



**OST**

Eastern Switzerland  
University of Applied Sciences

# Distributed Systems (DSy)

## Protocols 2

Thomas Bocek

15.03.2022

# Learning Goals

- Lecture 4 (Protocols, part 2)
  - How can new protocols improve latency?
  - What is an amplification attack?

# Wireshark – sometimes needed when designing protocols

- Decrypt TLS
- export SSLKEYLOGFILE=/tmp/keylogfile.txt

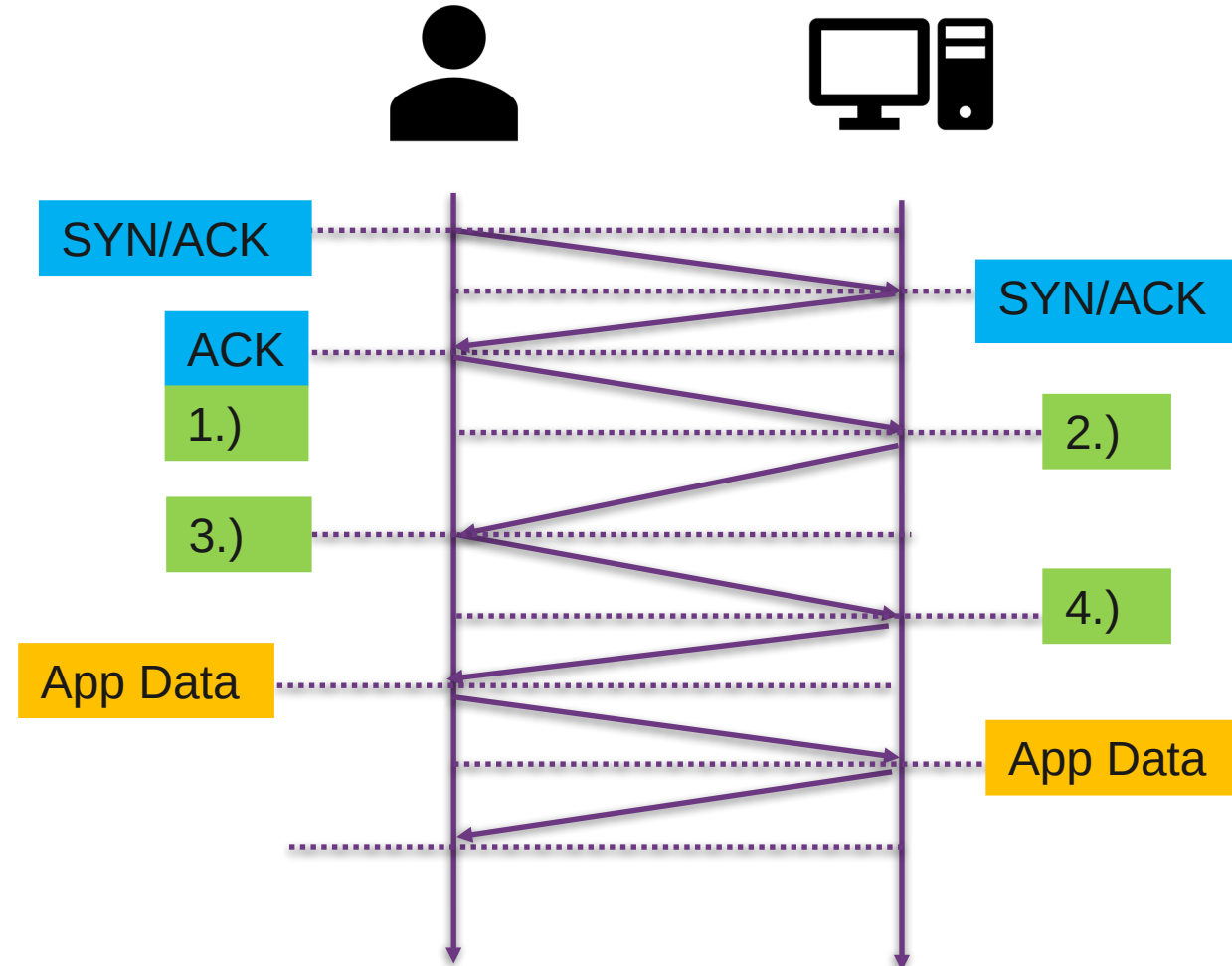
The screenshot shows the Wireshark interface with a network capture. The filter bar at the top displays the filter: `ip.src==152.96.80.48 || ip.dst==152.96.80.48`. The packet list pane shows several packets, with packet 30 selected. The packet details pane for packet 30 shows the following information:

- Frame 30: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
- Ethernet II, Src: Dell\_e2:c5:4d (10:65:30:e2:c5:4d), Dst: Cisco\_ff:fd:90 (00:08:e3:ff:fd:90)
- Internet Protocol Version 4, Src: 152.96.214.84, Dst: 152.96.80.48
- Transmission Control Protocol, Src Port: 50908, Dst Port: 443, Seq: 0, Len: 0
  - Source Port: 50908
  - Destination Port: 443
  - [Stream index: 5]
  - [TCP Segment Len: 0]
  - Sequence number: 0 (relative sequence number)
  - [Next sequence number: 0 (relative sequence number)]
  - Acknowledgment number: 0
  - 1010 ... = Header Length: 40 bytes (10)
  - Flags: 0x0c2 (SYN, ECN, CWR)
    - 000. .... = Reserved: Not set

The packet bytes pane at the bottom shows the raw data of the packet, including the Ethernet II header, IP header, and the beginning of the TCP header.

# Layer 4 – TCP + TLS

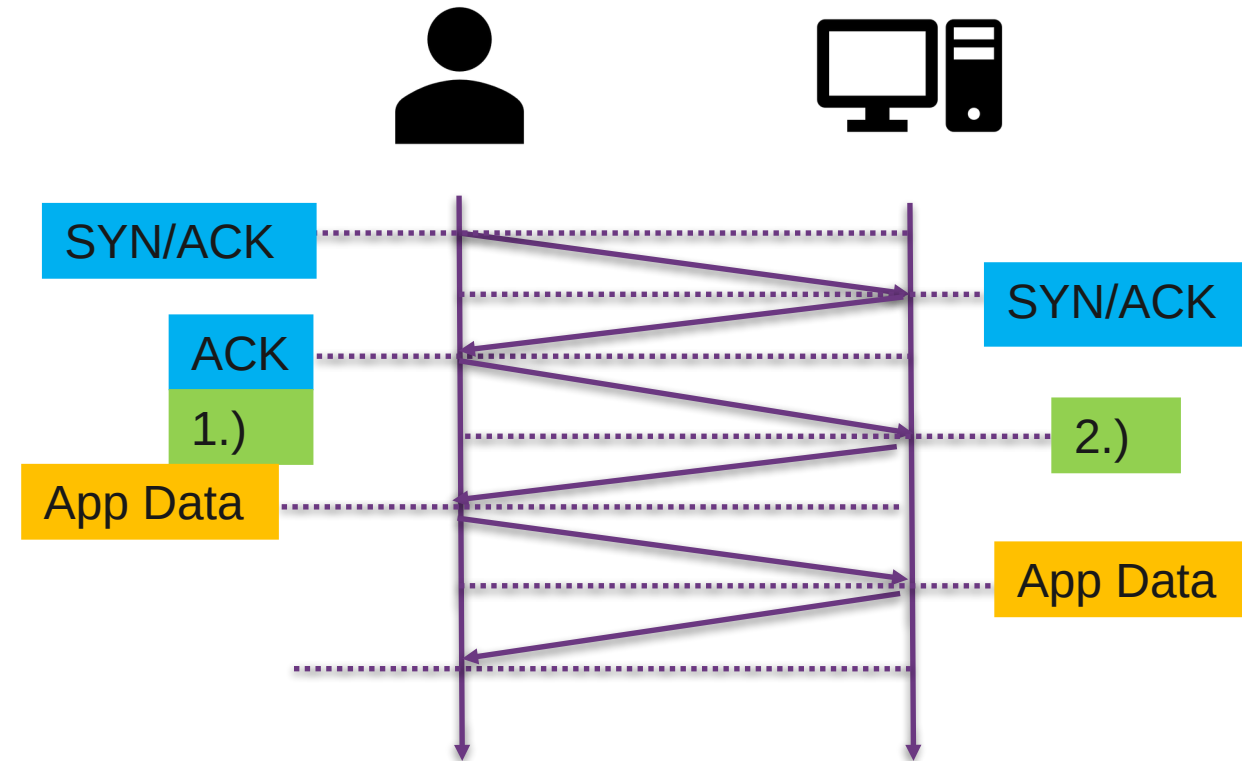
- Security: Transport Layer Security (TLS)
  1. "client hello" lists cryptographic information, TLS version, ciphers/keys
  2. "server hello" chosen cipher, the session ID, random bytes, digital certificate (checked by client), optional: "client certificate request"
  3. Key exchange using random bytes, 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



```
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
```

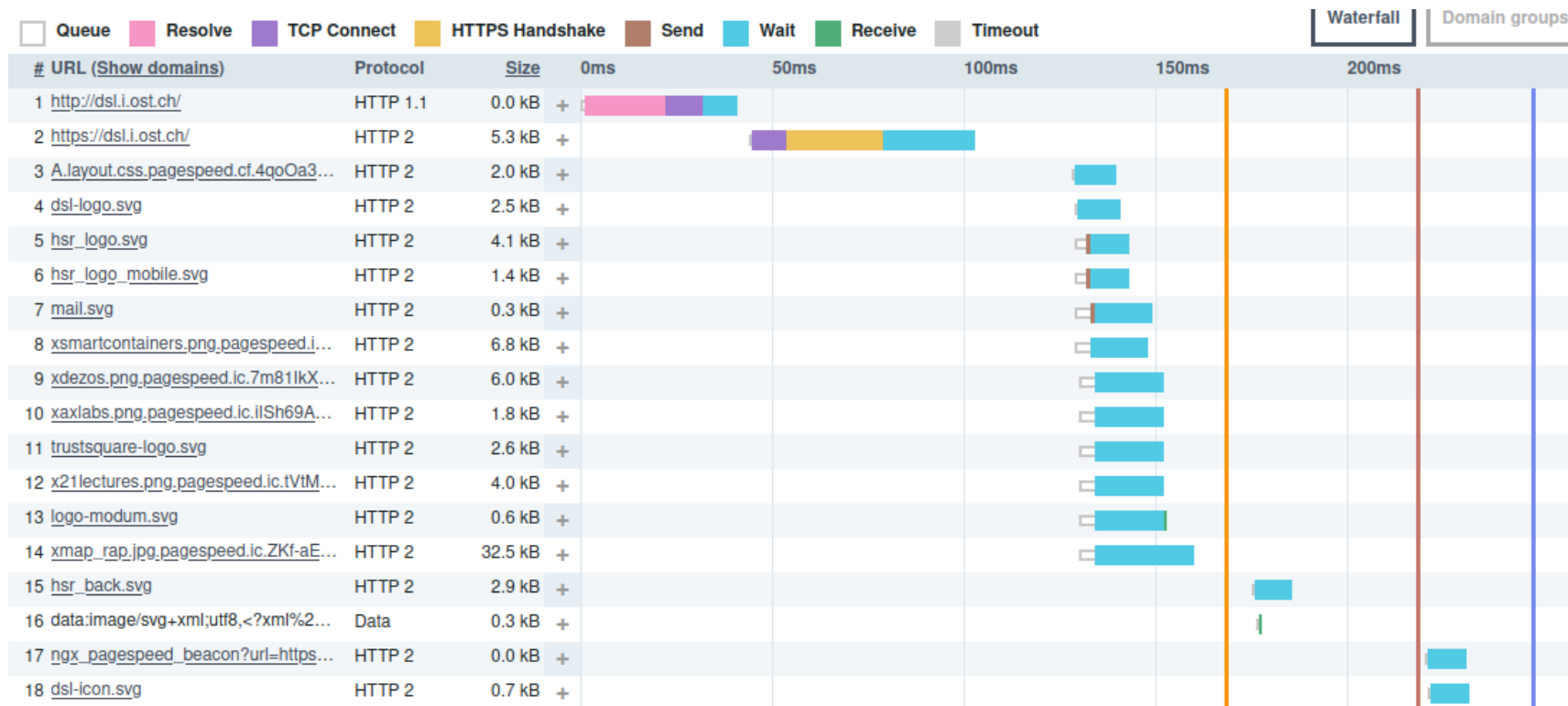
## Layer 4 – TCP + TLS

- 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
    - 1.) Client Hello, Key Share
    - 2.) Server Hello, key Share, Verify Certificate, Finished
  - 0 RTT possible, for previous connections, losing perfect forward secrecy
- 90% of browsers used already support it



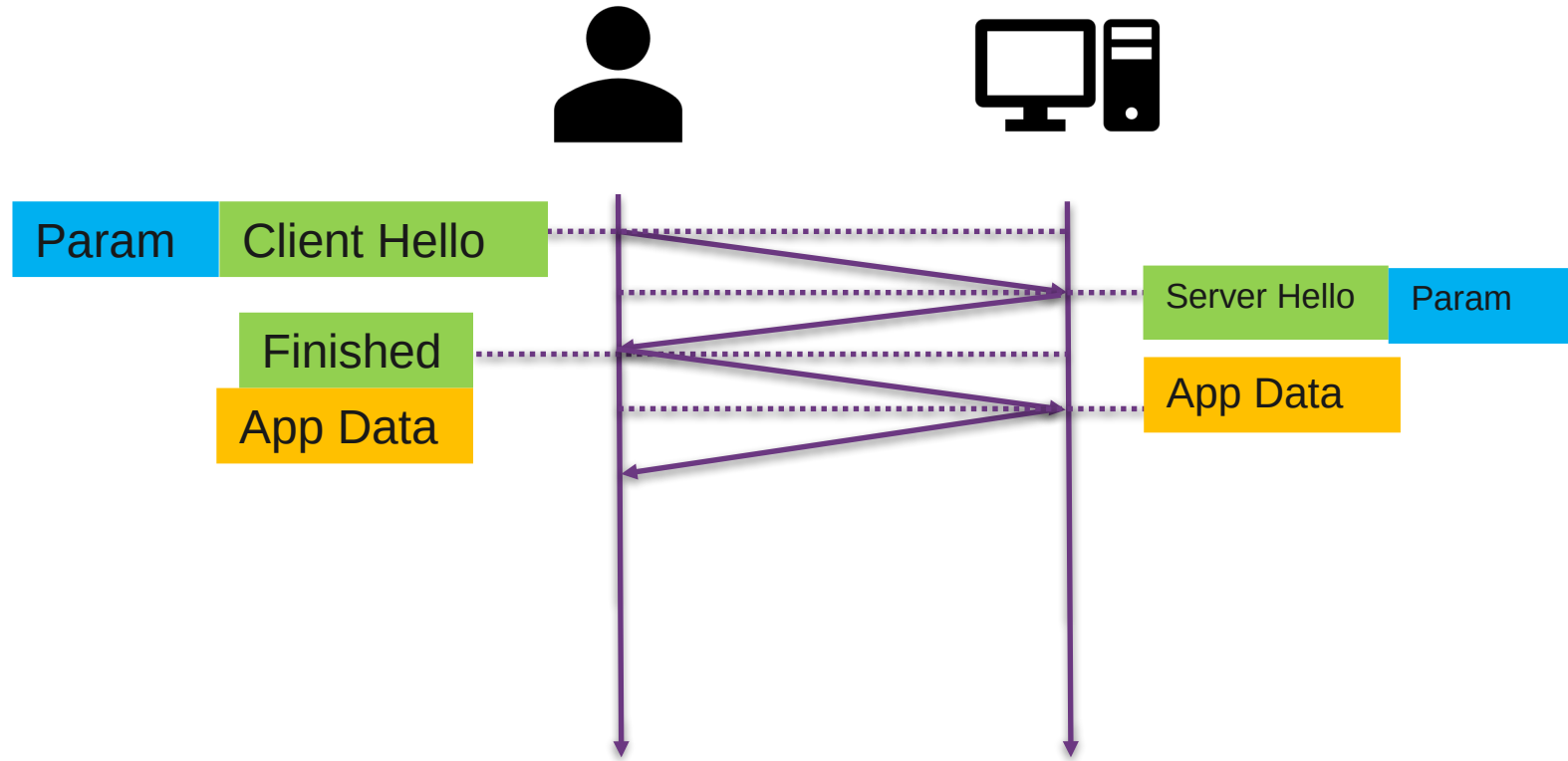
# Layer 4 – TCP + TLS

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



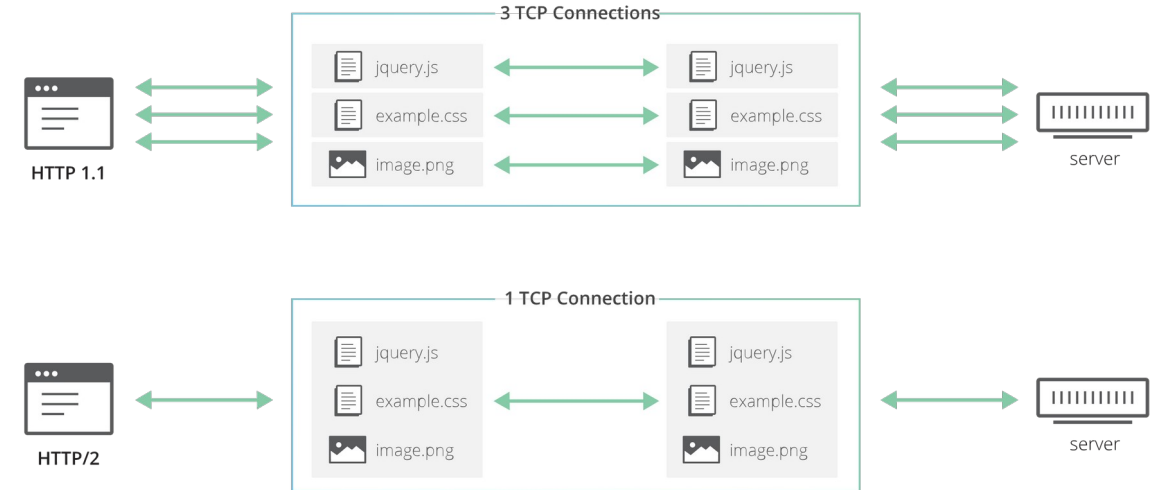
# QUIC / HTTP3

- QUIC: 1RTT (chrome example)
  - For known connections: 0RTT
  - [Built in security](#)
  - “Google's 'QUIC' TCP alternative slow to excite anyone outside Google” [link] ([7%](#), [25%](#))
    - [Facebook](#)
    - [Cloudflare](#)
  - [Can I use \(72.5%\)](#)
- Example Australia: from 987ms to 329ms



# 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/>





# 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

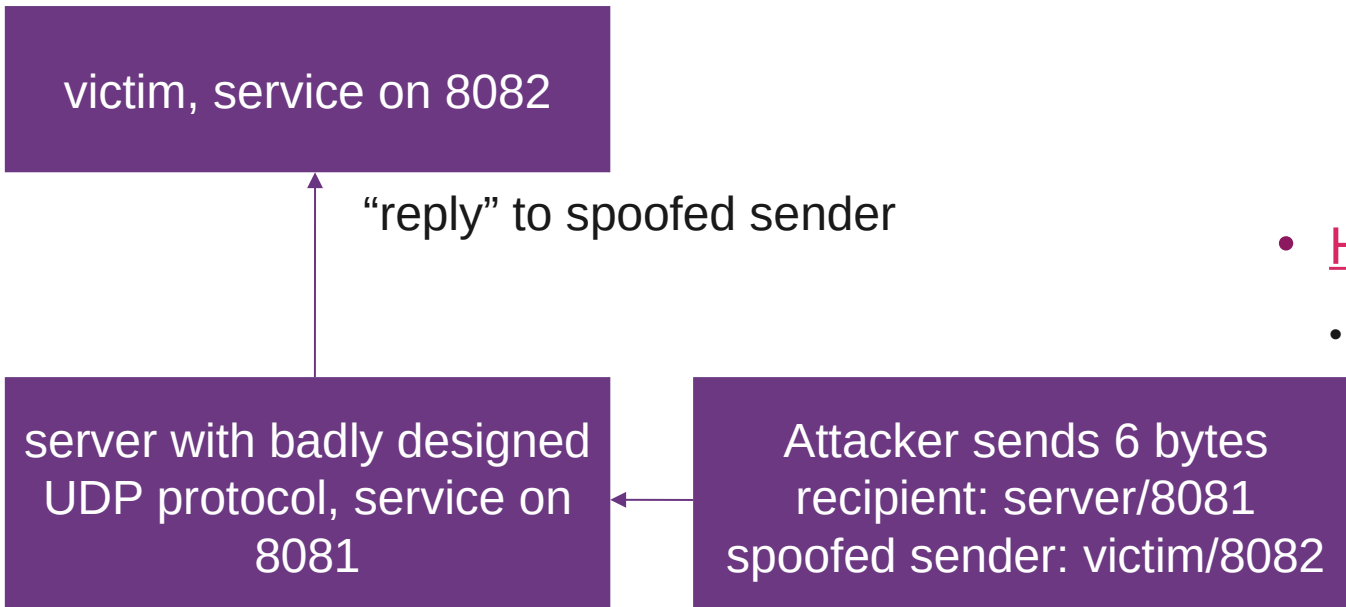
import (
    "net"
)

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 8081

# 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