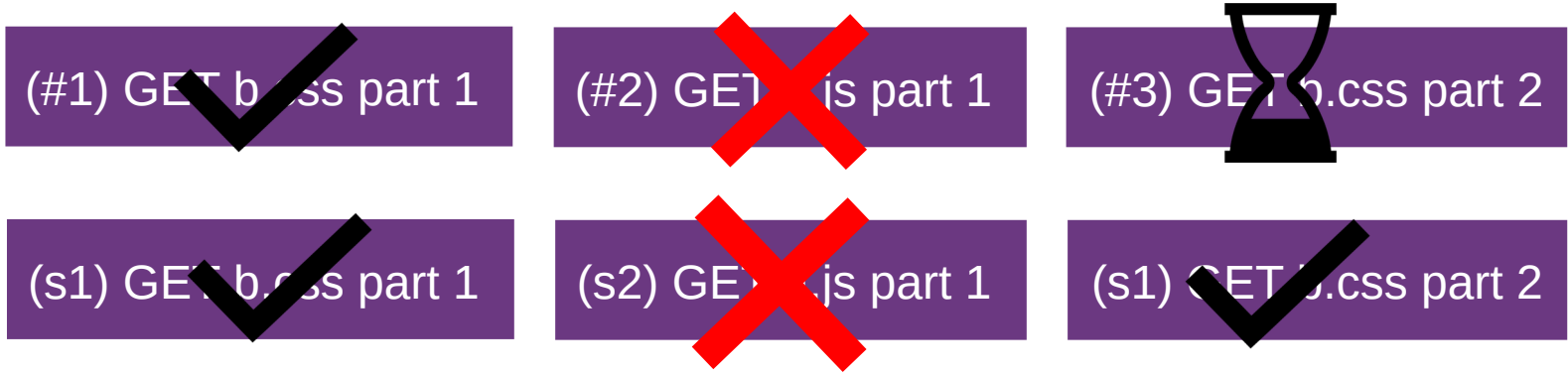
OST

QUIC / HTTP3

- Multiplexing in HTTP/2
  - HTTP/1 → HTTP/2
- HTTP/2: Head-of-line blocking
  - One packet loss, TCP needs to be ordered
  - QUIC can multiplex requests: one stream does not affect others
- HTTP/3 is great, but…
  - NAT → SYN, ACK, FIN, conntrack knows when connection ends, not with QUIC, timeouts, new entries, many entries
  - HTTP header compression, referencing previous headers
  - Many TCP optimizations



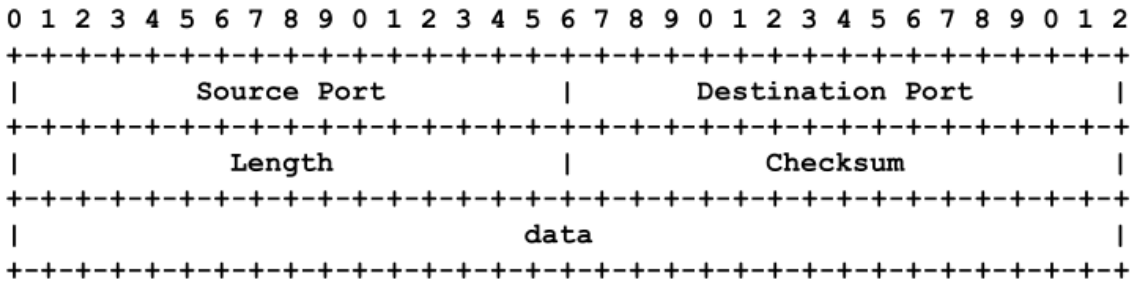
source: https://blog.cloudflare.com/the-road-to-quic/



OST

Layer 4 - Transport

- User Datagram Protocol (UDP)
  - UDP is used for DNS, streaming audio and video
  - Simple connectionless communication model
  - No guarantee
    - Delivery
    - Ordering
    - Duplicate protection

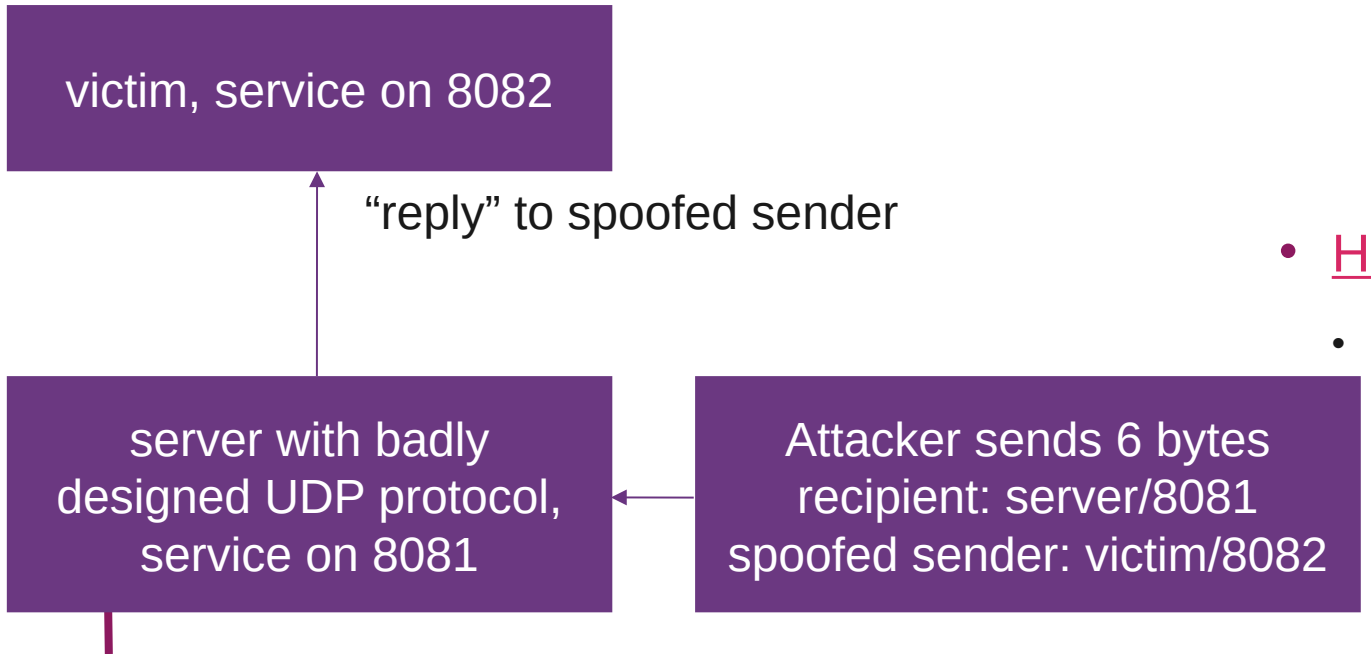


- SCTP (Stream Control Transmission Protocol)
  - Message-based
  - Allows data to be divided into multiple streams
  - Syn cookies - SCTP uses a four-way handshake with a signed cookie.
  - Multi-homing multiple IP addresses of endpoints
  - Not widely used: “We have been deploying SCTP in several applications now, and encountered significant problem with SCTP support in various home routers.”
    - E.g., OpenWRT – not enabled by default
    - E.g., UFW - Uncomplicated Firewall – not supported
  - SCTP used by WebRTC, but tunneled over UDP

OST

Layer 4 - Transport

- DDoS Amplification Attacks
  - Request 10 bytes, reply 100 bytes → factor 10
- Local demo with server-ra/victim, and hping3
  - hping3 --udp IP -p 8081 -E test.tmp -d 6 -s 8082 -c 1



- Attacker in go/Java/node/c#
  - You need to spoof UDP packets, typically not supported in those languages
  - Go: `func DialUDP(network string, laddr, raddr *UDPAddr) (*UDPConn, error)`
    - laddr: we need to set here the victims IP/port
    - But go tries to bind to that
    - Not yours: “bind: cannot assign requested address”
- Hping3: Pen test tool
  - hping3 is a command-line oriented IP, TCP, UDP, ICMP and RAW-IP packet assembler

OST

Comparison – Transport Layer

TCP *	UDP *	SCTP *	QUIC *
Transport layer	Transport layer	Transport layer	Transport layer*
Connection oriented	Connection less	Connection oriented	Connection oriented
Reliable transfer	Unreliable transfer	Reliable transfer	Reliable transfer
Streams	Messages	Messages	Multistream
Guaranteed order	Unordered	User can choose	Guaranteed order
Widely used – HTTP/1, HTTP/2	Widely used – DNS, HTTP/3	WebRTC	HTTP/3
Flow and congestion control	No flow, congestion	Flow and congestion control	Flow and congestion co
Heavyweight	Lightweight	Heavyweight	Heavyweight
Error checking and recovery	Error checking, no recovery	Error checking and recovery	Integrity check