FS20

PROTOCOLS

T. Bocek

thomas.bocek@hsr.ch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Agenda

- **Distributed Systems in the News**
- **Admin / Challenge Task**
- **Definition, Classification (rest of 1st lecture)**
- **Layers / Protocols**
- **Layer 4 - TCP/IP, UDP, QUIC**

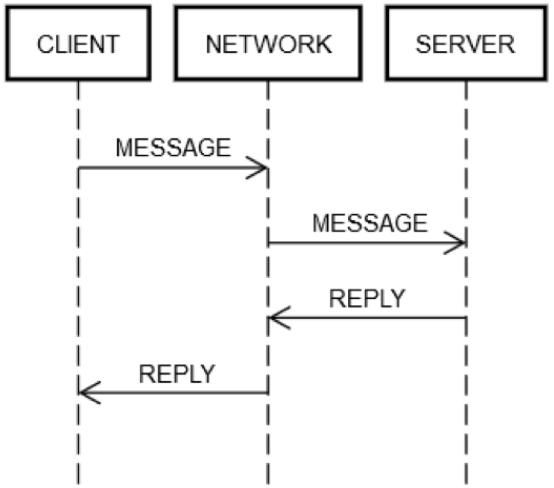
Distributed Systems - In the *News*

- **18.2.2020, Skynet, a decentralized CDN and file sharing platform**
  - At least one person pin the video, it will remain online to the world
  - Skynet is a layer 2 built on top of Sia
  - Sia blockchain: a proof-of-work blockchain that closely adheres to Bitcoin's design principles.
    - The Sia blockchain is used to power a special type of smart contract called a file contract. The file contract is an agreement between an uploader and a host that the host is required to store a specific piece of data for a specific period
  - Soon: users can download from portals using micro-transactions with the Lightning Network

- **Sia commercial offer: Filebase, Goobox S3**
  - Filebase: 1TB / month: 5$
  - Amazon: 1TB / month: 23$ + 1M requests 5$ + 90$ / TB bandwidth
  - Backblaze: 1TB / month: 5$ + 10$ / TB bandwidth

Distributed Systems - In the *News*

- **Stumbled upon recently, but article is older - 2019: Challenges with distributed systems**
  - **Types of distributed systems**
    - **Offline distributed systems: batch processing systems, big data analysis clusters, movie scene rendering farms**
    - **Soft real-time distributed systems: search index builders**
    - **Hard real-time distributed systems: request/reply services**
      - **"…the most difficult to get right."**



1. **POST REQUEST: CLIENT puts request MESSAGE onto NETWORK.**
2. **DELIVER REQUEST: NETWORK delivers MESSAGE to SERVER.**
3. **VALIDATE REQUEST: SERVER validates MESSAGE.**
4. **UPDATE SERVER STATE: SERVER updates its state, if necessary, based on MESSAGE.**
5. **POST REPLY: SERVER puts reply REPLY onto NETWORK.**
6. **DELIVER REPLY: NETWORK delivers REPLY to CLIENT.**
7. **VALIDATE REPLY: CLIENT validates REPLY.**
8. **UPDATE CLIENT STATE: CLIENT updates its state, if necessary, based on REPLY.**

Distributed Systems - In the *News*

- **Steps 1-8: all required in a distributed system**
- **CLIENT, SERVER, and NETWORK can fail independently**
  - **Must handle any failing steps**
- **Last lecture bit-flips: CPU could overheat, power supply could fail, memory/disk could fill up**
  - Unit tests never cover the “what if the CPU fails” scenario
- **Handling failure modes in hard real-time distributed systems**

```
(error, reply) = network.send(remote, actionData)
switch error
  case POST_FAILED:
    // handle case where you know server didn't get it
  case RETRYABLE:
    // handle case where server got it but reported transient failure
  case FATAL:
    // handle case where server got it and definitely doesn't like it
  case UNKNOWN: // i.e., time out
    // handle case where the *only* thing you know is that the server received
    // the message; it may have been trying to report SUCCESS, FATAL, or RETRYABLE
  case SUCCESS:
    if validate(reply)
      // do something with reply object
    else
      // handle case where reply is corrupt/incompatible
```

Distributed Systems - In the News

- **Testing hard real-time distributed systems**
  - Many more test-cases
  - Hardest thing to handle is the UNKNOWN error: how to handle correctly?
- **Real distributed systems have more complicated failure state matrices**
  1. Individual machines
  2. Groups of machines
  3. Groups of groups of machines
  4. And so on (potentially)
- **Distributed bugs are often latent**
  - “It then takes a while to trigger the combination of scenarios that actually lead to these bugs happening”

```
bool isEven(number)
  switch number % 2
    case 0
      return true
    case 1
      return false
```

```
bool distributedIsEven(number)
  switch mathServer.mod(number, 2)
    case 0
      return true
    case 1
      return false
    case UNKNOWN
      return WHAT_THE_FARG?
```

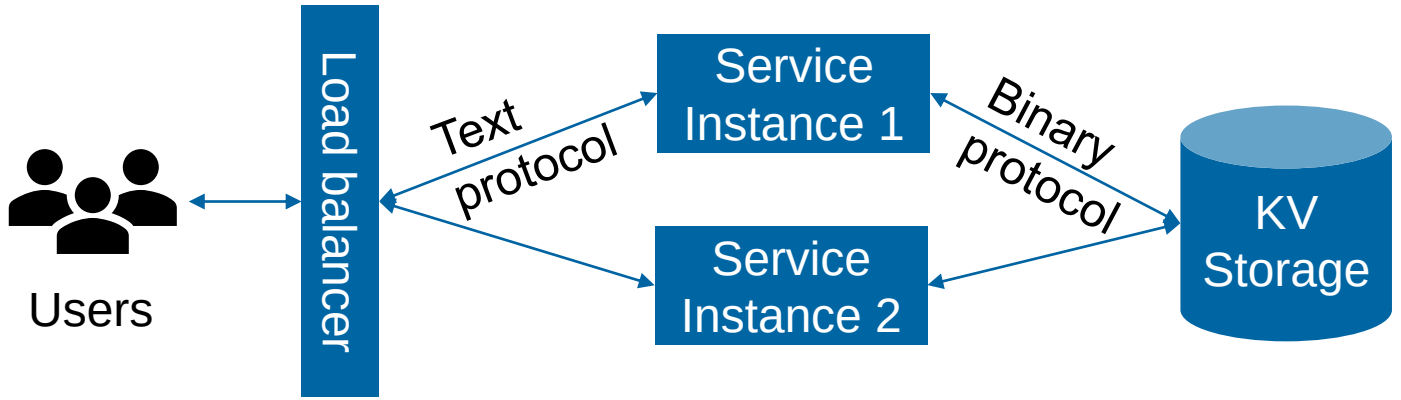
- **Summary of problems in distributed systems**
  - Engineers can’t combine error conditions. Instead, they must consider many permutations of failures. Most errors can happen at any time, independently of (and therefore, potentially, in combination with) any other error condition.
  - The result of any network operation can be UNKNOWN, ...
  - Distributed bugs often show up long after they are deployed to a system
  - Many of the above problems derive from the laws of physics of **networking**, which can’t be changed

Admin / Challenge Task

- **Group forming, who has no group?**
  - Raise your hands
- **Check if your group is here**

- **Ideas for Challenge Task:**
  - Manage and display the value of your cryptocurrencies
  - Shopping list
  - Manage your grades
  - Personal blog
  - URL-shortener
  - Expense tracker
  - Time tracking
  - Online Quiz (e.g., assessment)

- **Unprecise/open requirements**
  - Happens in practice ☺
- **Some students were in the exercises, brainstormed ideas, asked for clarifications, and wrote ideas down**
- **Possible architecture:**



URL: b.socrative.com/student/

Room: HSR

Admin

- **Alpine Image online: https://dsl.hsr.ch/lect/alpine-fs20_v3.ova**
- **Install VirtualBox, click on "File → Import Appliance“ → Start**
- **Alpine user: root, pw: hsr12345**
  - ssh –p 2222 root@localhost
  - ssh-copy-id –p 2222 root@localhost
  - scp –P 2222 vss root@localhost:/root
  - Update package list: apk update
  - List packages: apk list
    - More options: /etc/apk/repositories
  - Search: apk search openjdk
  - Install: apk add openjdk11-jre-headless (+ ~65MB)
  - Packages: https://pkgs.alpinelinux.org/packages

- Alpine Linux
  - “Alpine Linux is built around musl libc and busybox. This makes it smaller and more resource efficient than traditional GNU/Linux distributions. A container requires no more than 8 MB and a minimal installation to disk requires around 130 MB of storage...”
  - Often used for virtualization / containers (there is a 32bit version, arm, ppc, etc)

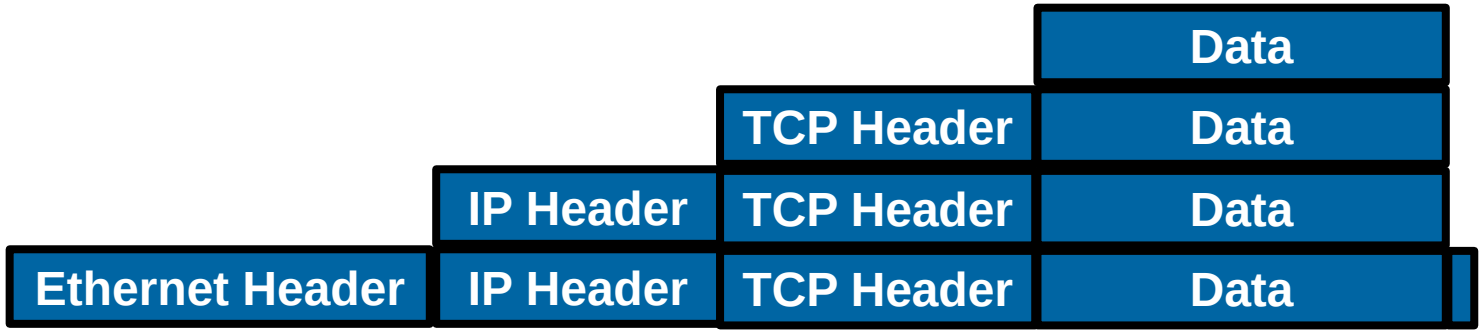
Goals

- **Achieved: I know that distributed systems are hard and involve networking**
- **I know what a layer/protocol is (high level)**
- **I know the latencies of protocols**

Networking: Layers

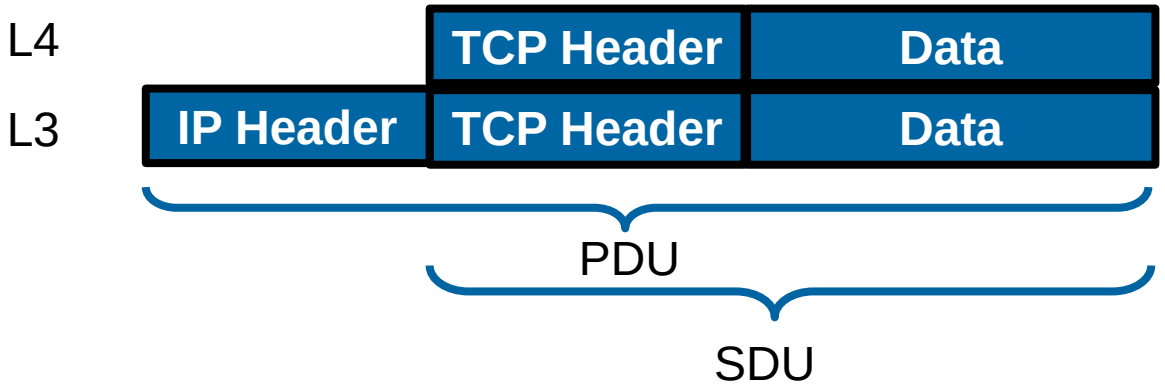
- **Networking: Each vendor had its own proprietary solution - not compatible with another solution**
  - IPX/SPX – 1983, AppleTalk 1985, DECnet 1975, XNS 1977
- **Nowadays most vendors build compatible networks hardware/software from different vendors**
  - Cisco, Dell, HP, Huawei, Juniper, Lenovo, Linksys, Netgear, MicroTik, Siemens, Ubiquiti, etc.
- **Goal of layers: interoperability**
  - 1984: ISO 7498 - The Basic Reference Model for Open Systems Interconnection

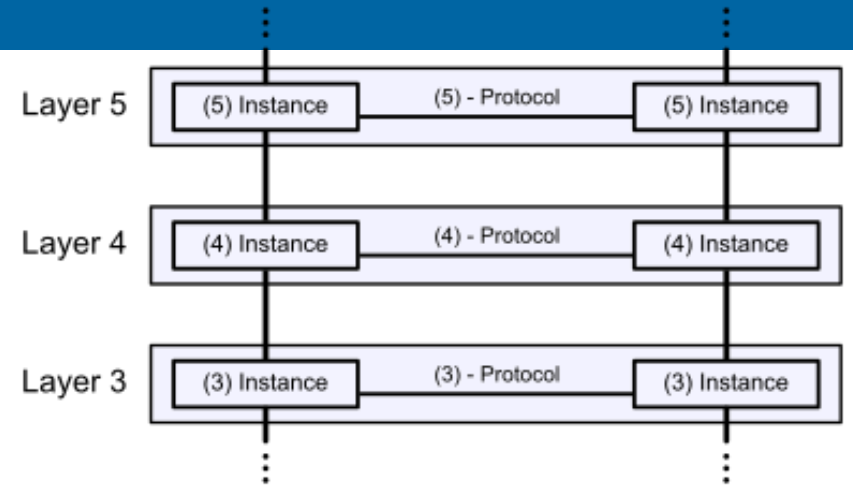
OSI model	"Internet model"
Application	Application
Presentation	
Session	
Transport	Transport
Network	Internet
Data link	Link
Pysical	



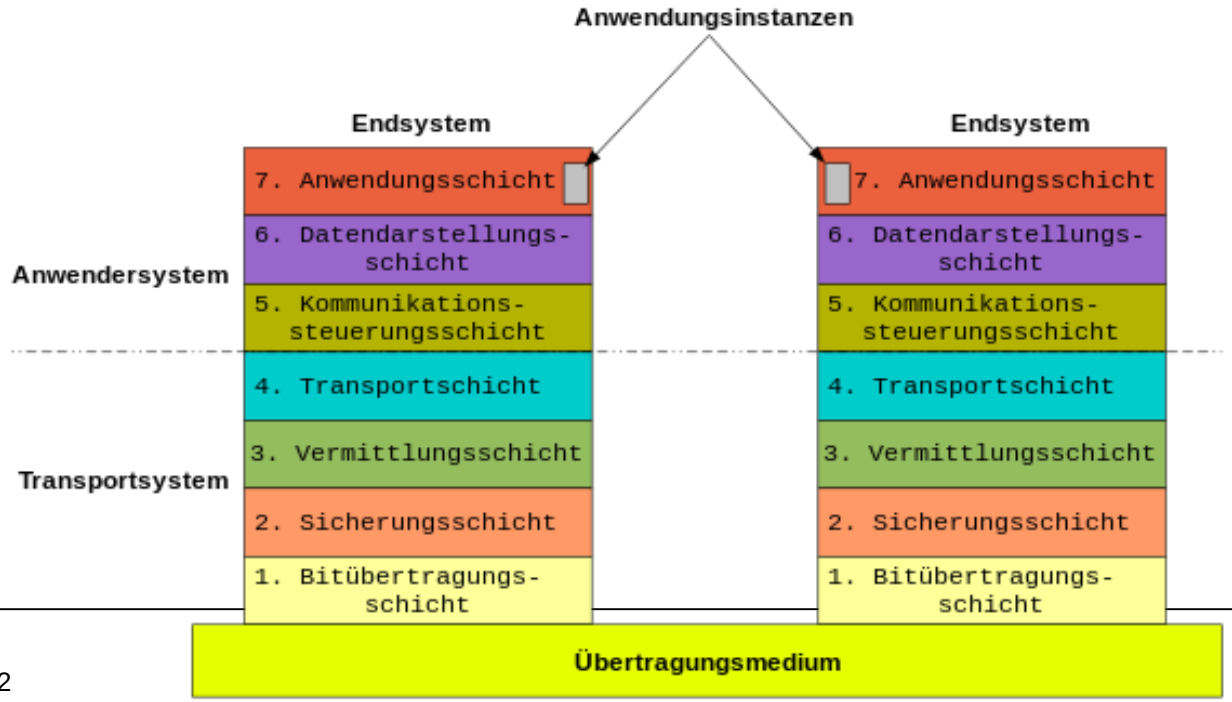
Layer Abstraction

- **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**
- **Layer N exchange protocol data units (PDUs) with layer N protocol. Each PDU contains a protocol header and payload, the service data unit (SDU). E.g. PDU of L3:**





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



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

Internet Model – Different Sources, Different Naming

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

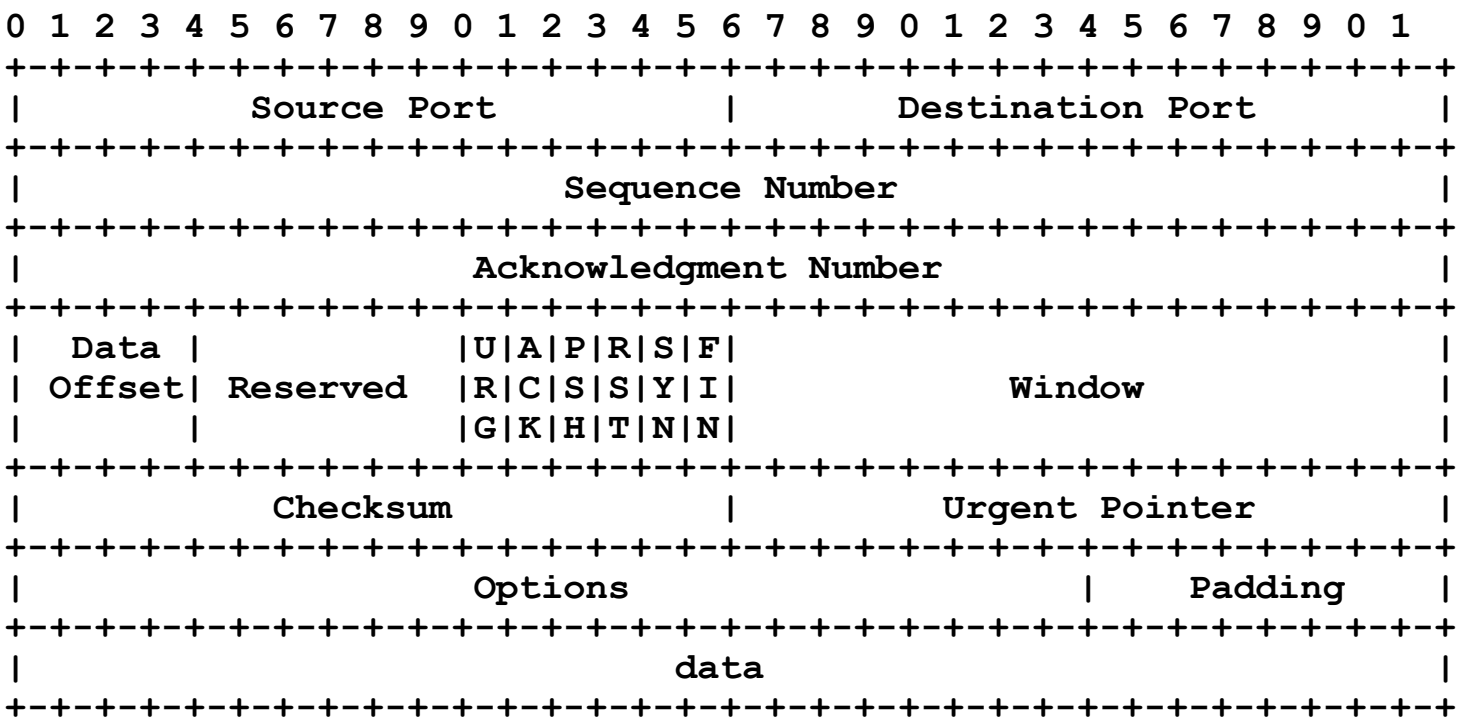
https://en.wikipedia.org/wiki/Internet_protocol_suite

URL: b.socrative.com/student/

Room: HSR

Layer 4 - Transport

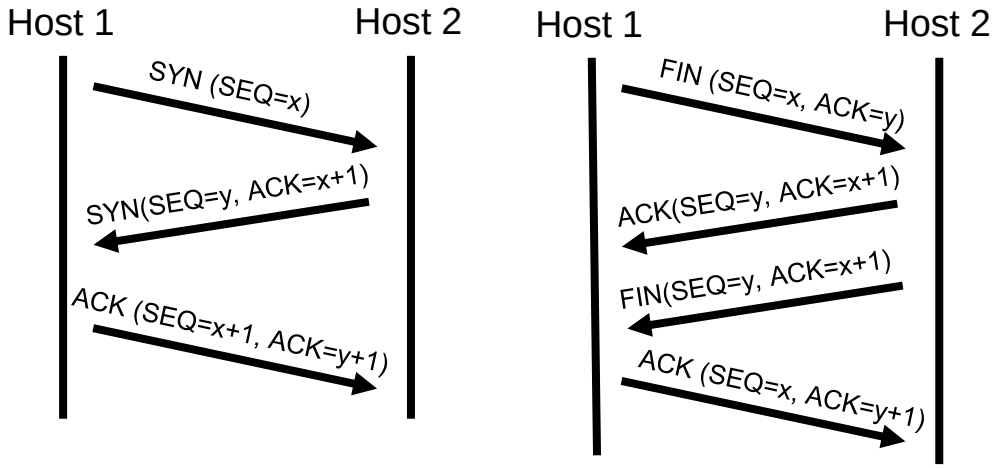
- **TCP (Transmission Control Protocol)**
  - Reliable (retransmission)
  - Ordered
  - Window – capacity of receiver
  - Checksum – 16bit (crc16)
  - TCP overhead: 20bytes
    - IP overhead: 20bytes
    - Ethernet frame: 18bytes (crc32)
- **TCP tries to correct errors; you don't need to worry…**
  -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



http://freesoft.org/CIE/Course/Section4/8.htm

Layer 4 - TCP

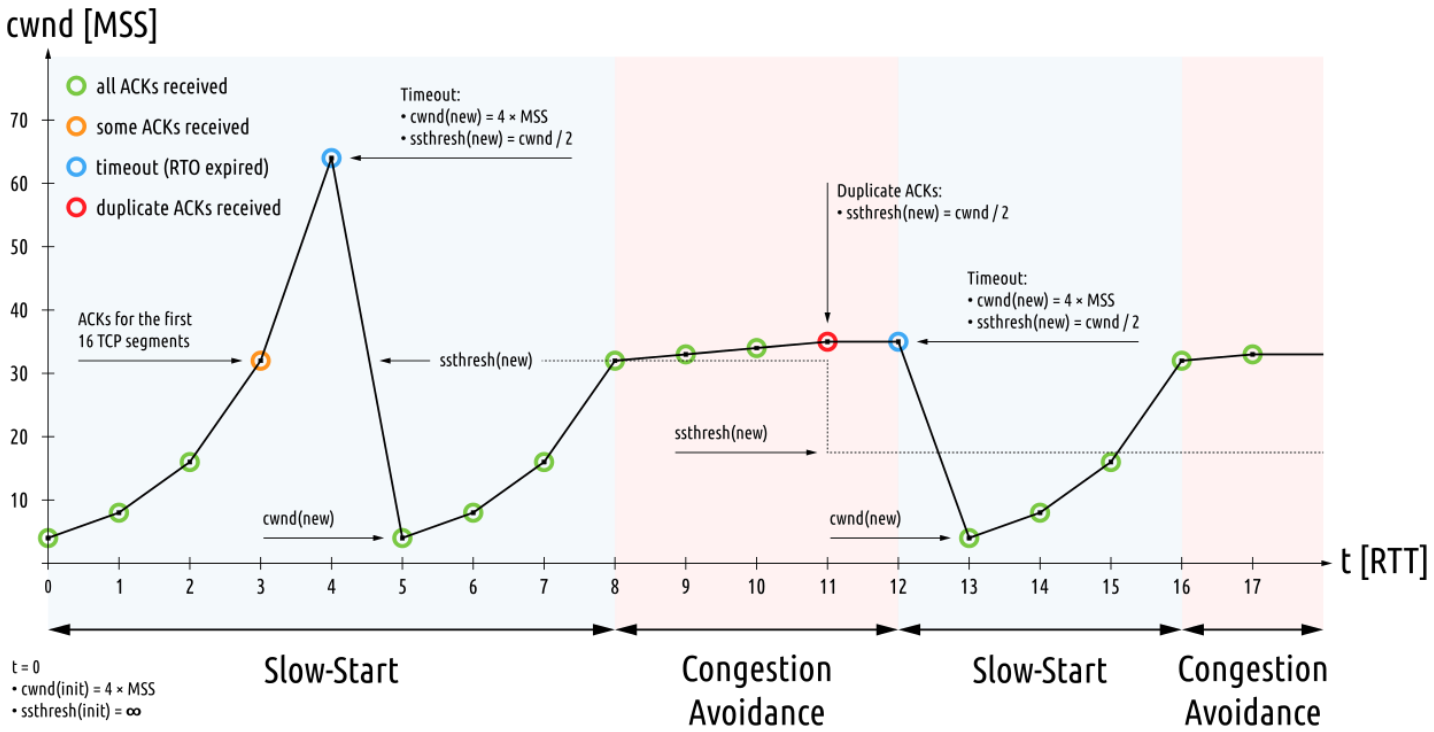
- **Connection establishment**
  - SYN, SYN-ACK, ACK (three way)
  - Initiates TCP session: initial sequence number is ~random
- **Connection termination**
  - FIN, ACK + FIN, ACK (three/four way)
  - 3-way handshake, when host 1 sends a FIN and host 2 replies with a FIN & ACK
- **Sequences and ACKs**
  - 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**
  - ACKs for last consecutive packets
  - 3 times same ACK → retransmit missing packets (fast retransmit)

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**



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

TCP/IP from an Application Developer View

- **Server in golang (repo)**
  - git clone https://github.com/tbocek/FS20.git
  - Download GoLand, or others
  - go build server.go → server
  - GOOS=linux GOARCH=amd64 go build server.go
  - Running on alpine → glibc/musl
    - Install glibc if you cross-compile on your machine (v2 and v3)
    - Or compile in Alpine…
- **Listening on TCP port 8081**
  - Return string in uppercase
- **Node.js version: apk add nodejs (+12MB)**
  - Download WebStorm, or other

```
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
    }
}
```

```
const net = require('net');
const rl = require('readline');
const server = net.createServer(onClientConnected);

server.listen(8081, 'localhost', function() {
    console.log('Launching server...');
});

function onClientConnected(sock) {
    const i = rl.createInterface(sock, sock);

    i.on('line', function(line) {
        console.log('Message Received: %s', line);
        sock.write(line.toUpperCase());
    });
};
```

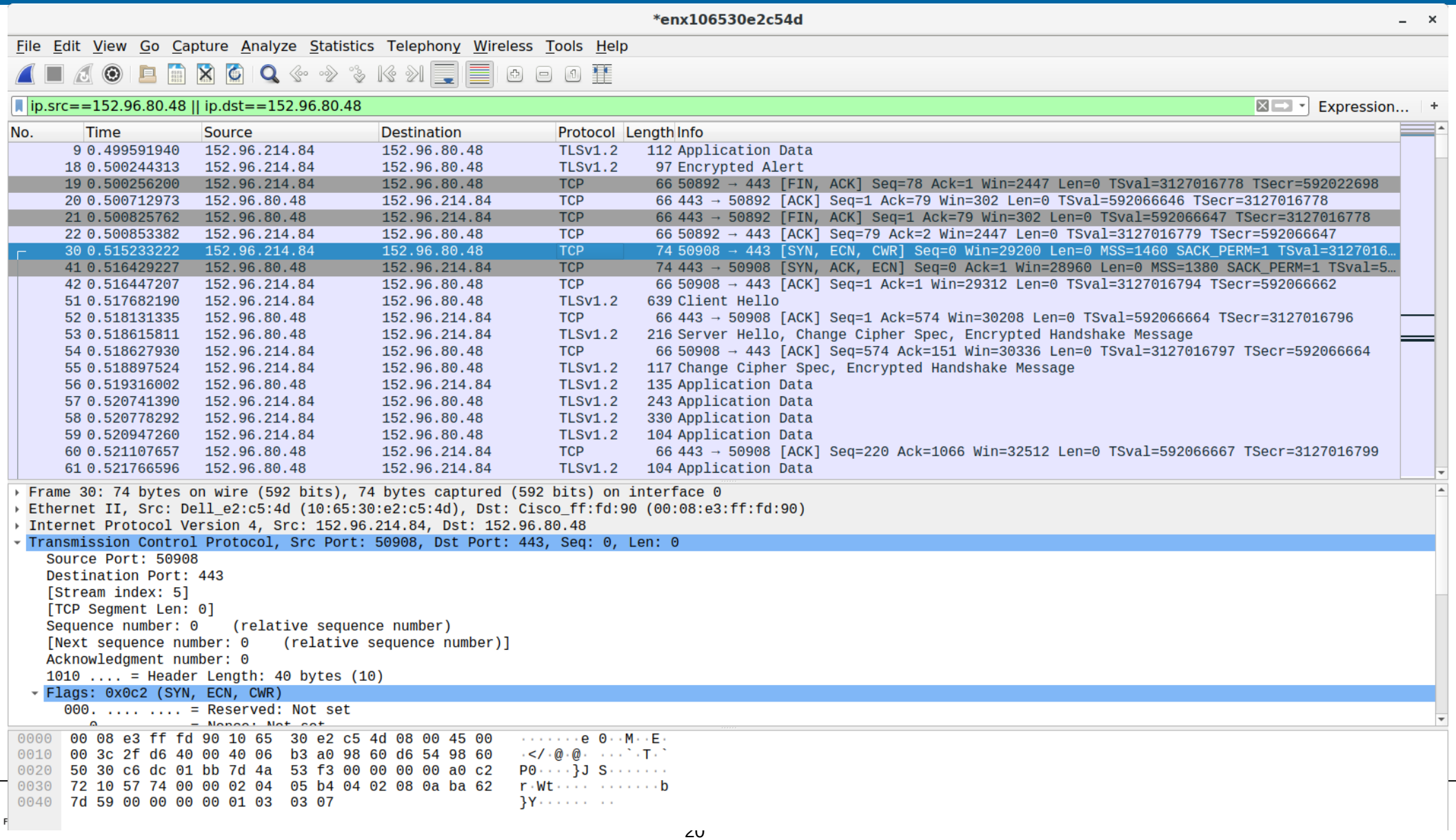
TCP/IP from an Application Developer View

- **Client in Java, IntelliJ IDEA, Eclipse, Netbeans**
- **Golang, node.js, Java in same directory**
  - Bad idea, for demo only
- **TCP/UDP is supported in any major language and interoperable with each other.**
- **Netcat**
  - Networking utility for reading from and writing to network connections using TCP or UDP
  - More complex scenarios (e.g., SCTP): socat
- **Challenge Task: user outside of VM**
  - Port mapping, 8888 to 80
- **TCP tries to correct errors; you don't need to worry… Sometimes, you need to worry…**

```
import java.io.*;
import java.net.*;

class TCPClient {
    public static void main(String argv[]) throws Exception {
        Socket clientSocket = new Socket("localhost", 8081);
        DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
        outToServer.writeBytes("5Anybody there?\n");
        String modifiedSentence = inFromServer.readLine();
        System.out.println("Client received from server: " + modifiedSentence);
        clientSocket.close();
    }
}
```

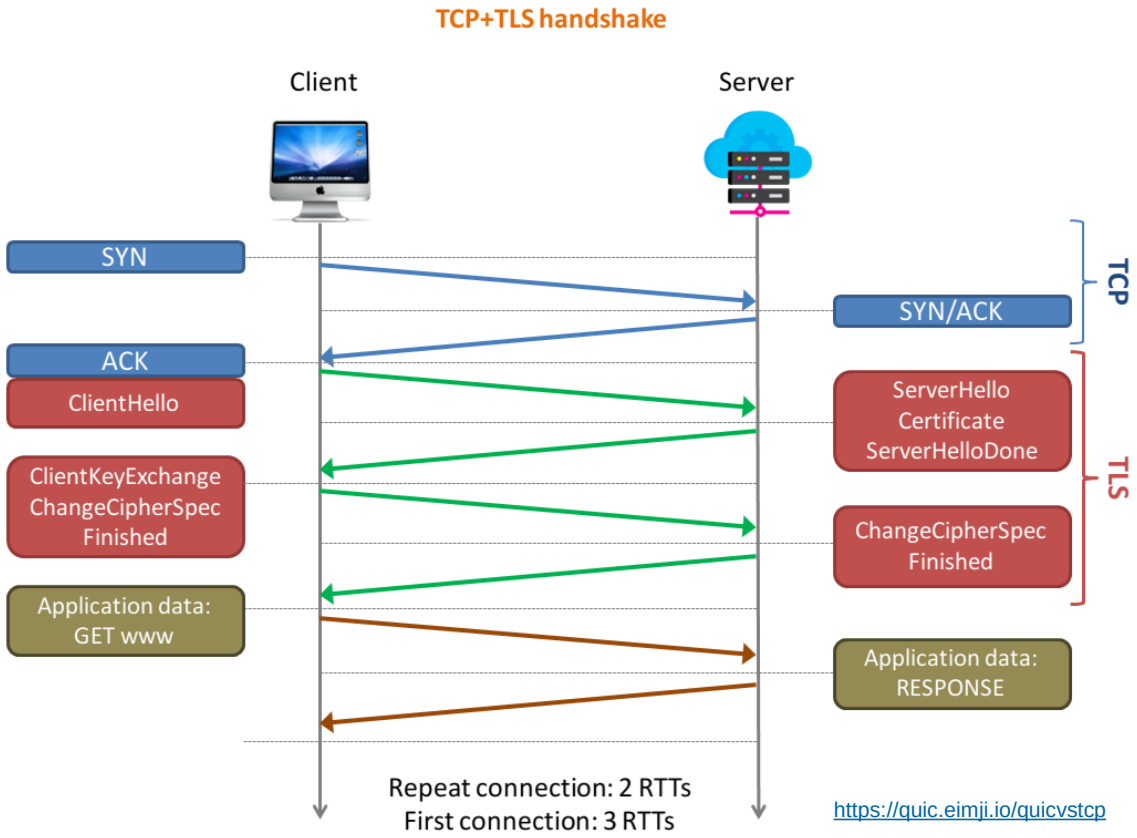
```
nc localhost 8081
nc -l -p 8081 -e awk '{print toupper($0)}'
```

Wireshark – sometimes needed when designing protocols



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



https://www.ibm.com/support/knowledgecenter/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm

Layer 4 – TCP + TLS

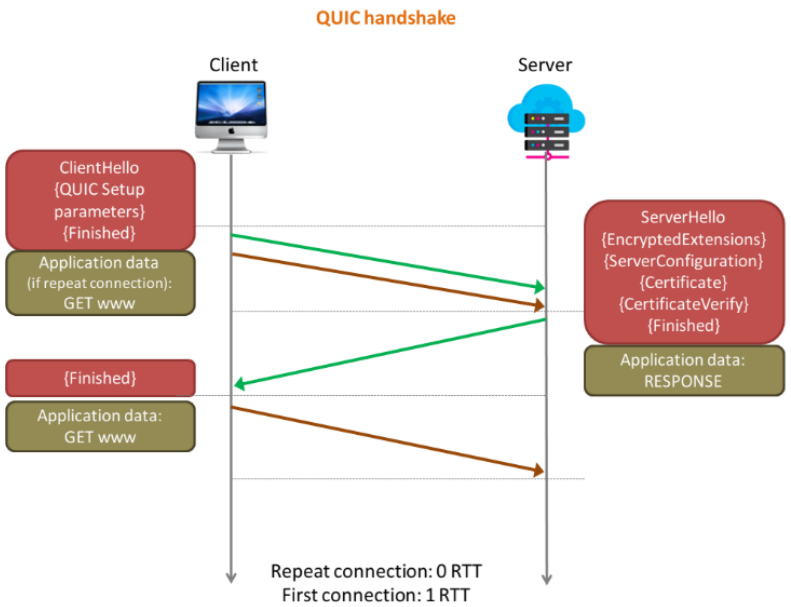
- **Ping to Australia: 329ms**
  - One way ~ 165ms
- **TCP + TLS handshake:**
  - 3RTT = 987ms! No data sent yet
- **TLS 1.3, finished Aug 2018**
  - 1 RTT instead of 2
  - 0 RTT possible, for previous connections, loosing perfect forward secrecy
  - [78% of browsers used already support it]


```
draft@schleppi: ping www.curtin.edu.au
File  Edit  View  Search  Terminal  Help
.com (52.64.39.142): icmp_seq=4 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=5 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=6 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=7 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=8 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=9 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=10 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=11 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=12 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=13 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=14 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=15 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=16 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=17 ttl=42 time=329 ms
```

https://www.thesslstore.com/blog/tls-1-3-handshake-tls-1-2/

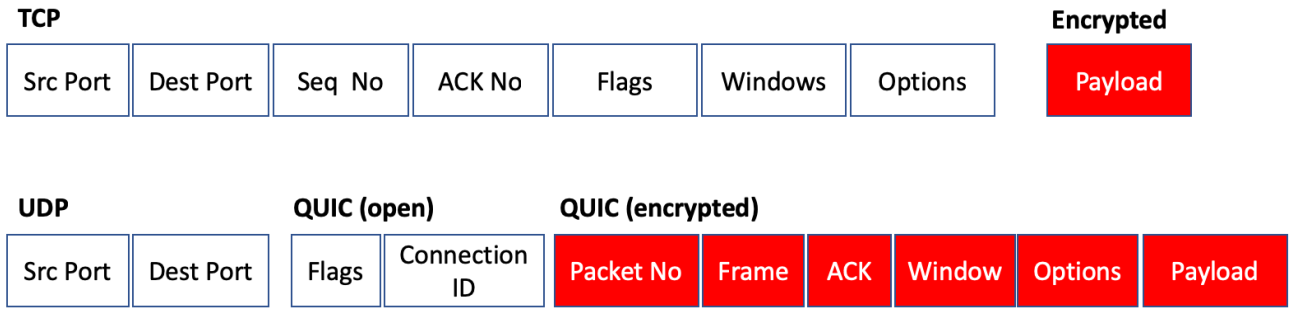
QUIC

- **QUIC: 1RTT (chrome example)**
  - For known connections: 0RTT
  - Built in security
  - “between 2.6 per cent and 9.1 per cent of traffic (3%)” - end-user 0%



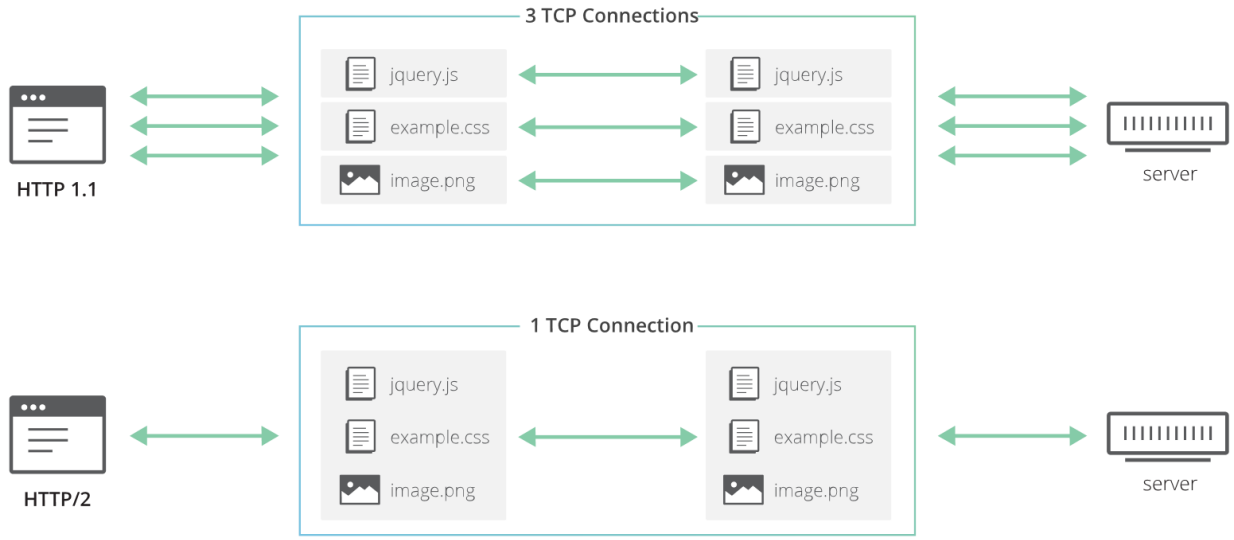
- **QUIC Standard, short header, long header**

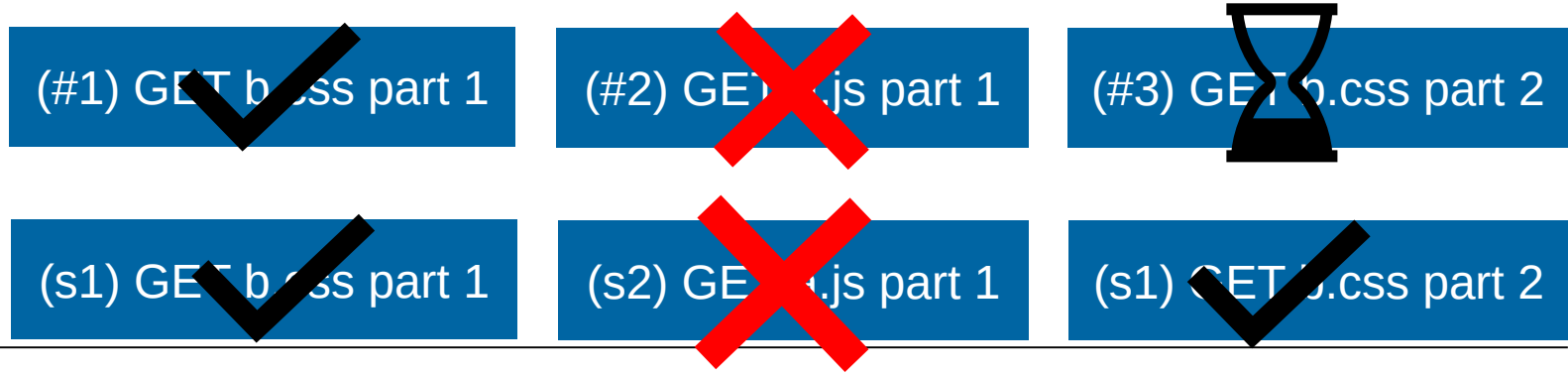
```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+
|1|1|T T|X X X X|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Version (32)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|DCIL(4)|SCIL(4)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Destination Connection ID (0..160)             ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Source Connection ID (0..160)                ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```



QUIC

- **Multiplexing in HTTP/2**
  - HTTP/1 → HTTP/2
- **HTTP/2: Head-of-line blocking**
  - One packet loss, TCP needs to be ordered, everything stops
  - 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



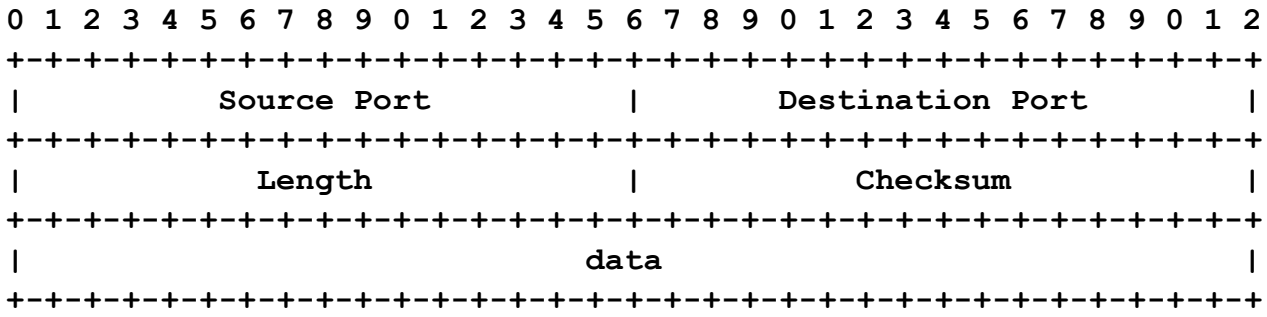


URL: b.socrative.com/student/

Room: HSR

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

UDP example

- **UDP Server (Java)**

```
import java.net.*;

class Server
{
    public static void main(String args[]) throws Exception
    {
        DatagramSocket serverSocket = new DatagramSocket(8081);
        byte[] receiveData = new byte[1024];
        while(true)
        {
            DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
            String s = new String( receivePacket.getData());
            System.out.println("Message Received: " + s);
        }
    }
}
```

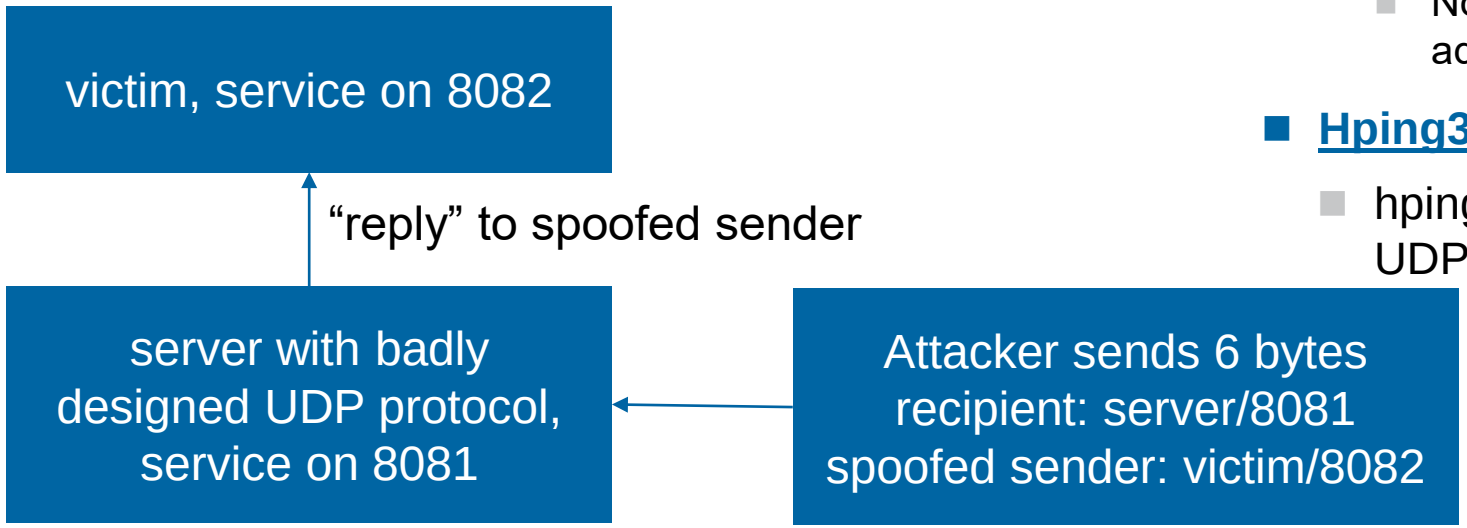
- **UDP Client (golang)**

```
package main

import (
        "net"
)

func main() {
        srv, _ := net.ResolveUDPAddr("udp","127.0.0.1:8081")
        local, _ := net.ResolveUDPAddr("udp", "127.0.0.1:0")
        conn, _ := net.DialUDP("udp", local, srv)
        defer conn.Close()
        conn.Write([]byte("5Anybody there?"))
}
```

- **nc –u localhost 8888**
  - From outside: port mapping!

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

Comparison – Transport Layer

TCP *

- Transport layer
- Connection oriented
- Reliable transfer
- Streams
- Guaranteed order
- Widely used – HTTP/1, HTTP/2
- Flow and congestion control
- Heavyweight
- Header size is 20 bytes
- Error checking and recovery

UDP *

- Transport layer
- Connection less
- Unreliable transfer
- Messages
- Unordered
- Widely used – DNS, HTTP/3
- No flow, congestion
- Lightweight
- Header size is 8 bytes
- Error checking, no recovery

SCTP *

- Transport layer
- Connection oriented
- Reliable transfer
- Messages
- User can choose
- WebRTC
- Flow and congestion control
- Heavyweight
- Common header is 12 bytes
- Error checking and recovery

URL: b.socrative.com/student/

Room: HSR

Questions?

Dr. Thomas Bocek
thomas.bocek@hsr.ch