FS19

PROTOCOLS

T. Bocek

thomas.bocek@hsr.ch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Agenda

- **Admin**
- **Distributed Systems in the News**
- **Layers / Protocols**
- **Layer 4 - TCP/IP, UDP, QUIC**
- **NAT Issues**

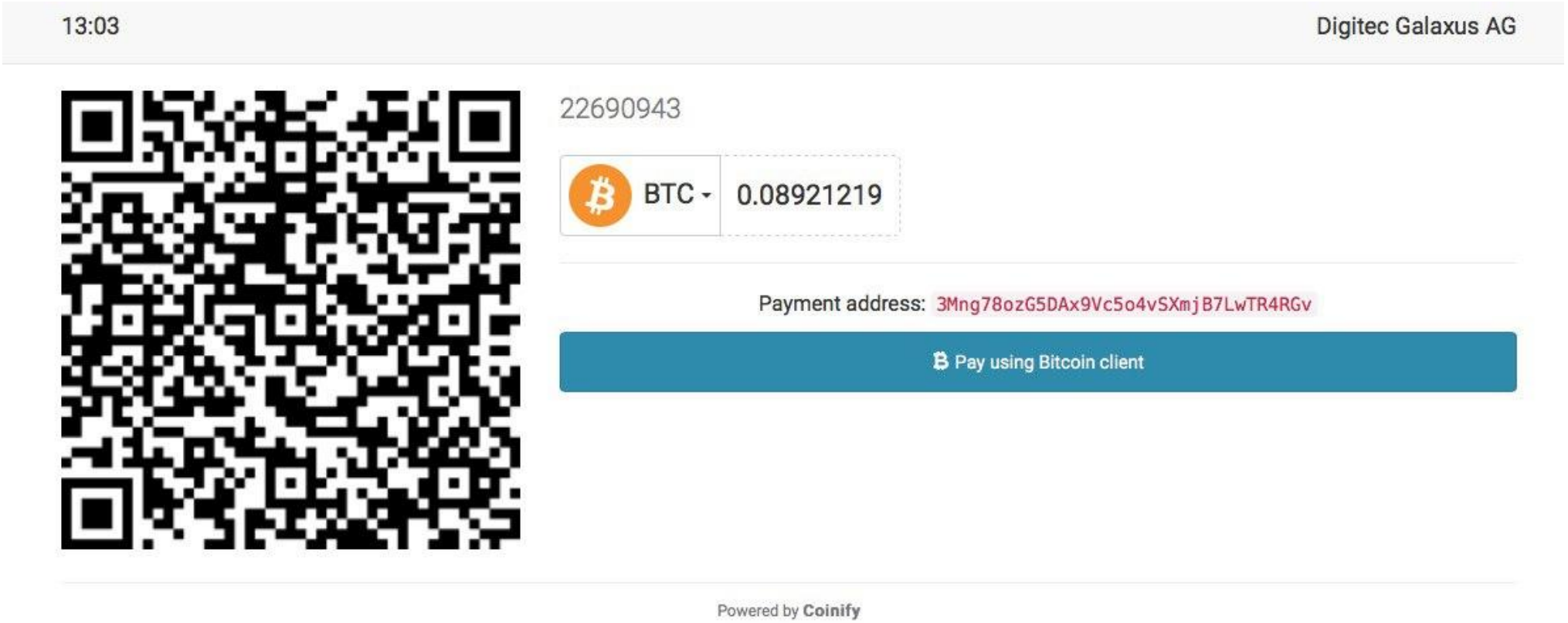
Admin - Exercices / Exam

- **Exam questions will be in English – if this is a problem, tell us now!**
- **Exercises will be practical**
  - 10.4. No exercises (next week)
  - 23.4. No lecture (in three weeks)
  - 24.4. exercises: Q&A (in three weeks)
- **Distributed Systems in the News**

Distributed Systems - In the *News*

- **19.03.2019: Digitec Galaxus akzeptiert ab sofort Bitcoin und Co. – das musst du wissen**
  - «Paukenschlag im Schweizer E-Commerce: Der grösste Onlinehändler des Landes führt Kryptowährungen als Zahlungsmittel ein – und macht sie damit massentauglich. Hier sind die wichtigsten Fragen und Antworten.»
    - Bitcoin (BTC), Bitcoin Cash (BCH), Bitcoin SV (BSV), Ethereum (ETH), Ripple (XRP), Binance Coin (BNB), Litecoin (LTC), TRON (TRX), OmiseGo (OMG), NEO (NEO)
  - Fixed rate for 15 min, 1.5% fee, coinify
- **22.03.2019: You Can Now Donate to the Tor Project in 9 Different Cryptocurrencies**
  - Augur (REP), Bitcoin (XBT), Bitcoin Cash (BCH), Dash (DASH), Ether (ETH), Litecoin (LTC), **Monero** (XMR), Stellar Lumen (XLM), **Zcash**
- **22.03.2019: Three protocols and a future of the decentralized internet**
  - “power to be redistributed from the hands of large centralized corporate interests and back into the hands of individual people and small groups”
  - Secure ScuttleButt (SSB)
    - P2P, friends of friends, message passing
  - ActivityPub (W3C)
    - Federation, web protocol, coordinating messages
  - Dat, P2P, sharing large files
  - “how can we have a Google Drive style system where a bunch of people can just connect to each other's computers and share files that way, removing the giant tech company in the middle?” → (BT sync) Resilio

Distributed Systems - In the *News*



- **Not payed with crypto, ~10CHF fees (3%, exchange rate)**

Distributed Systems - In the *News*

- **29.03.2019 Bitcoin Price Hits 5-Week High With Chart Echoing 2015 Pre-Rally Pattern**
  - Here we go again ☺
- **02.04.2019 Bitcoin Price Jumps to 4-Month High Above $4,900**
- **01.04.2019 Police Freeze Bank Accounts, Seize Luxury Cars in Probe of $22 Million ICO Promoter Vanbex**
  - 2017 initial coin offering (ICO) raised $22 million
  - Token sale: FUEL
  - "Vanbex told investors that the token would be usable in a forthcoming smart contract system called Etherparty"
  - "No intention to build product, but use funds for own personal benefit." – no criminal charges filed

- **Recently discovered (13.11.2018): Let's Take a Crack at Understanding Distributed Consensus**
  - From "What is a Distributed" System to "Proof-of-Work"

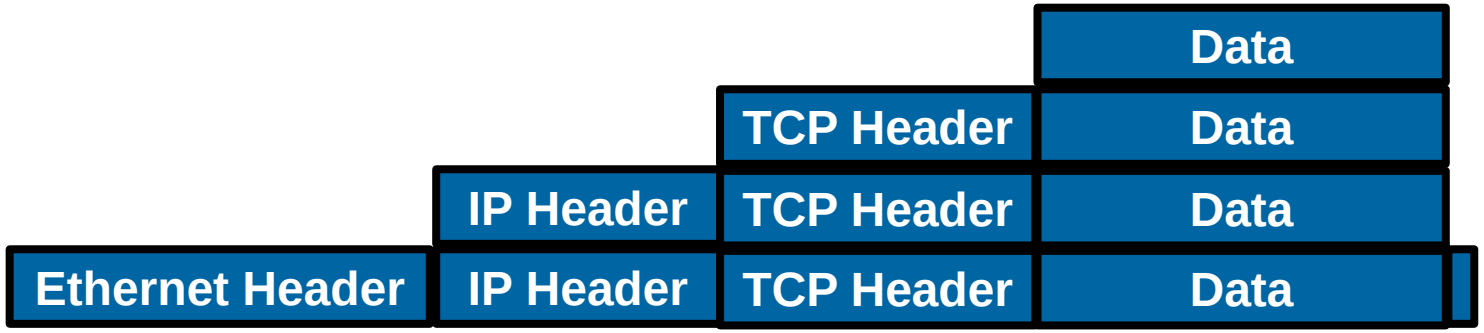
Goals

- **I know what a layer/protocol is**
- **I know the latencies of protocols**
- **I know why NAT was introduced and its advantages and disadvantages**
- **I can punch holes and can configure NAT-PMP via a UDP protocol**

Networking: Layers

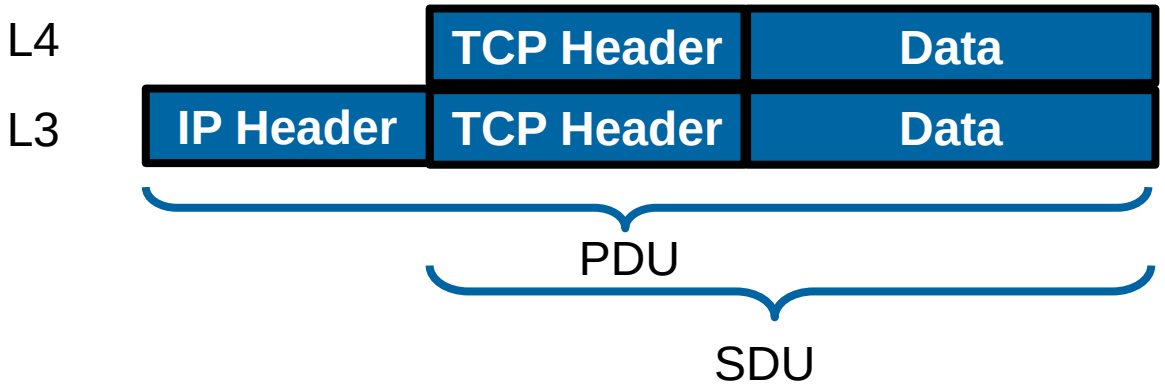
- **Networking: Each vendor had its own proprietary solution - not compatible with another solution**
- **Nowadays most vendors build compatible networks hardware/software from different vendors**
- **Goal of layers: interoperability**
- **1984: ISO 7498 - The Basic Reference Model for Open Systems Interconnection**

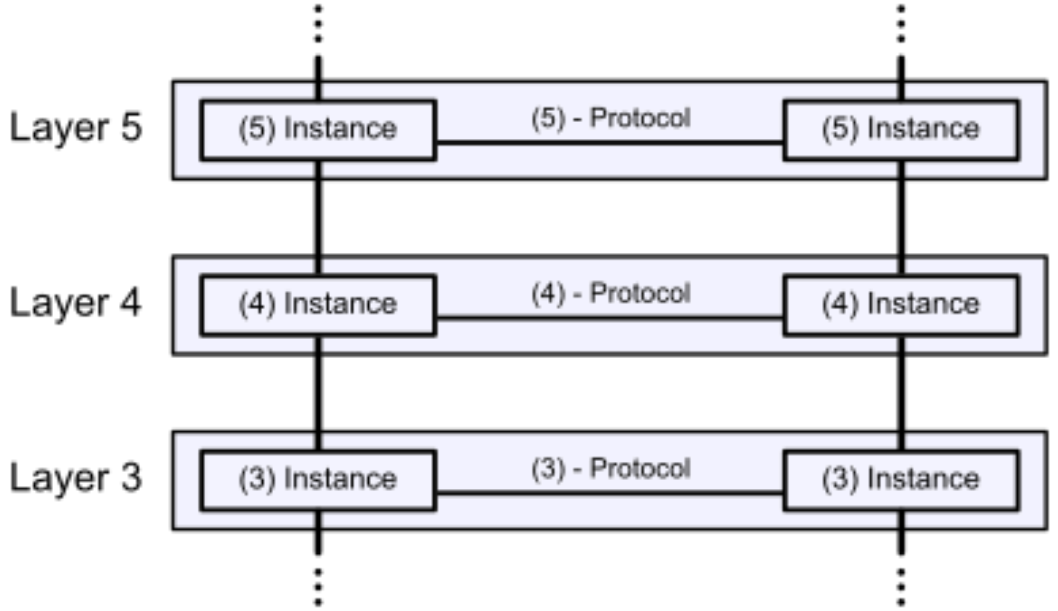
OSI model	"Internet model"
Application	Application
Presentation	
Session	
Transport	Transport
Network	Internet
Data link	Link
Pysical	



Layer Abstraction

- **Protocols enable an entity to interact with a entity 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

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

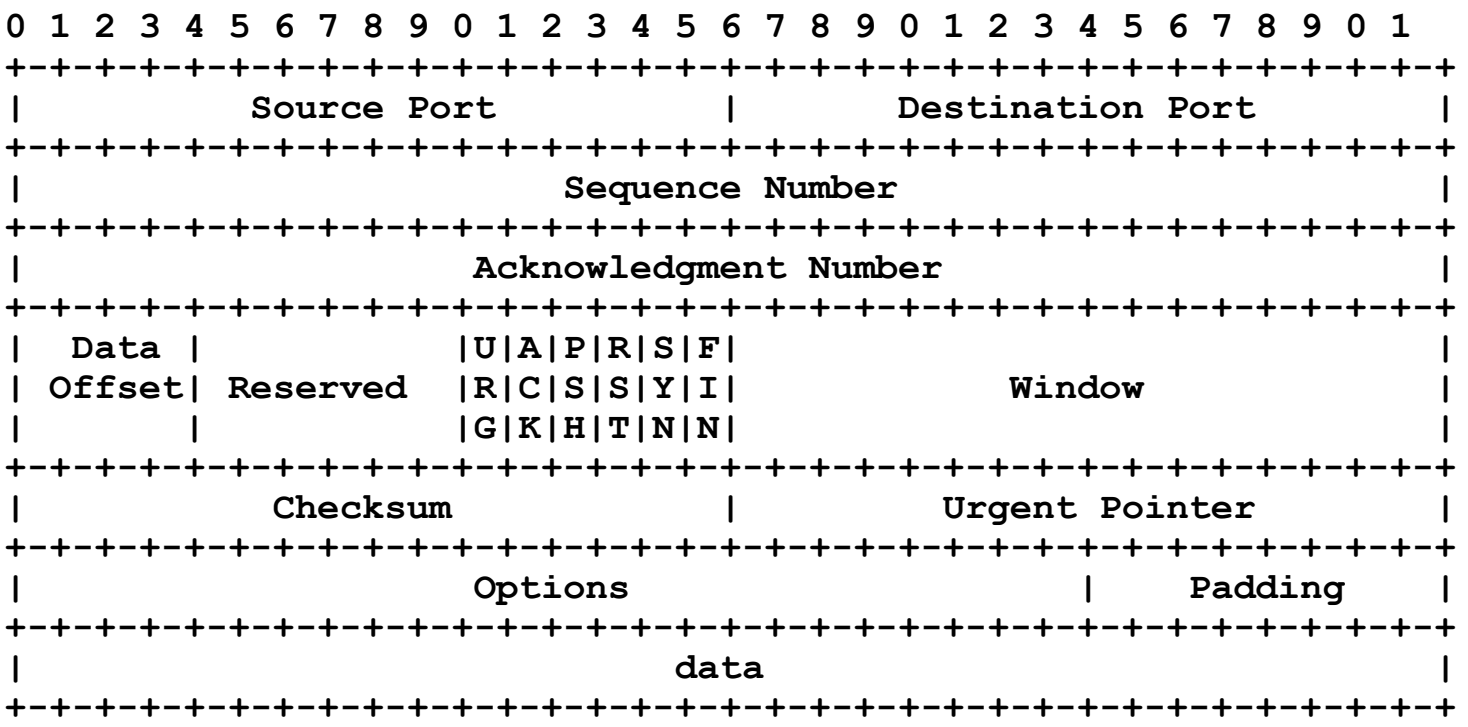
https://en.wikipedia.org/wiki/Internet_protocol_suite

URL: b.socrative.com/student/

Room: HSR

Layer 4 - Transport

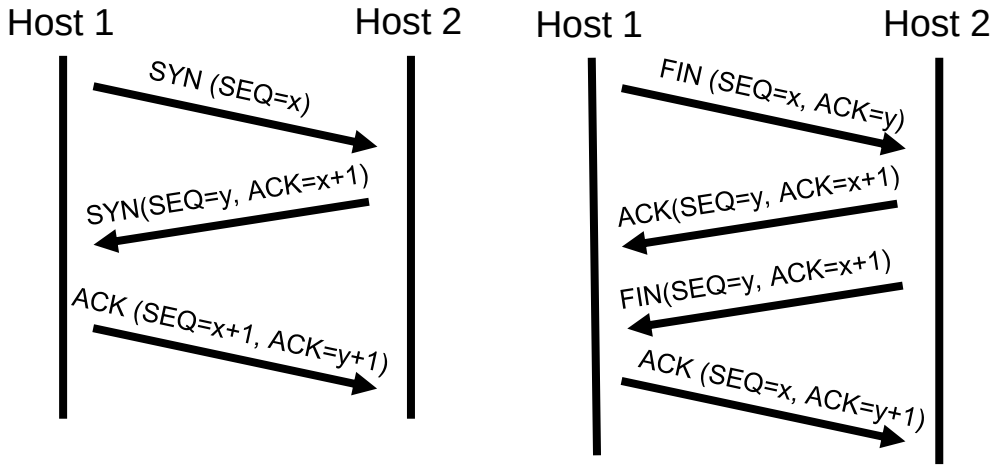
- **TCP (Transmission Control Protocol)**
  - Reliable (retransmission)
  - Ordered – unordered?
  - Check summed – 16bit
- **In the news (25.1.2019): Undersea cable damage wipes out most Internet access in Tonga islands**
  - Internet connections were lost on the country's more than 170 islands, international calls wouldn't go through and credit card payments couldn't be processed.
  - Restored after 13 days, relaying on satellite communication



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

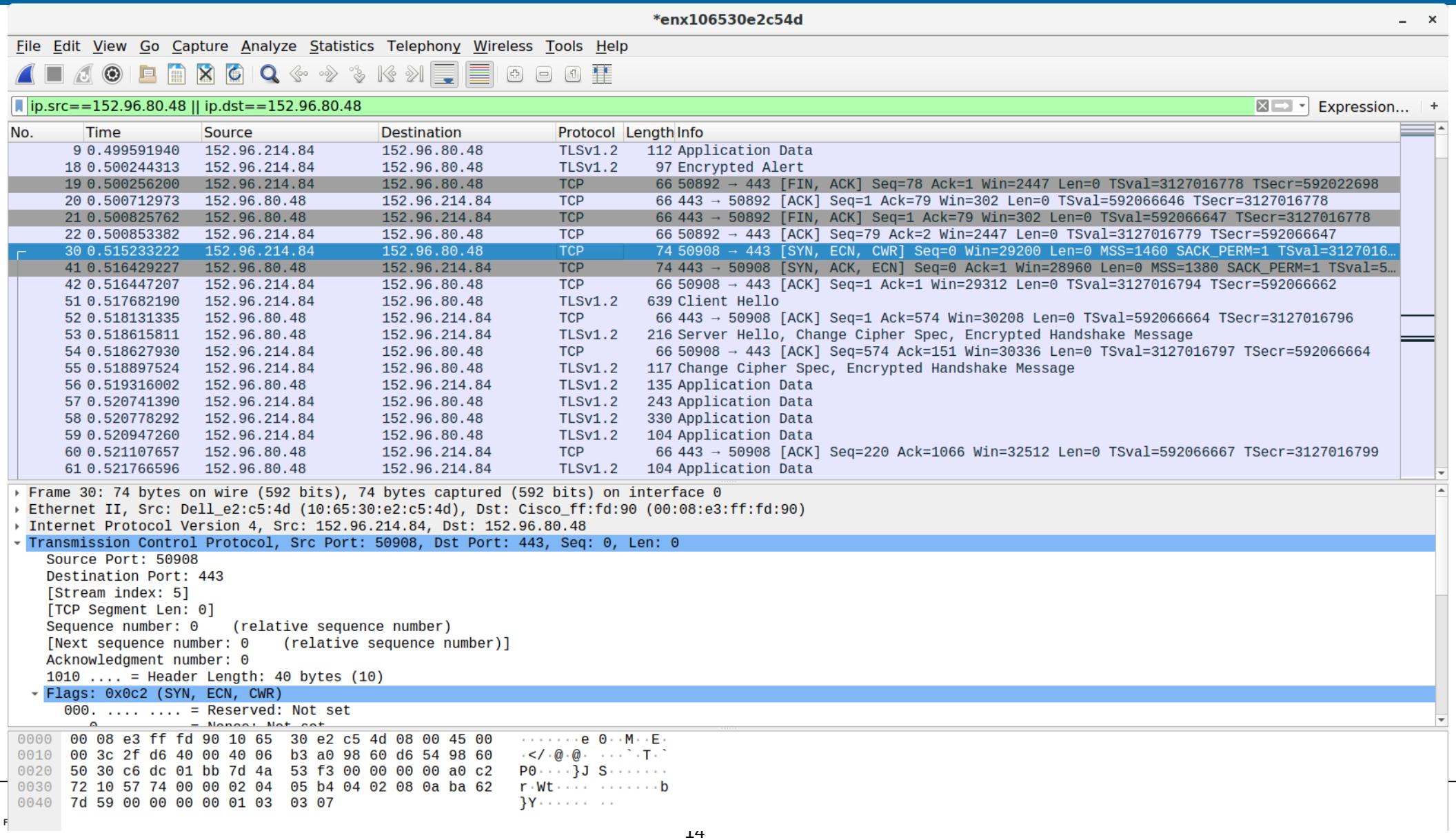
Layer 4 - TCP

- **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:



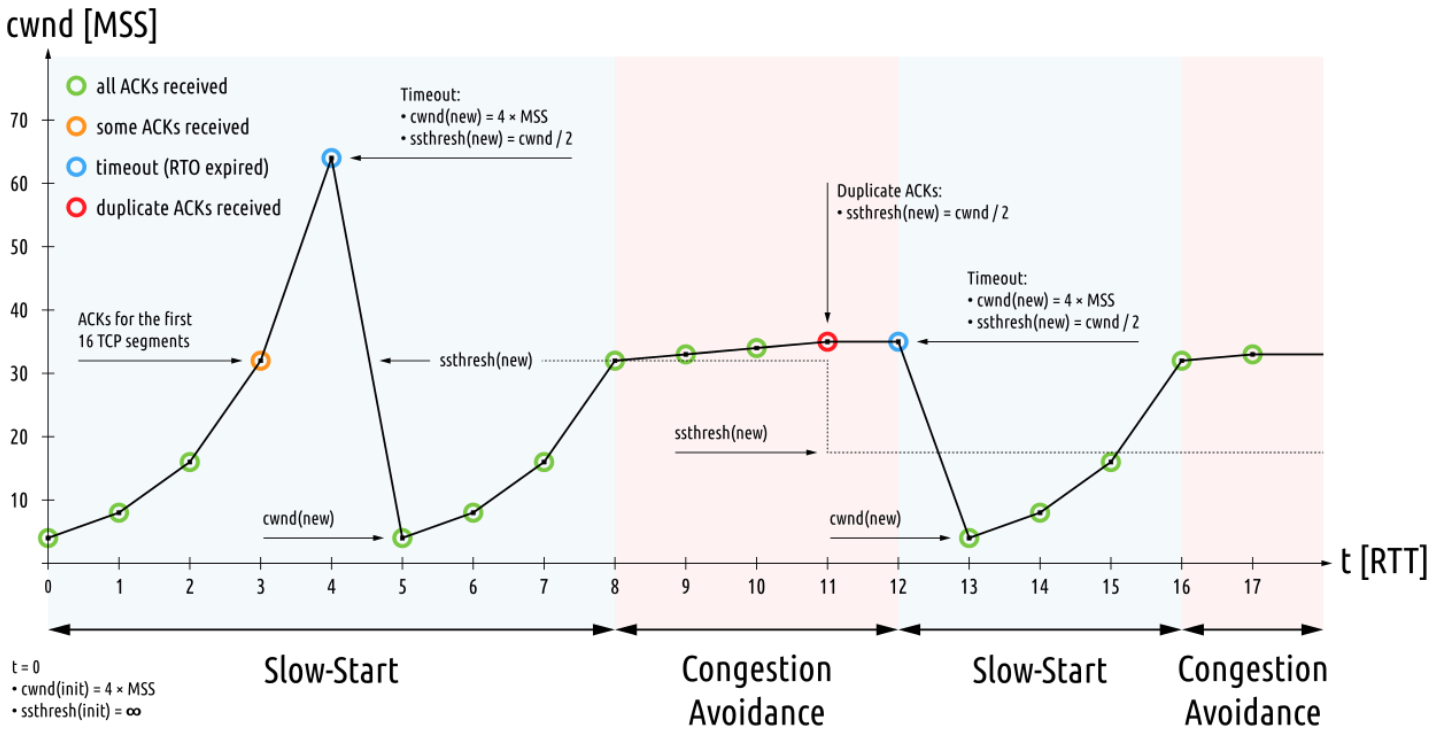
- **Retransmission timeout**
  - If no ACK is received aftert timout (e.g. 2xRTT), resend.
- **Duplicate cumulative acknowledgements, selective ACK**
  - ACKs for last consecutive packets
  - 3 times same ACK → retransmit (fast retransmit)

Wireshark – sometimes needed when designing protocols



Layer 4 - TCP

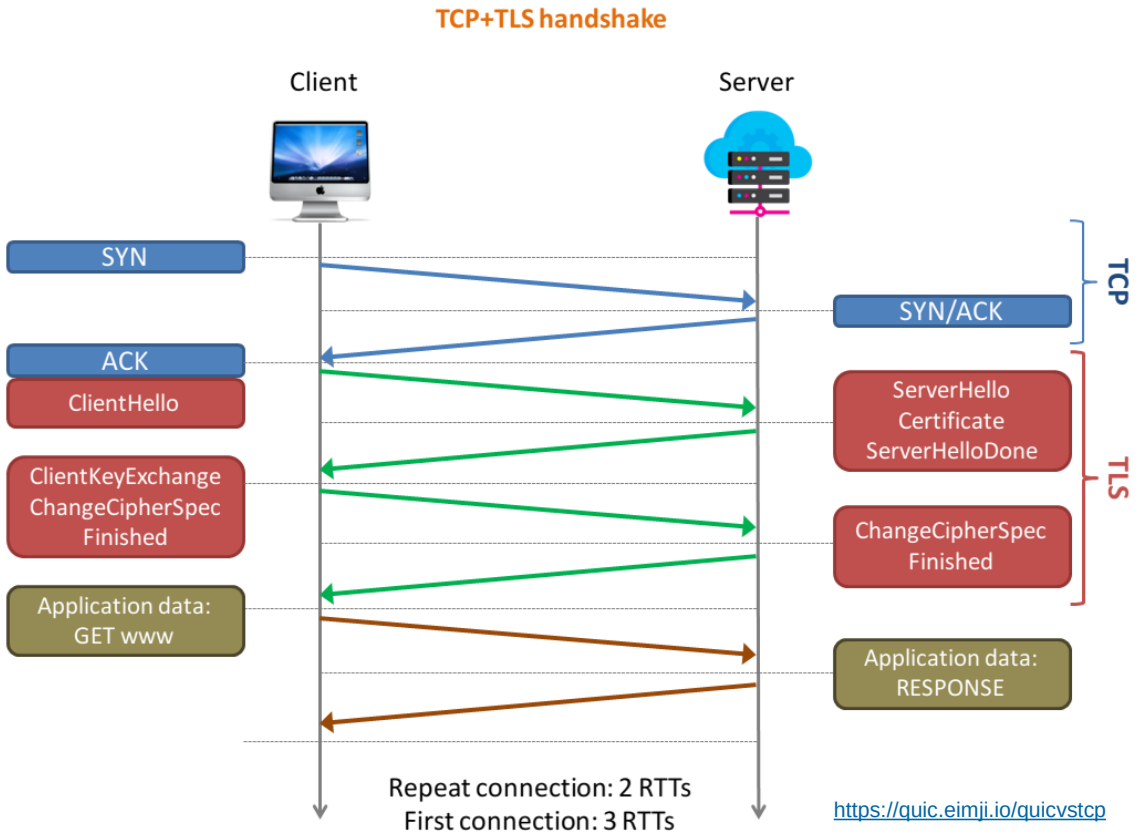
- **Flow control**
  - Sender is not overwhelming a receiver
  - Back pressure
  - Sliding window:
    - Receiver specifies the amount of additionally received data in bytes that can be buffered
    - Sender up to that amount of data before ACK
- **Congestion control**
  - slow-start
  - congestion avoidance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

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

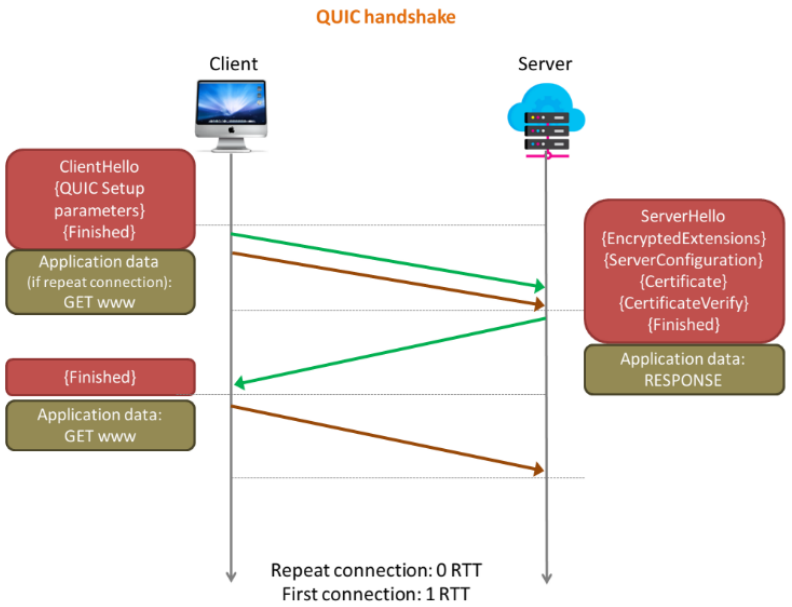
- **Ping to Australia: 329ms**
  - One way ~ 165ms
- **TCP + TLS handshake:**
  - 3RTT = 987ms! No data sent yet
- **TLS 1.3, finished Aug 2018**
  - 1 RTT instead of 2
  - 0 RTT possible, for previous connections, loosing perfect forward secrecy
  - 67% of browsers used already support it
- **ETS, (eTLS), disable forward secrecy**
  - ETA instead DPI

```
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/

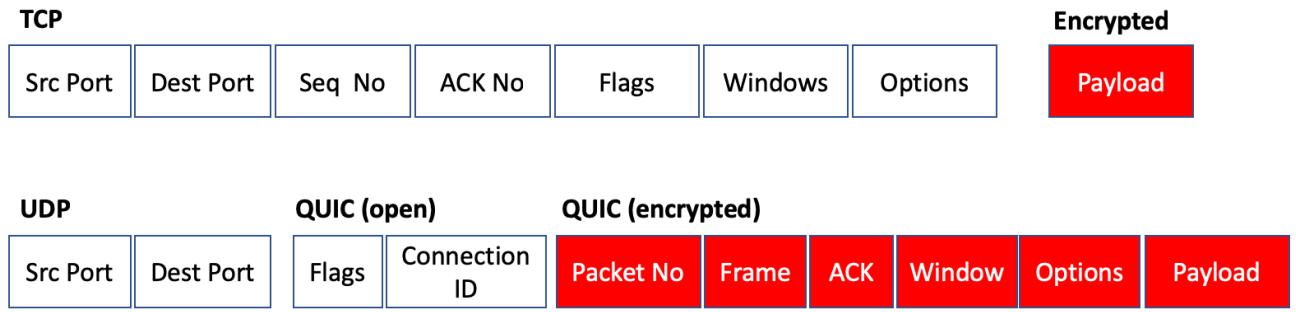
QUIC

- **TCP for short lived connections?**
  - Bad idea
  - TIME_WAIT → SO_LINGER = 0
- **QUIC: 1RTT (chrome example)**
  - Built in security
  - For known connections: 0RTT
  - "between 2.6 per cent and 9.1 per cent of traffic (2.1%)"



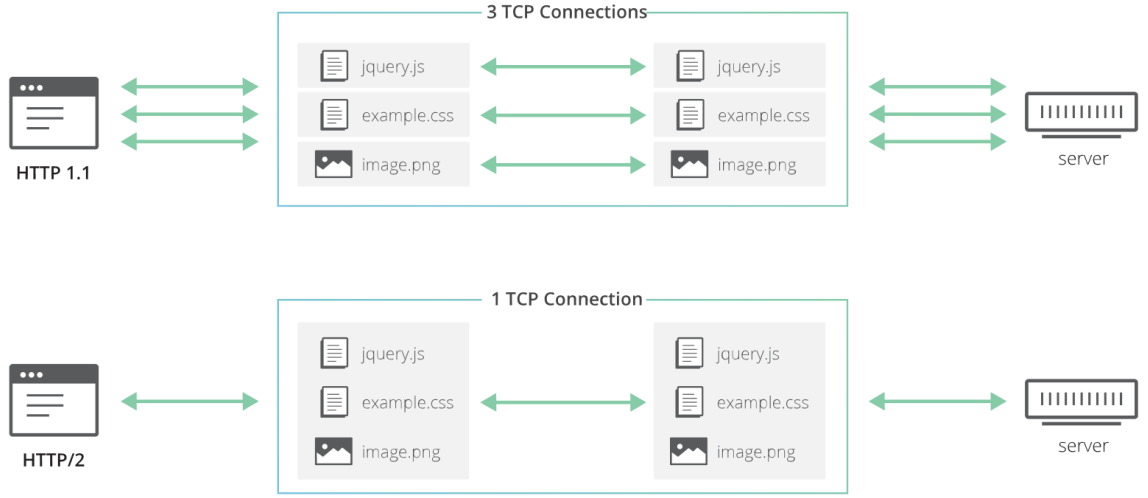
- **QUIC Standard, short header, long header**

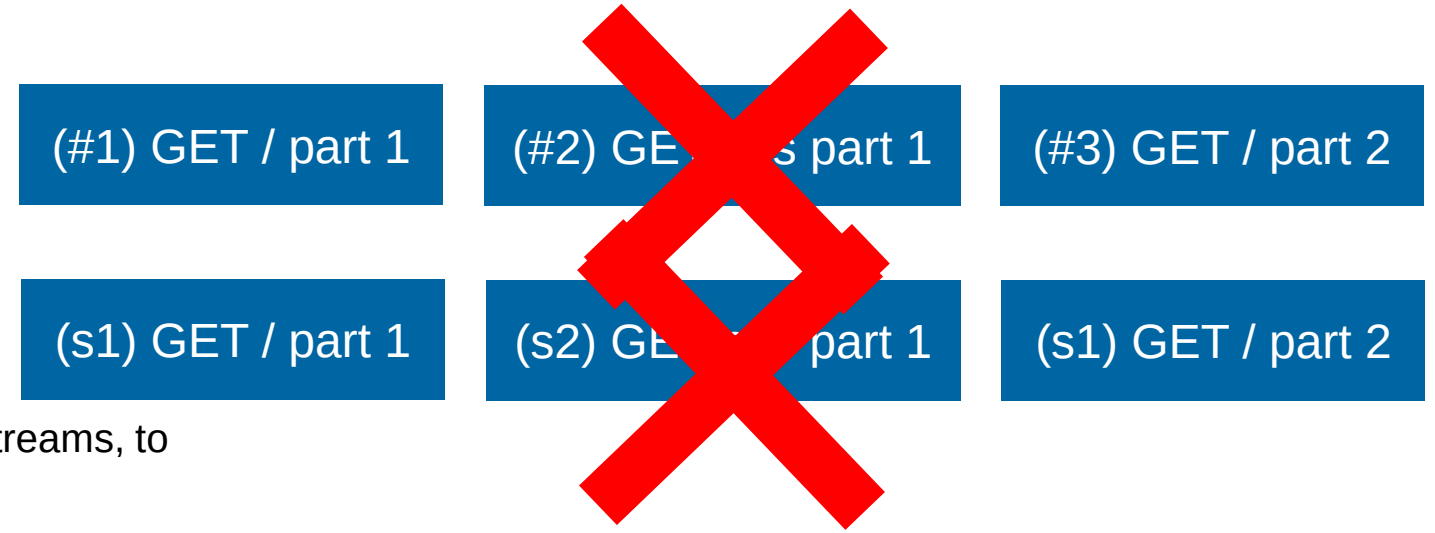
```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+
|1|1|T T|X X X X|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Version (32)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|DCIL(4)|SCIL(4)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Destination Connection ID (0/32..144)          ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Source Connection ID (0/32..144)              ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```



QUIC

- **Head-of-line blocking**
  - HTTP/1 → HTTP/2
  - One packet loss, TCP needs to be ordered, everything stops
  - QUIC can multiplex requests: one stream does not affect others
- **QUIC 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,
    - Two additional unidirectional QUIC streams, to sync mapping
  - Many TCP optimizations



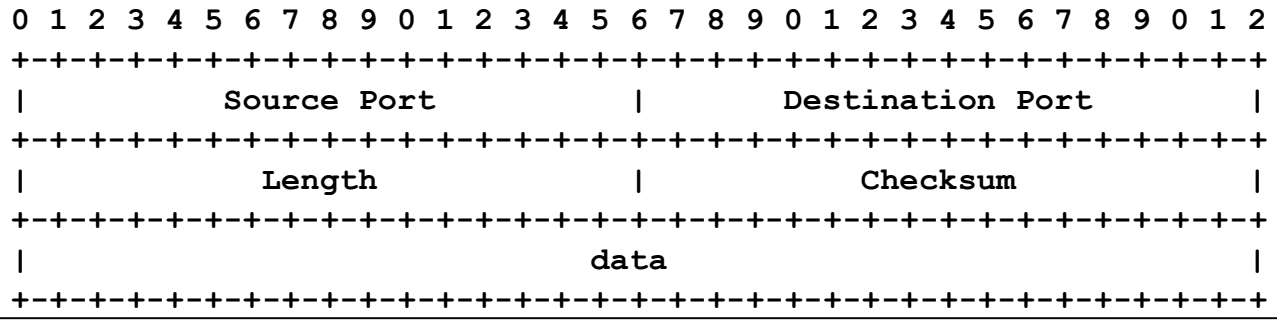


URL: b.socrative.com/student/

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
- **Simple Client/Server (wireshark)**
  - nc –l –u 4000 / nc –u localhost 4000



- **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

Comparison – Transport Layer

TCP *

- Transport layer
- Connection oriented
- Reliable transfer
- Streams
- Guaranteed order
- Widely used – HTTPS
- 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
- No flow, congestion
- Lightweight
- Header size is 8 bytes
- Error checking, no recovery

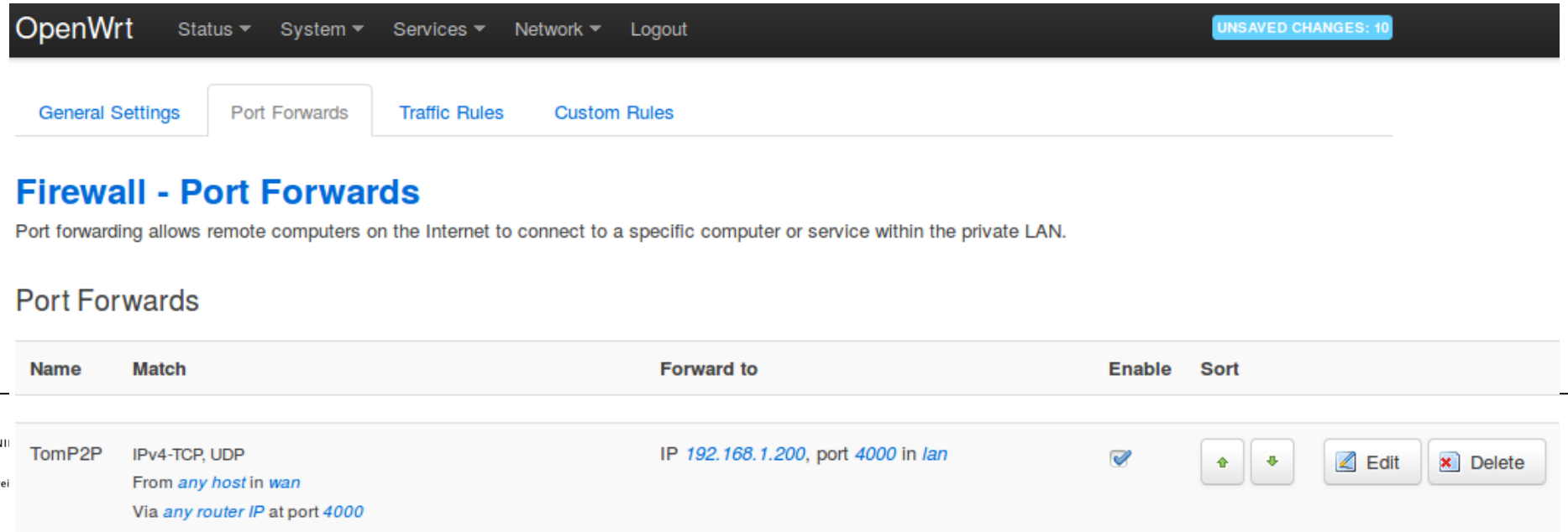
SCTP *

- 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

- **NAT**
  - Network Address Translation – breaks end-to-end
  - “If nothing else, [NAT] can serve to provide temporarily relief while other, more complex and far-reaching solutions are worked out”
    (RFC 1631 - The IP Network Address Translator (NAT))
- **How to deal with NAT: “Easy solution”:**
  - Manual port forwarding: e.g., setup on your router



- **Easy solution: UPNP / NAT-PMP**
  - Both configure port forwarding, but UPNP is more: discover devices - uses broadcasting to find router (Simple Service Discovery Protocol)
  - UPNP: configure devices - uses HTTP and XML to configure port forwarding (Internet Gateway Device Protocol)
  - NAT-PMP: protocol made for configuring port-forwarding, but no discover (how to find router?)

Router / NAT Configuration

UPnP, IGD

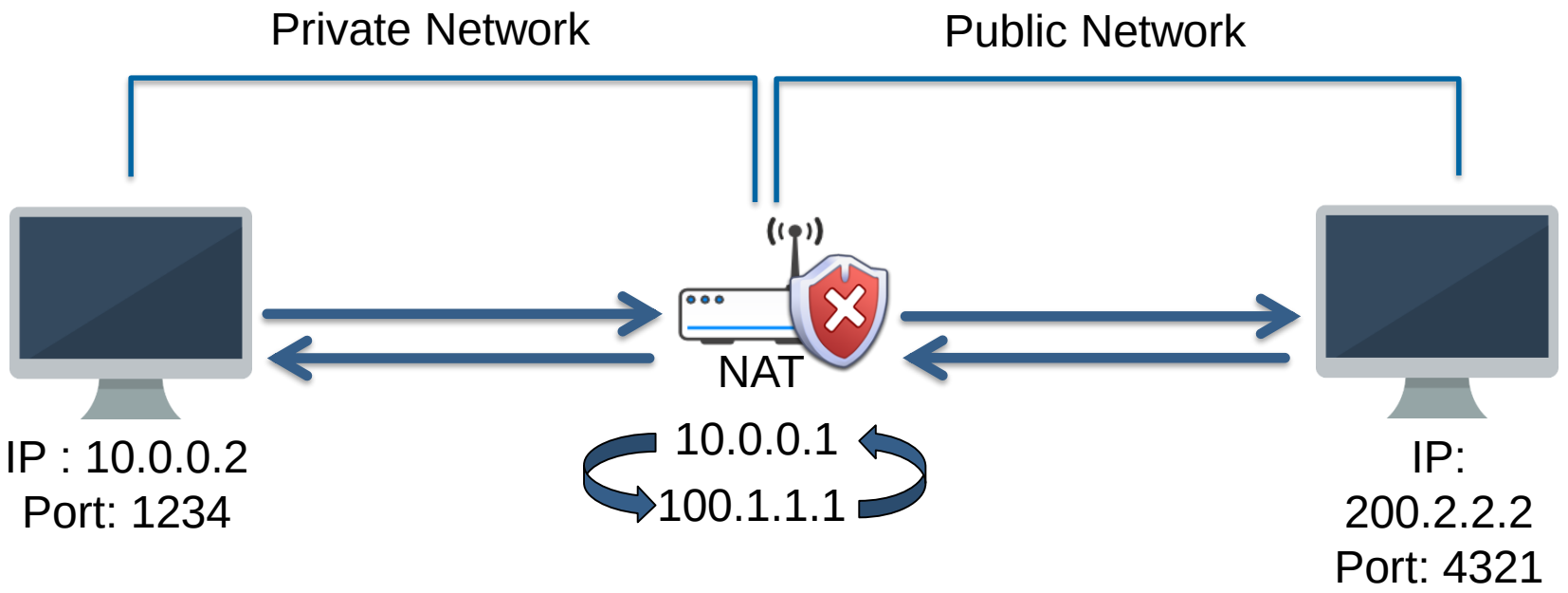
- Networking protocols for data sharing, communications, and entertainment
- Internet Gateway Device (IGD) Standardized Device Control Protocol: UDP mDNS broadcast to 239.255.255.250:1900
- Returns information about devices, e.g. p43
  - friendlyName, manufacturer, modelDescription, presentationURL, localAddress, modelNumber, modelName
  - Responses sent over UDP, with HTTPU
- HTTP connection to router with SOAP / XML message
  - No authentication, no HTTPS
  - If not configured properly, access from Internet

NATPMP

- Introduced in 2005 by Apple as an alternative to IGD – RFC 6886
- UDP, port 5351, simple binary protocol
  - Simplicity!
- No authentication
- How to find router? (Java (tomp2p)vs. C#)

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Vers = 0      | OP = x        | Reserved                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Internal Port                 | Suggested External Port       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Requested Port Mapping Lifetime in Seconds                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 Opcodes supported:
1 - Map UDP
2 - Map TCP
```

- **Difficult solution: hole punching**
  - rendezvous / relay peer which does “hole punching”, in worst case relay traffic.
- **NAT: translation table for private / public network**



NAT Table Entry: (10.0.0.2:1234, 200.2.2.2:4321; 200.2.2.2:4321, 100.1.1.1:1234)

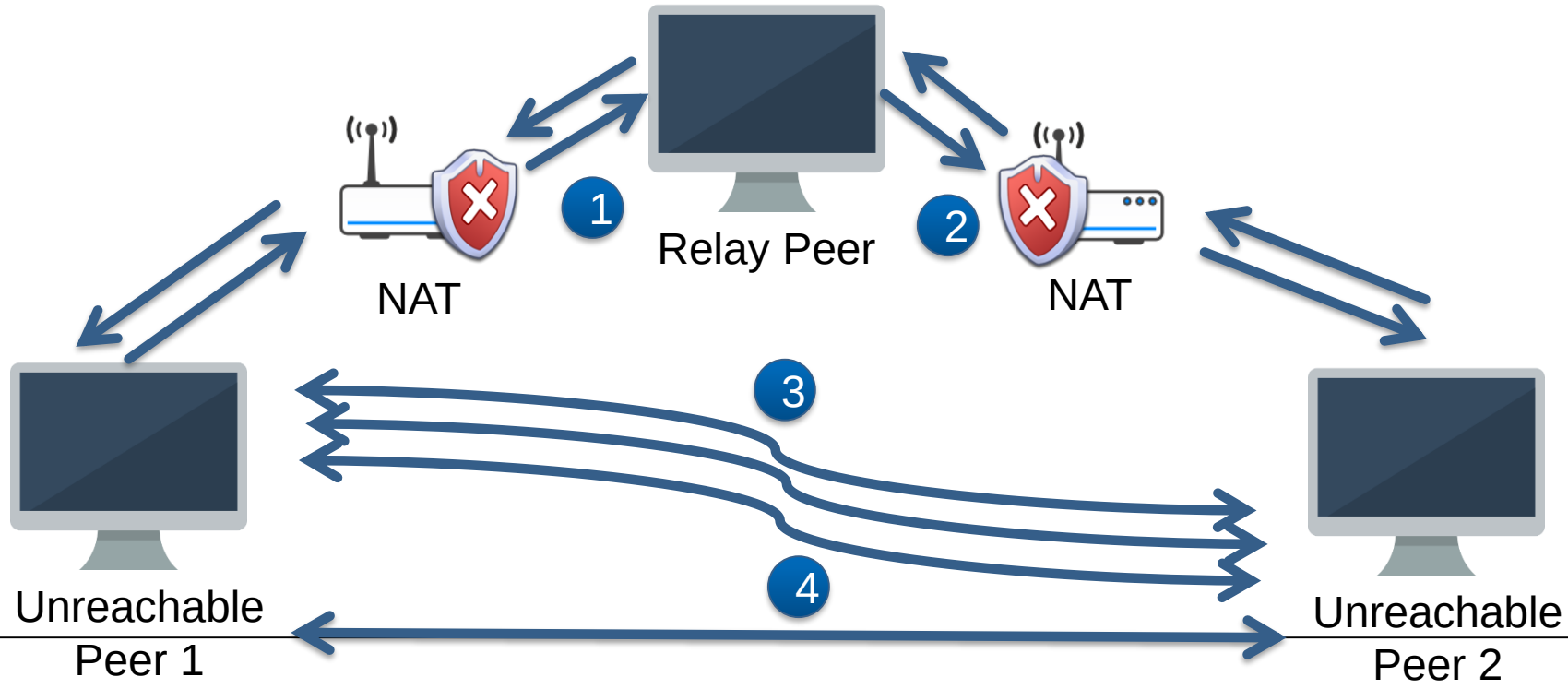
Connectivity, Security, and Robustness: Hole punching in P2P Networks

1) Peer 1 initiates a new connection trial to peer 2 via relay and signals its source ports and IP (relay/rendez-vous peer has connection to Unreachable Peer 2)

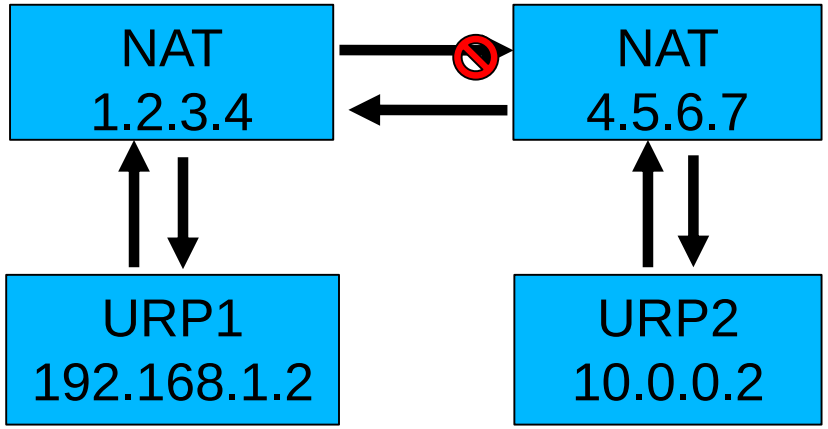
2) Peer 2 answers back with its source ports and IP

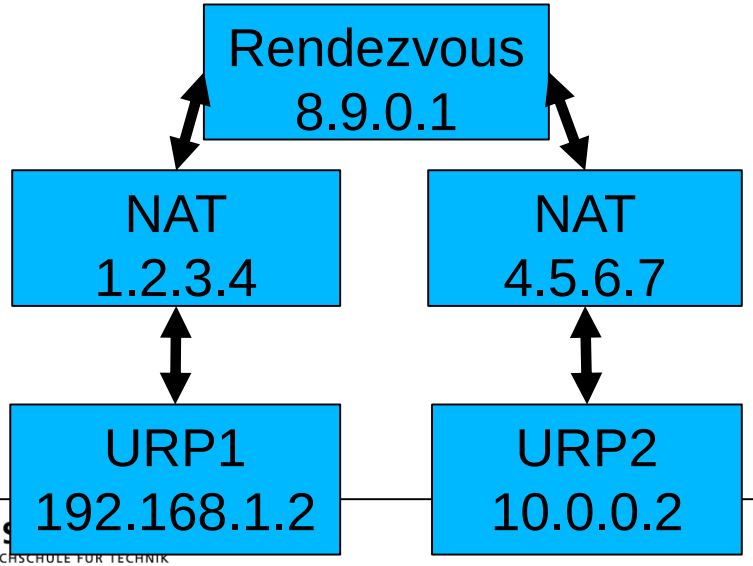
3) Both of the peers punch holes into their firewall/NAT

4) Established a connection



- **Hole punching**
  - URP1 got 4.5.6.7:5000, URP2 got 1.2.3.4:4000
  - Unreachable peer 1 request to NAT 4.5.6.7, will fail – no mapping, however, unreachable peer 1 creates mapping with that request
  - Unreachable peer 2 sends request to unreachable peer 1 (1.2.3.4:4000) success!



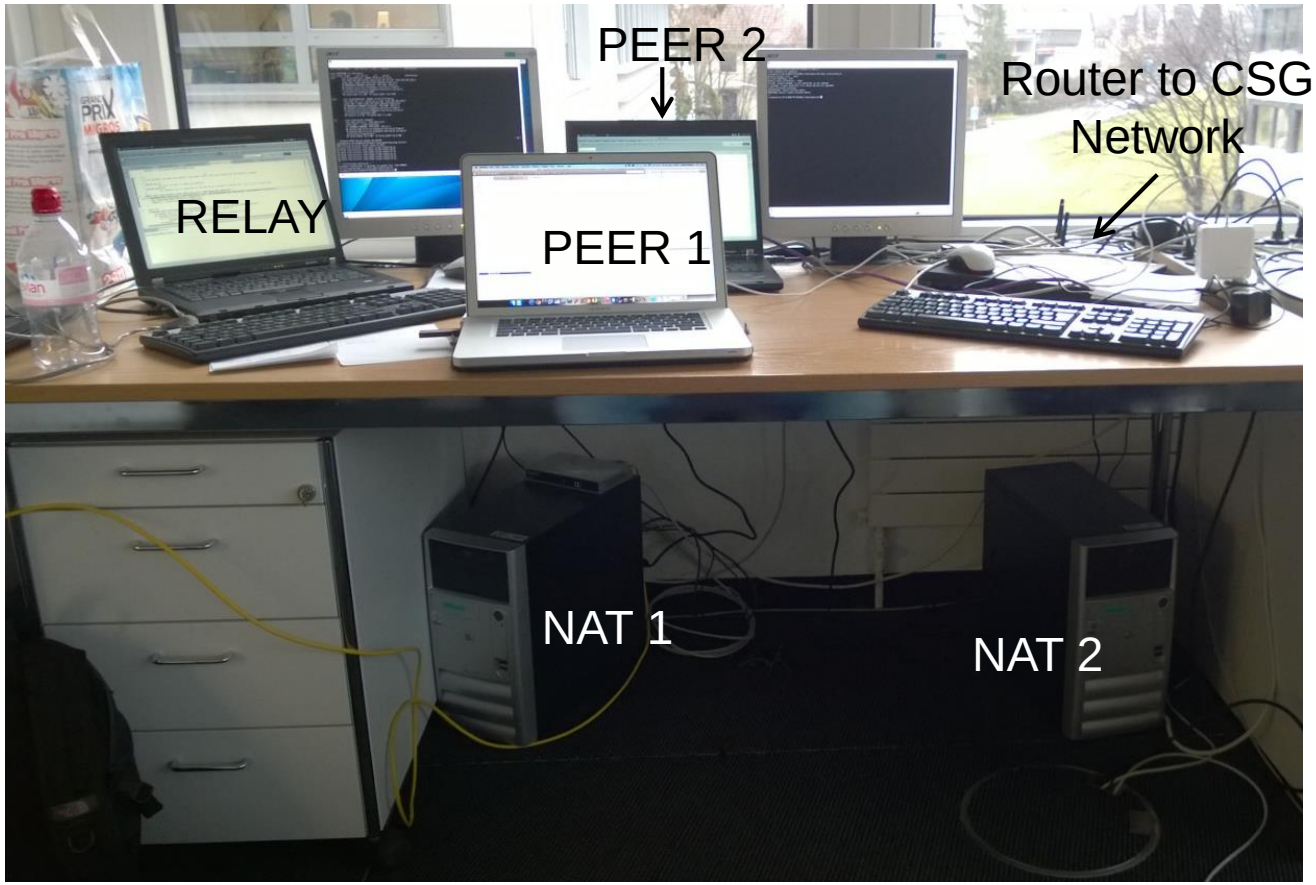


Mapping for NAT 1.2.3.4 (Unreachable peer 1)			
192.168.1.2:4000	4.5.6.7:5000	4.5.6.7:5000	1.2.3.4:4000

Mapping for NAT 4.5.6.7 (Unreachable peer 2)			
10.0.0.2:5000	1.2.3.4:4000	1.2.3.4:4000	4.5.6.7:5000

Connectivity, Security, and Robustness

- **Hole Punching Development (in the old days)**
- **Currently: network namespaces (since Linux 2.6.24)**

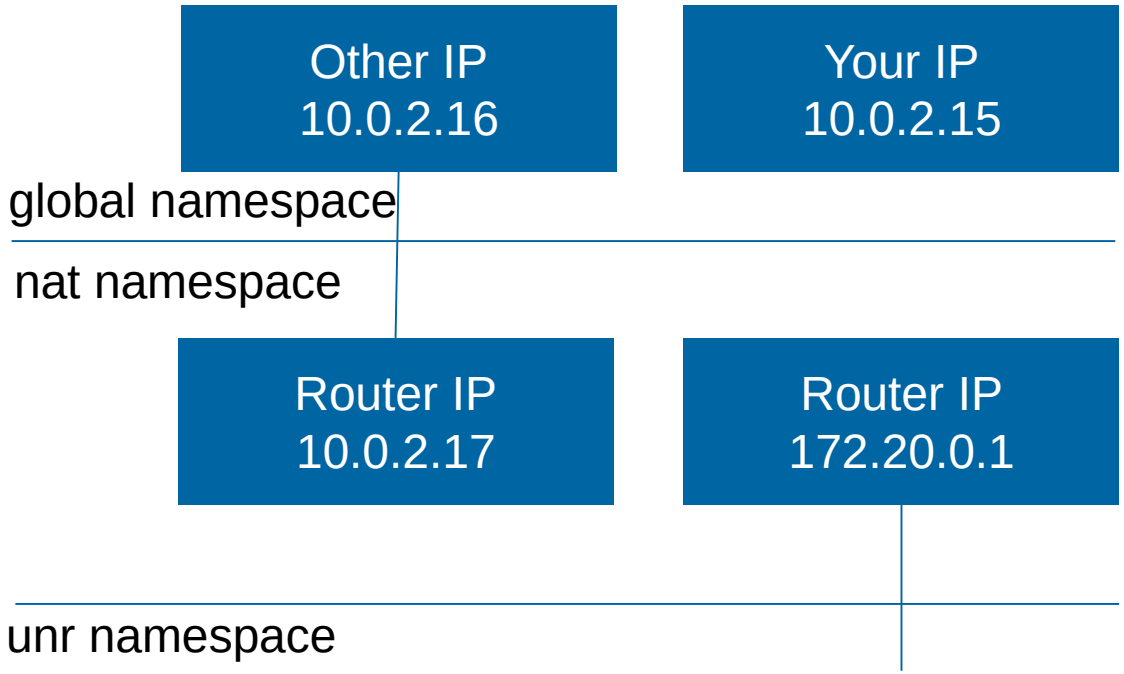


URL: b.socrative.com/student/

Room: HSR

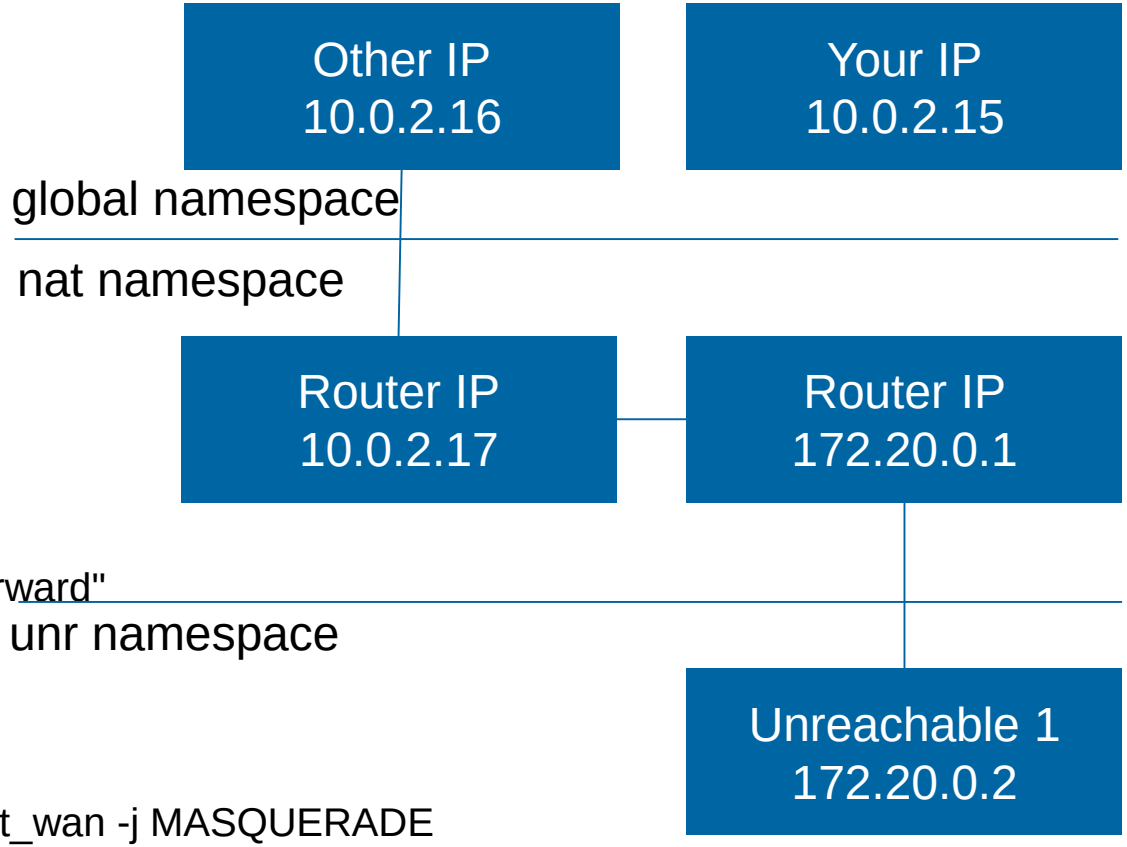
Make your own Testbed for P2P System

- **Network namespaces – also used in docker**
  - We do it manually
- **veth - Virtual Ethernet Device**
  - Tunnels between network namespaces
    - ip netns add unr (nat) / ip netns list
    - ip link add nat_wan type veth peer name nat_real
    - ip link add nat_lan type veth peer name unr_lan
    - ip link set nat_wan netns nat
    - ip link set nat_lan netns nat
    - ip link set unr_lan netns unr
    - ip address add 10.0.2.16/24 dev nat_real
    - ip link set nat_real up
    - ip netns exec nat ip address add 10.0.2.17/24 dev nat_wan
    - ip netns exec nat ip link set nat_wan up
    - ip netns exec nat ip address add 172.20.0.1/24 dev nat_lan
    - ip netns exec nat ip link set nat_lan up



Make your own Testbed for P2P System

- **veth - Virtual Ethernet Device**
  - Tunnels between network namespaces
    - ip netns exec unr ip address add 172.20.0.2/24 dev unr_lan
    - ip netns exec unr ip link set unr_lan up
    - ip route add 10.0.2.17 dev nat_real
    - ip netns exec unr route add default gw 172.20.0.1
    - ip netns exec nat route add default gw 10.0.2.16
    - ip netns exec nat sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
    - echo 1 > /proc/sys/net/ipv4/ip_forward
    - echo 1 > /proc/sys/net/ipv4/conf/all/proxy_arp
    - ip netns exec nat iptables -t nat -A POSTROUTING -o nat_wan -j MASQUERADE
    - ip netns exec nat iptables -A FORWARD -i nat_wan -o nat_lan -m state --state REL
    - ATED,ESTABLISHED -j ACCEPT
    - ip netns exec nat iptables -A FORWARD -i nat_lan -o nat_wan -j ACCEPT



Exercises

- **Download VirtualBox**
- **Download alpine image (41MB)**
- **SSH forwarding enabled on port 2222**
- **Login: ssh root@localhost –p 2222**
- **Password: hsr12345**
- **./nw.sh is the networking setup – the two previous slides**
- **./nat.sh run shell in nat namespace**
- **./unr.sh run shell in unr namespace**
- **./upnp.sh run UPNP/NAT-PMP**
- **ssh-copy-id root@localhost -p 2222**

- **Exercise 1: Setup VM**
- **Exercise 2: In the unr namespace, ping an outside IP (8.8.8.8) – make sure it works**
  - Check the conntrack table in the nat namespace – what do you see?
- **Exercise 3: use netcat (nc) and create a UDP server in the global name space**
  - Use nc to connect to that server from the unr namespace
- **Exercise 4: use nc and create a UDP server in the unr namespace**
  - Use nc to connect to that server from the global namespace – why does it fail?
- **Exercise 5: Punch a whole into the connection tracking table in the nat namespace**

Exercises

- **Download GoLand**
- **Sources must go into *yourhome*/go/src/**
- **Compile with: CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build -a -tags netgo -ldflags '-w' -o vss *.go**
  - To run on the Alpine image, otherwise go build
- **scp –P 2222 vss root@localhost:/root**
- **Exercise 6: run ./upnp.sh and launch the UPNP/NAT-PMP daemon**
- **Exercise 7: write a go client to configure NAT-PMP and run it in the unr namespace**
  - Run a server in unr, run a client in global, connect via configured ports

```
fmt.Println("connecting...")
conn, err := net.DialTimeout("udp", "172.20.0.1:5351", 1*time.Second)
if err != nil {
          fmt.Println(err)
          os.Exit(1)
}
fmt.Println("connected.")
n, err := conn.Write([]byte{…}) //your protocol here
if err != nil {
          fmt.Println(err)
          os.Exit(1)
}
fmt.Println("wrote %d bytes", n)
tmp := make([]byte, 512)
n, err = conn.Read(tmp)
if err != nil {
          fmt.Println("read:", err)
          os.Exit(1)
}
str := hex.EncodeToString(tmp[:n])
fmt.Println(str)
fmt.Println("read %d bytes", n)
conn.Close()
```

Questions?

Prof. Dr. Thomas Bocek
thomas.bocek@hsr.ch