Distributed Systems (DSy)

Application Protocols

Thomas Bocek

15.04.2026

Learning Goals

- Lecture 9 (Application Protocols)
  - Overview over important protocols on layer 7
  - Custom protocols, ASN.1, RPC, HTTP, JSON, WebSockets, Server Sent Events
  - Bencoding, WebRTC, DNS, Let's Encrypt, Mail Protocols

OST

Protocols

- Protocols, lecture 8: layer 4
  - TCP, UDP, (QUIC/HTTP/3)
- Designing custom protocols (e.g. Kafka)
  - (-) More development and testing time
  - (+) Can be more efficient (space/performance)
- Protocol generators (binary): Thrift / Avro / Protocol Buffers / (ASN1)
  - (+) IDL (Interface Description Language) generates code
  - (+) Standardized format
  - (−) Additional overhead

Example: Avro IDL – higher-level language for authoring Avro schemata → generates Avro schema

```
//Avro IDL
@namespace("ch.ost.i.dsl")

protocol MyProtocol{
  record AMessage {
     string request;
     int code;
  }
  record BMessage {
     string reply;
  }

  BMessage GetMessage(AMessage msg);
}
```

```
{"namespace": "ch.ost.i.dsl",
 "type": "record",  "name":
"AMessage",
 "fields": [
  {"name": "request", "type":
"string"},
  {"name": "code", "type": "int"}
 ]
}
```

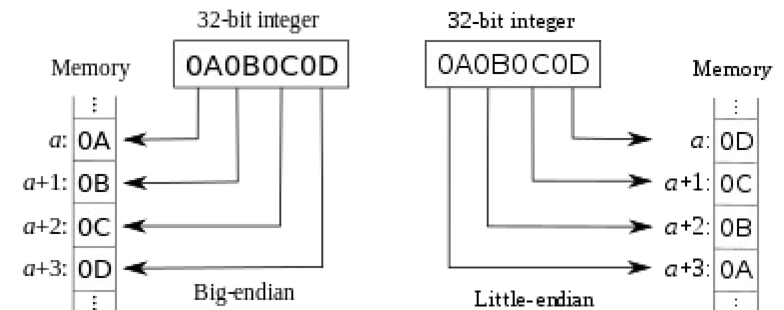

OST

Protocols

- Custom encoding/decoding
  - You control every aspect
  - You send more time on it

```
115     public static boolean decodeHeader(final ByteBuf buffer, final InetSocketAddress recipientSocket,
116             final InetSocketAddress senderSocket, final Message message) {
117         LOG.debug("Decode message. Recipient: {}, Sender:{}.", recipientSocket, senderSocket);
118         final int versionAndType = buffer.readInt();
119         message.version(versionAndType >>> 4);
120         message.type(Type.values()[(versionAndType & Utils.MASK_0F)]);
121         message.protocolType(ProtocolType.values()[versionAndType >>> 30]);
122         message.messageId(buffer.readInt());
123         final int command = buffer.readUnsignedByte();
124         message.command((byte) command);
125         final Number160 recipientID = Number160.decode(buffer);
126
127         //we only get the id for the recipient, the rest we already know
128         final PeerAddress recipient = PeerAddress.builder().peerId(recipientID).build();
129         message.recipient(recipient);
130
131
132         final int contentTypes = buffer.readInt();
133         message.hasContent(contentTypes != 0);
134         message.contentTypes(decodeContentTypes(contentTypes, message));
```

- Little-endian / Big-endian
  - sequential order where bytes are converted into numbers
- Networking, e.g. TCP headers: Big-endian
- Most CPUs e.g., x86: Little-endian, RISC-V: Bi-endianness



[source]

OST

Protocols Examples with Golang

- Example repo
  - TCP in C, golang, custom serialization
    → Anybody there? Example: 15 bytes

```
func main() {
   fmt.Println("connecting...")
   conn, _ := net.Dial("tcp", "127.0.0.1:7000")
   defer conn.Close()
   buf := make([]byte, 15)
   buf[0]=5
   copy(buf[1:],[]byte("5Anybody there?"))
   _, _ = conn.Write(buf)
}
```

```
func main() {
   fmt.Println("listening...")
   tcpConn, _ := net.Listen("tcp", ":7000")
   conn, _ := tcpConn.Accept() //do this in a go routine
   b := make([]byte, 15)
   n, _ := conn.Read(b)
   fmt.Printf("connecting... read: %d, addr: %v, data:
[% x], decoded: %v\n",
      n, conn.RemoteAddr(), b[:n], string(b[1:]))}
```

- ASN1, defined in 1984. Standard interface description language (IDL), used for serializ / deserializ – used e.g., certs
- Avro: data serialization system, remote procedure call and data serialization framework - Hadoop (Big-data framework)
- Protobuf: data serialization from Google, IDL, goals: smaller and faster than XML
- Thrift - RPC Framework from Facebook, IDL and binary protocol
- CBOR / MessagePack
  - And even more: Benconding, Cap'n Proto , FlatBuffers, ...

OST

RPC Examples

- RPC - (Remote Procedure Call) lets programs execute code on remote machines.
  - E.g., gRPC, uses HTTP/2 for transport, uses Protocol Buffers
  - Features: authentication, bidirectional streaming and flow control, blocking or nonblocking bindings, and cancellation and timeouts, many languages
  - Example: 171 / 124 (wireshark) – request/reply

```
syntax = "proto3";

service MessageService {
  rpc SendMessage (AMessage)
returns(Empty);
}

message AMessage {
  int32 code = 1;
  string message = 2;
}

message Empty {}
```

- JSON/REST/HTTP
  - Human-readable text to transmit data
  - Is it RPC? Yes/No
    - REST ideally stateless, RPC can be stateful
    - REST responses contain all metadata, RPC often needs interface definitions
  - Parsing overhead, JSON slower than binary protocol - benchmarks
  - Often used for web apps, example: 39 bytes


```
[
    {
        "id": "bitcoin",
        "name": "Bitcoin",
        "symbol": "BTC",
        "rank": "1",
        "price_usd": "9324.08",
        "price_btc": "1.0",
        "24h_volume_usd": "9039300000.0",
        "market_cap_usd": "158560288125",
        "available_supply": "17005462.0",
```

Application Protocol: HTTP

- HTTP (HyperText Transfer Protocol): foundation of web data communication
- History: 1989 (Tim Berners-Lee)
  - HTTP/1.1 (1997) → HTTP/2 (2015, header compression, multiplexing) → HTTP/3 (2022, QUIC-based)
- Request/response model, resources identified by URL
- URL structure: scheme, user info, host, port, path, query, fragment
- HTTP1: text-based protocol

```
openssl s_client -connect dsl.i.ost.ch:443 -showcerts
… TLS handshake …
GET / HTTP/1.1
```

- Stateless (server keeps no state)
- Browser sends additional headers (User-Agent, Accept, Accept-Encoding, etc.)

Request Headers (359 B)

```
Host: dsl.hsr.ch
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:73.0) Gecko/20100101 Firefox/73.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
TE: Trailers
```

Scheme User info Host Port Path Query Fragment

```
http://tbocek:password@dsl.i.ost.ch:443/lect/fs21?id=1234&lang=de#topj
```


OST

Application Protocol: HTTP

- Response: header + status code + content body
  - Header

```
Response Headers (227 B)
HTTP/2 200 OK
server: cloudflare-nginx
content-type: text/html; charset=UTF-8
date: Mon, 02 Mar 2020 14:29:39 GMT
x-page-speed: 1.13.35.2-0
cache-control: max-age=0, no-cache
content-encoding: gzip
X-Firefox-Spdy: h2
```

  - Status codes: 1xx (informational), 2xx (success – 200 OK), 3xx (redirection), 4xx (client error – 404 Not Found, 403 Forbidden), 5xx (server error), [list]
  - Content

```
<!DOCTYPE html>
<html>
<head>
    <title>Distributed Systems and Ledgers Lab</title>
    <link rel="stylesheet" type="text/css"
href="design/layout.css"/>
…
```

- Request methods: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH
- Web server one-liner - with netcat:
  - while true; do echo -e "HTTP/1.1 200 OK\r\n\r\n<h1>Hallo</h1>" | nc -vlN -p 8080; done
- Every major browser has dev tools to show request / responses
  - Firefox, Chrome: ctrl+shift+i / F12
  - Used regularly

OST

WebSockets

- Full-duplex communication over TCP [overview]
  - REST / JSON is in one direction
- How can the server notify the browser (client?)
  - Polling
    - Short: request e.g. every 0.5s
    - Long: request until timeout or reply
  - Server Sent Events (alternative) SSE
    - One way communication from server to browser (client)
    - Server receives a regular HTTP request, keeps connection open (`Accept or Content-Type: text/event-stream`), server can now push data to the client
  - WebSockets
    - Two way communication, RFC 6455

- HTTP handshake, then upgrade to communication channel

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Server response:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

  - Data can be text or binary
- With SSL/TLS → wss://
  - Some configuration required on LBs / RRs

OST

WebSockets / SSE

- GoLang / HTML/JS Example
  - Many languages supports WebSockets

```
<script type="text/javascript">
    const connection = new WebSocket('ws://localhost:8080/ws');
    connection.onopen = function () {
        connection.send('start');
    };
    let counter = 0;
    connection.onmessage = function (e) {
        console.log('update websocket: ' + counter++);
        const date = JSON.parse(e.data);
        const element = document.getElementById('text');
        element.innerHTML = 'Time: ' + date.now;
    };
</script>
```

- Server Sent Events Example
  - Uses standard HTTP connections
  - Native browser support via EventSource API
  - Automatic reconnection if connection is lost
  - Max number of concurrent connections per domain

```
<script type="text/javascript">
    const evtSource = new EventSource("/sse");
    let counter = 0;

    evtSource.onmessage = function(event) {
        console.log('update SSE: ' + counter++);
        const date = JSON.parse(event.data);
        const element = document.getElementById("text");
        element.innerHTML = 'Time: ' + date.now;
    };

    evtSource.onerror = function() {
        console.log('SSE connection error');
        evtSource.close();
    };
</script>
```

WebRTC – Introduction

- WebRTC for browser to browser communication
  - Browser-to-browser, P2P (mostly no server)
  - Real time communication (RTC) via API
  - Goal: eliminate plugins or native apps
  - Supported by Google, Microsoft, Mozilla, Opera, Apple
- Google bought in 2010 Global IP Solutions (GIPS) and open sourced WebRTC in 2011
  - Protocol standardized by IETF (codec requirements, media protocol),
    JavaScript API by W3C
- First cross-browser call (Chrome ↔ Firefox) in February 2013

WebRTC

- Compatibility, 97%
- W3C Recommendation finalized: 26.01.2021, updated 25.03.2025
- Replaced need for Flash, Java plugins, proprietary conferencing apps
- Used in WhatsApp, Facebook Messenger, MS Teams, and others

OST

WebRTC – Concerns

- HTML browsers get bloated
  - Several GB RAM to open couple of tabs?
  - Hint: adblocker
- WebRTC API could be simplified
- Security Concerns
  - Private IP / IP behind VPN, Tor? (https://dsl.i.ost.ch/lect/webrtc/)
- But: WebRTC forbids unencrypted communication
  - DTLS (data), SRTP (media)
  - Complexity - SCTP over DTLS over UDP

Demo for: https://github.com/diafygi/webrtc-ips

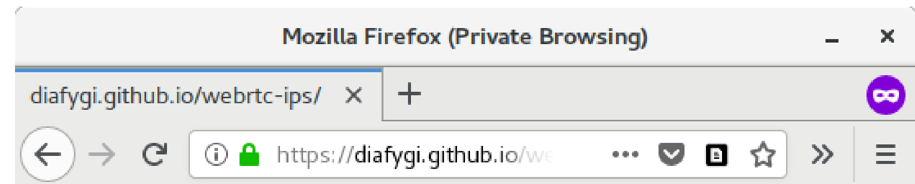
This demo secretly makes requests to STUN servers that can log your request. These requests
by browser plugins (AdBlock, Ghostery, etc.).

Your local IP addresses:

- 10.8.0.18

Your public IP addresses:

Your IPv6 addresses:



Demo for: https://github.com/diafygi/webrtc-ips

This demo secretly makes requests to STUN servers that can log your request. These requests do not show up in developer consoles and cannot be blocked by browser plugins (AdBlock, Ghostery, etc.).

Your local IP addresses:

- 192.168.1.139

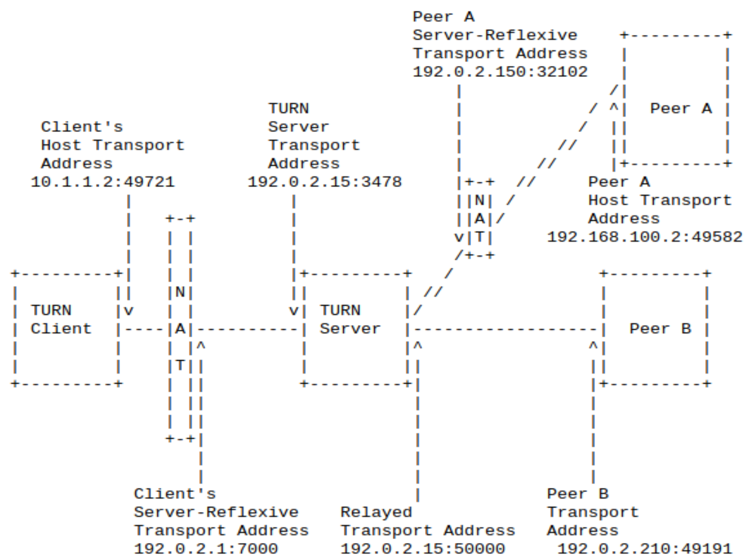
Your public IP addresses:

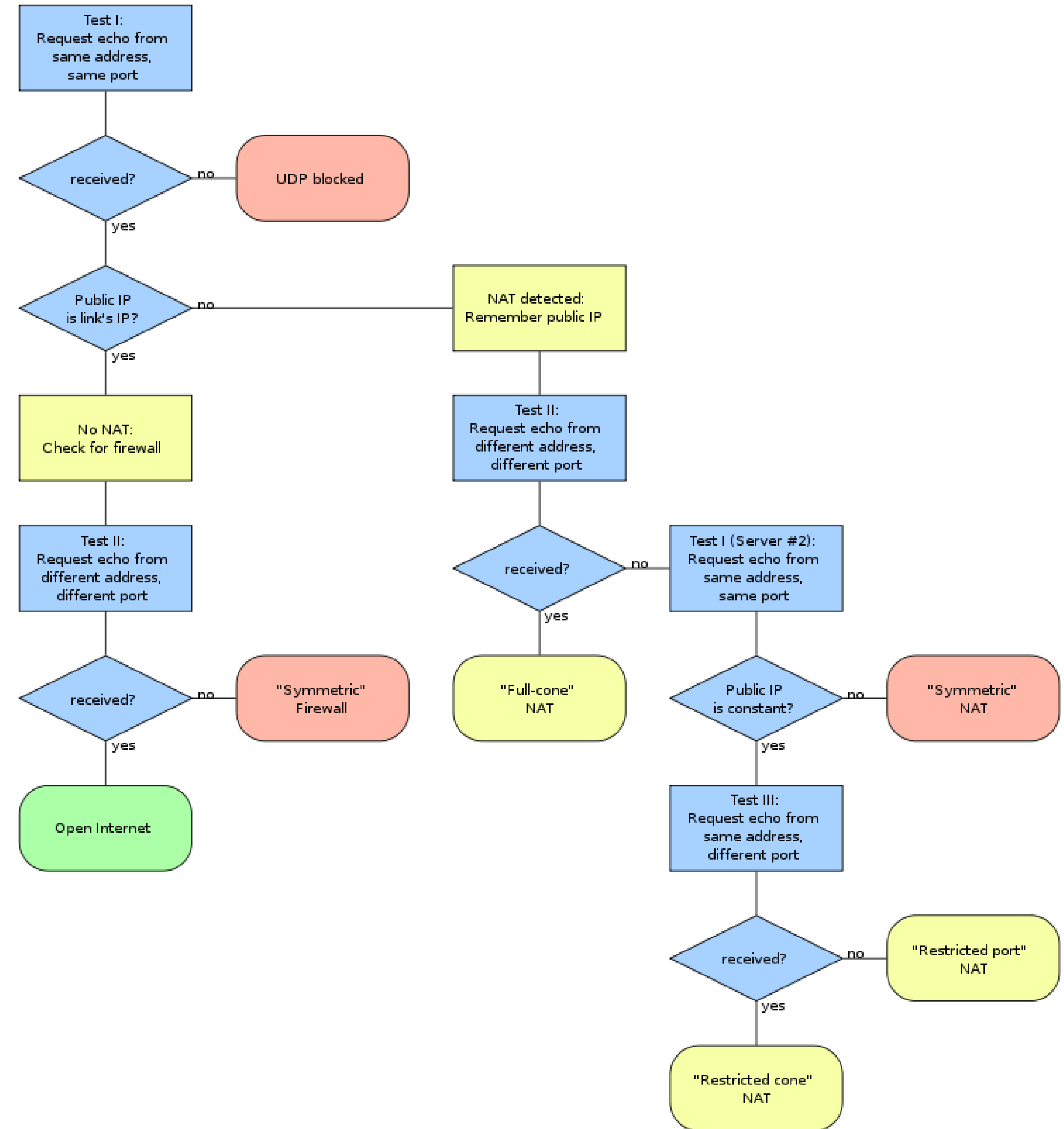
Your IPv6 addresses:

OST

WebRTC – Introduction

- Developer does not need to handle NAT directly NAT directly
  - Abstraction using STUN, ICE, TURN
  - STUN: session traversal utilities for NAT (detect which kind of NAT, rfc5389)
    - “STUN is not a NAT traversal solution by itself”
  - TURN: traversal using relays around NAT

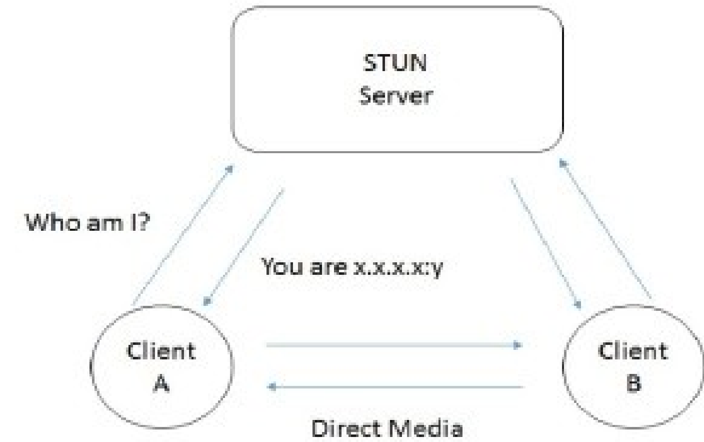


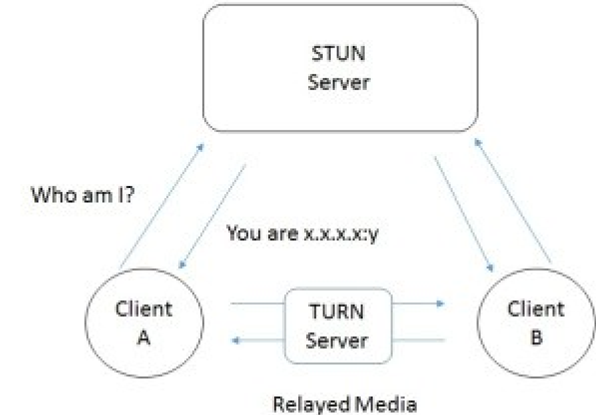


OST

WebRTC – Introduction

- TURN
  - TURN client/server uses UDP, TCP/TLS
    - Some firewalls block UDP entirely
  - UPnP / NAT-PMP setup by browser optional
    - Bugzilla@Mozilla – Bug 860045
- ICE - Interactive Connectivity Establishment
  - RFC 8445 a protocol for NAT traversal
  - “ICE works by exchanging a multiplicity of IP addresses and ports, which are then tested for connectivity by peer-to-peer connectivity checks.”

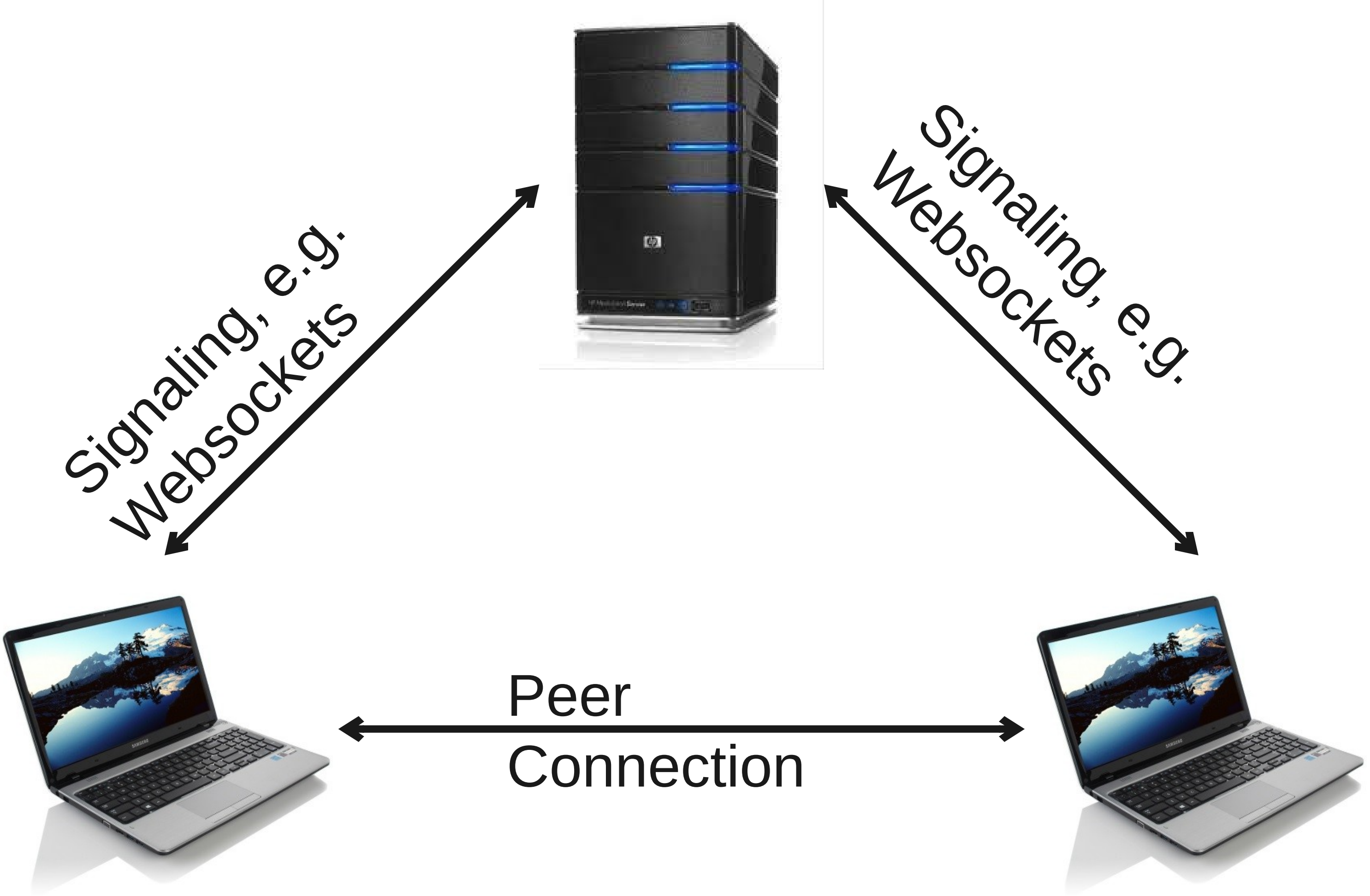




source

OST

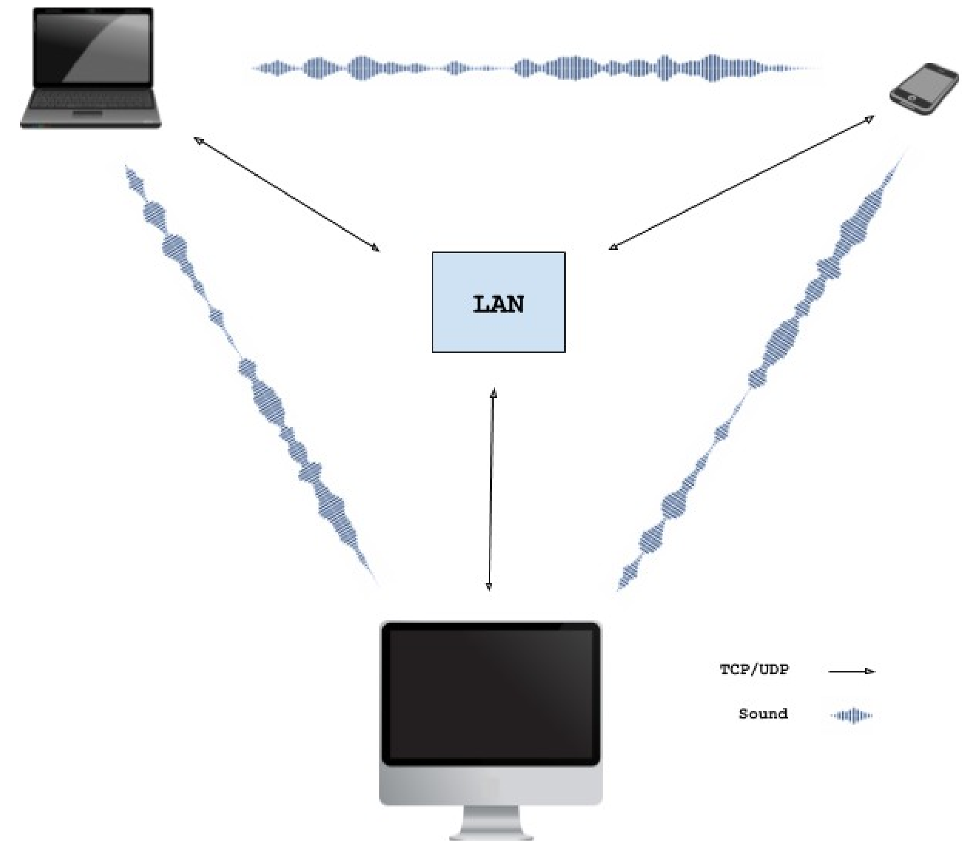
WebRTC Architecture - Triangle



OST

WebRTC Signaling

- A proof-of-concept for WebRTC signaling using sound. Works with all devices that have microphone + speakers. Runs in the browser:
  - https://github.com/ggerganov/wave-share



OST

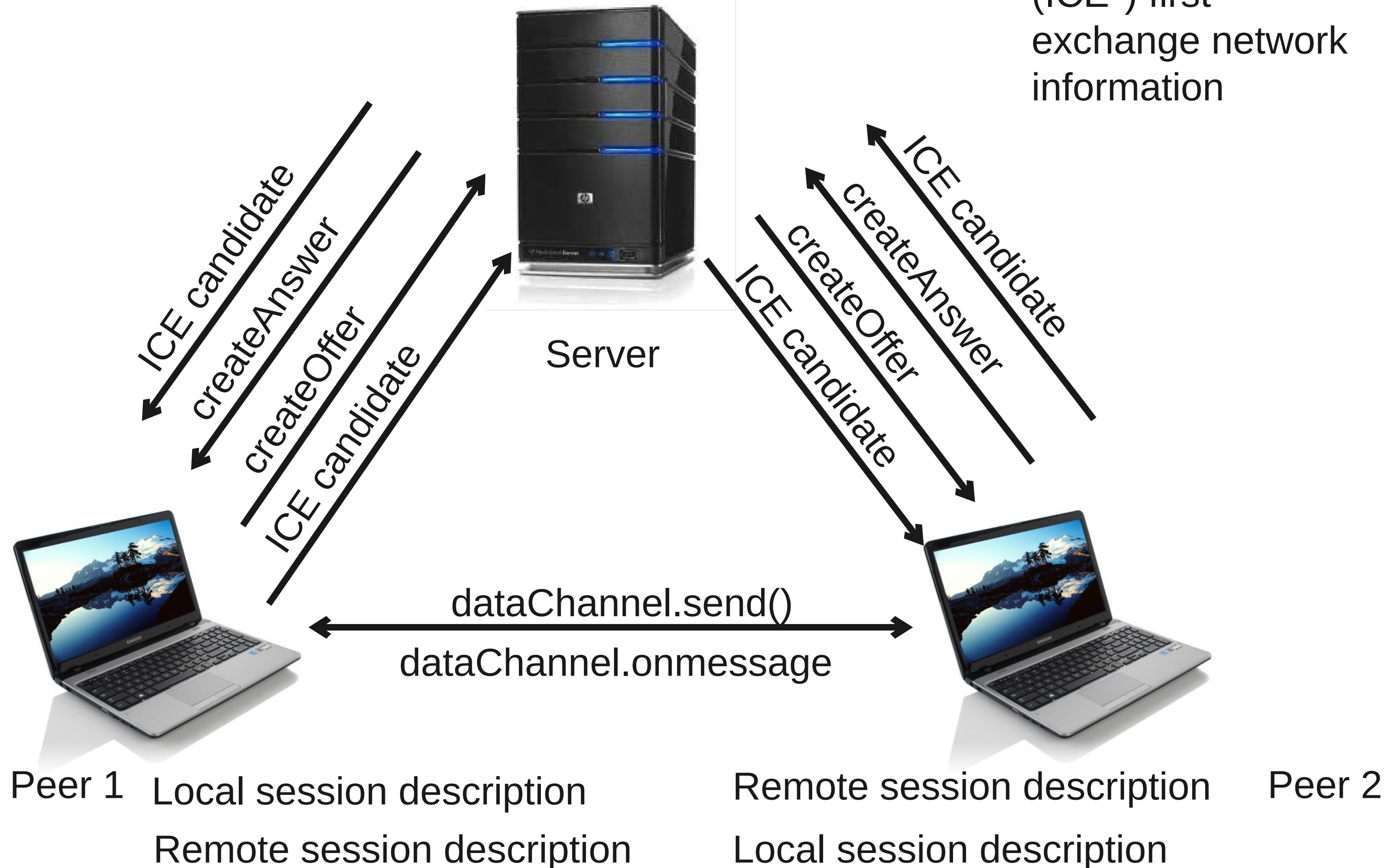
WebRTC - Demo

- Server - server.js
  - Node.js server
  - Serves files (index.html / express)
  - Listens to incoming WS messages and broadcast messages to all connected clients
- npm i / node server.js
- Minimal example

```
const express = require('express');
const WebSocket = require('ws');

let app = express();
// setup static files
app.use(express.static('.'));
// setup listening
let server = app.listen(4000, function () {
    console.log("listening on " +
server.address().address + ":" + server.address().port);
});

//WebSocket broadcast setup
const wss = new WebSocket.Server({ server });
wss.on('connection', function(ws) {
    ws.on('message', function(message) {
        // Broadcast any received message to all clients
        console.log('received: %s', message);
        wss.clients.forEach(function(client) {
            if(client.readyState === WebSocket.OPEN) {
                client.send(message.toString());
            }
        });
    });
});

console.log('Server running.');
```


OST

WebRTC Architecture - Demo

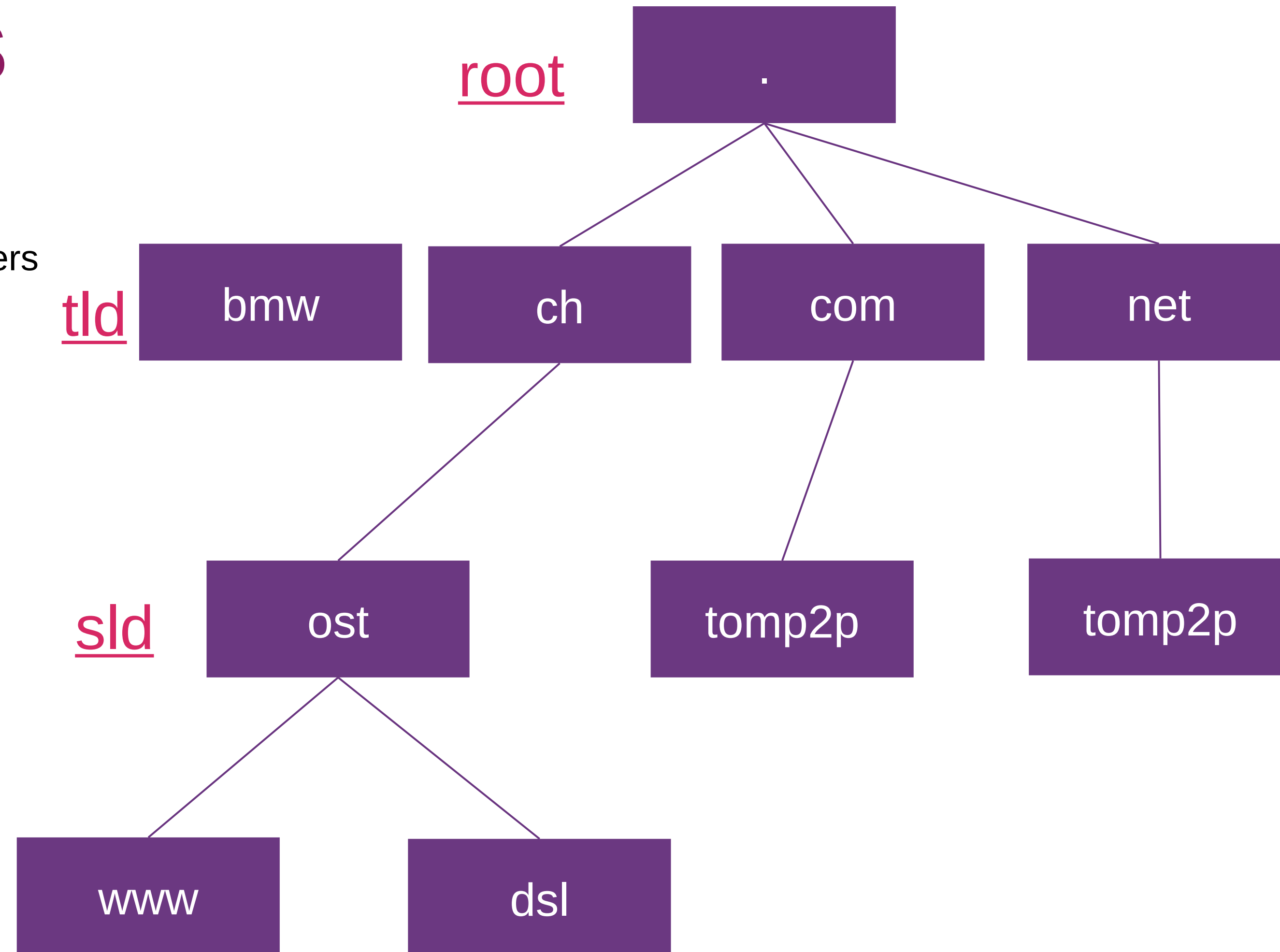


(ICE*) first exchange network information

OST

Application Protocol: DNS

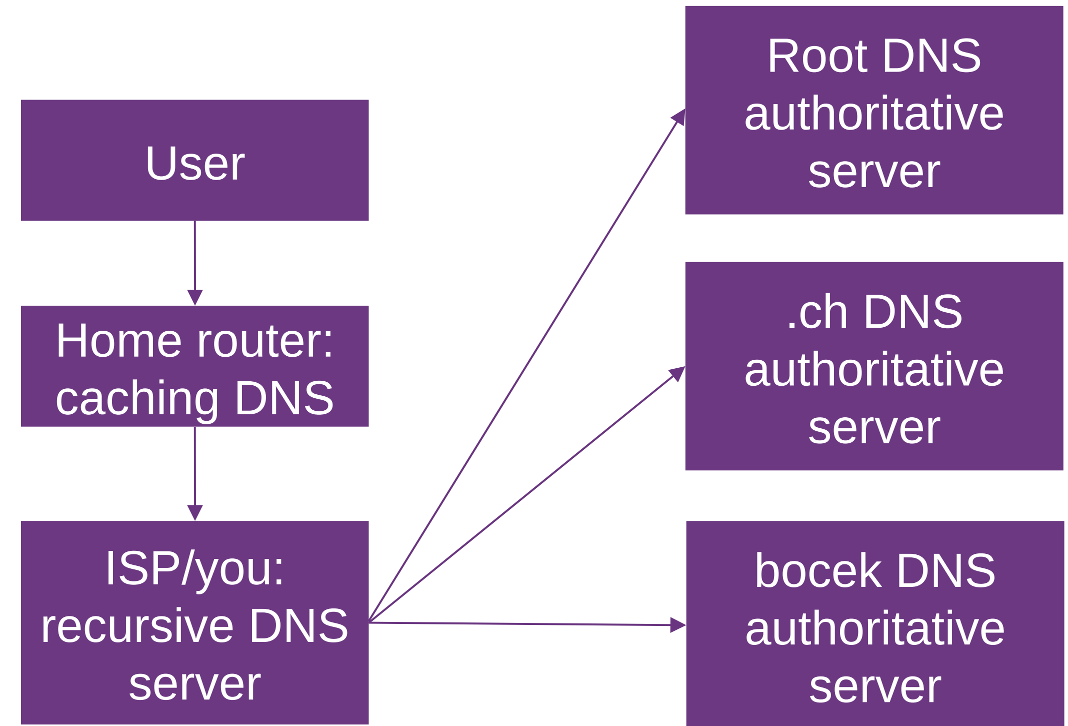
- Translates human readable domain names to IP addresses "phonebook of the Internet"
  - Delegate authority over sub-domains to other name servers
- Lots of new TLD: .zuerich, .bmw, .americanexpress, .youtube, .გე (application fee 185k USD) - not as widely used
  - No special characters: ASCII (no UTF)
  - Punycode: bücher.tld → xn--bcher-kva.tld
- Hierarchical and decentralized naming system for computers
  - E.g., dsl.i.ost.ch
  - Uses UDP, port 53
  - Designed in 1983: unencrypted, unsigned
- Before DNS: manual exchange of hosts.txt – did not scale



OST

Application Protocol: DNS

- Primary + secondary DNS for redundancy (zone transfer)
- Typical setup
  - User
  - Caching/forwarding DNS (e.g., dnsmasq)
  - Recursive servers: DNS name resolution for applications (e.g, bind/unbound)
  - Authoritative servers: providing a definitive answer of e.g., bocek.ch (e.g., bind/nsd)
    - Authoritative DNS: allows others to find your domain; Recursive DNS: allows you to resolve other domains
- Restriction to 13 root servers due to 512 byte packet limit
  - With anycast: ~2000 root server instances worldwide

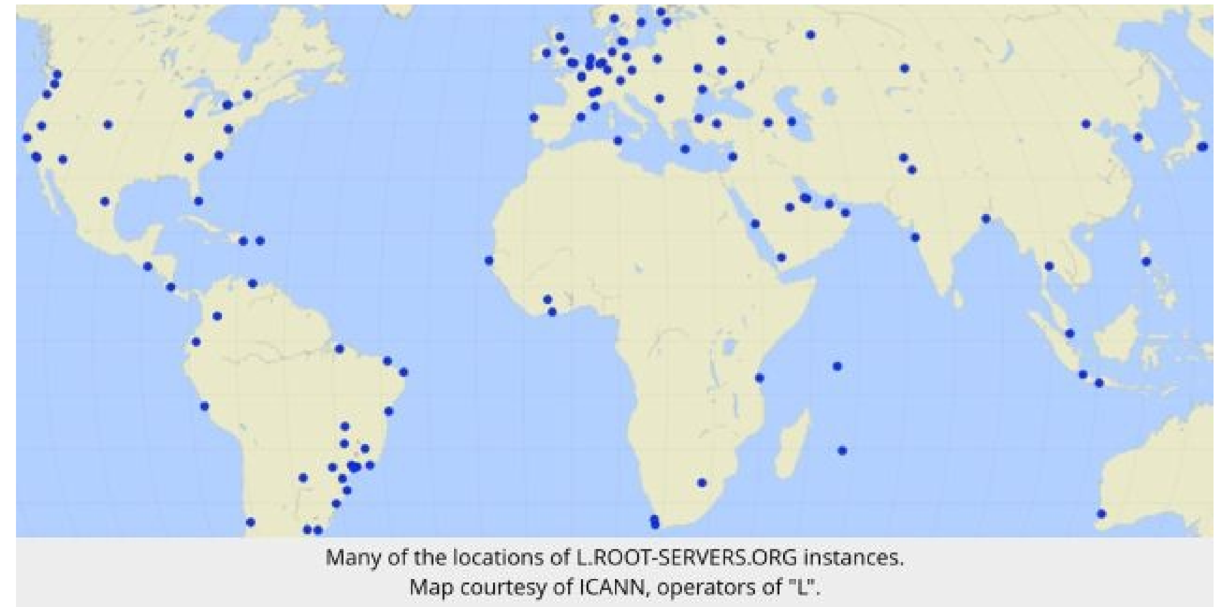


Application Protocol: DNS

- l.root-servers.net , 1 root IP with anycast mirrored in 90 locations
  - All root servers
- 2015: DNS root servers withstood major DDoS attack (~5M requests/s) [link]
  - Root servers are regularly attacked – system remains robust due to anycast redundancy and DNS caching
- Root zone managed by ICANN/PTI, Verisign as root zone maintainer – all US-based organizations under US jurisdiction
- Root zone file: download

https://www.iana.org/domains/root/servers

HOSTNAME	IP ADDRESSES	MANAGER
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	VeriSign, Inc.
b.root-servers.net	199.9.14.201, 2001:500:200::b	University of Southern California (ISI)
c.root-servers.net	192.33.4.12, 2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10, 2001:500:a8::e	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4, 2001:500:12::d0d	US Department of Defense (NIC)
h.root-servers.net	198.97.190.53, 2001:500:1::53	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	VeriSign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:9f::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project



Many of the locations of L.ROOT-SERVERS.ORG instances.
Map courtesy of ICANN, operators of "L".

OST

Application Protocol: DNS

- DNS structure
  - TTL defines the duration in seconds that the record may be cached by any resolver. “0” means no cache. Recommendation: > 1d
- Type of records
  - SOA - Start of Authority record: serial number and different caching times
  - NS - Name Server Record – sets the authoritative name server for this zone. 2 NS records – round robin! more sophisticated LB: split horizon
  - MX - name and relative preference of mail servers
  - A/AAAA - IPv4/IPv6 Address Record
  - TXT - arbitrary and unformatted text
  - PTR - opposite of A /AAAA

```
$TTL 3D
$ORIGIN bocek.ch.
@ SOA ns.nope.ch. root.nope.ch. (2025031401 8H 2H 4W
1D)
     NS   ns.nope.ch.
     NS   ns.jos.li.
     MX   mail.nope.ch.

     A    188.40.119.115
     TXT  "v=spf1 mx include:_spf.google.com -all"
www  A    188.40.119.115
cal  A    188.40.119.115
lb   A    188.40.119.115
lb   A    152.96.86.25

$INCLUDE "/etc/opendkim/keys/mail.txt"
$INCLUDE "/etc/bind/dmarc.txt"
```


OST

Application Protocol: DNS

- To run your own DynDNS service: TSIG
  - Enables DNS queries to authenticate updates to a DNS database
  - Uses shared secret and cryptographic hashing for authentication
- DNSSEC (security extension)
  - Authenticated and data integrity, not confidentiality
  - Can be used to bootstrap other security systems
    - Certificates, SSH fingerprints, IPSec pub keys
  - KSK: key signing keys to sign ZSK
  - ZSK: zone signing keys to sign records
    - Example: dig DNSKEY bocek.ch
- New record types: RRSIG, DNSKEY, DS, …
  - RRSIG, sign all resource sets
  - DS (delegation signer) record in the parent zone
    - dig DS tomp2p.net
  - ZSK to sign RRset
    - How to validate ZSK?
  - KSK to sign ZSK pub key
    - With 2 keys, its easier to change ZSK
- DoT and DoH
  - DNSSEC → signatures, DoT/DoH → encryption

DoH vs. DoT

DoH

- Provides confidentiality of lookups in transit
- Uses standard HTTP/2, on port (443)
  - Cannot distinguish between traffic/DNS
- Trivially deployed, DNS response are served like simple web pages
- Performance: TCP+TLS handshake → 2/3 RTT
  - But: Cloudflare is close to you
- Difficult upgrade path for clients: per-application installation
- Browsers can perform DNS queries using Javascript

DoT

- Provides confidentiality of lookups in transit
- DNS over TLS, separate port (853)
  - Can be blocked
- Widely supported by serving software (Bind, PowerDNS, Unbound) and public resolvers (Cloudflare, Quad9, Google)
- Performance: TCP+TLS handshake → 2/3 RTT
  - But: ISP is close to you
- Easy upgrade path for clients: clients can test if the configured resolver supports DoT on port 853, fall back to DoU53 otherwise)

Let’s encrypt

- Non-profit CA
  - Provides certificates for TLS
  - Golive in 2016 (started in 2012), now serving more than 700 million websites [link]
  - 6-day short-lived certificates available since Jan 2026
- Domain-validation certificates only – cannot compete with traditional CAs (identity checks)
- Automated renewal via ACME protocol – challenge response
  - HTTP-01: place file on web server
  - DNS-01: place TXT record (required for wildcard certs)

- Certbot – client for ACME
  - `certbot certonly --webroot -w /tmp -d boekch.ch --debug-challenges`
  - Copy the challenge where Let’s encrypt server can find it (in my case /var/www/html)
  - Old Nginx config (w/o ngx_http_acme_module)

```
server_name bocek.ch;
  ssl_certificate /etc/letsencrypt/live/bocek.ch/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/bocek.ch/privkey.pem;
```

  - This needs to be automated!

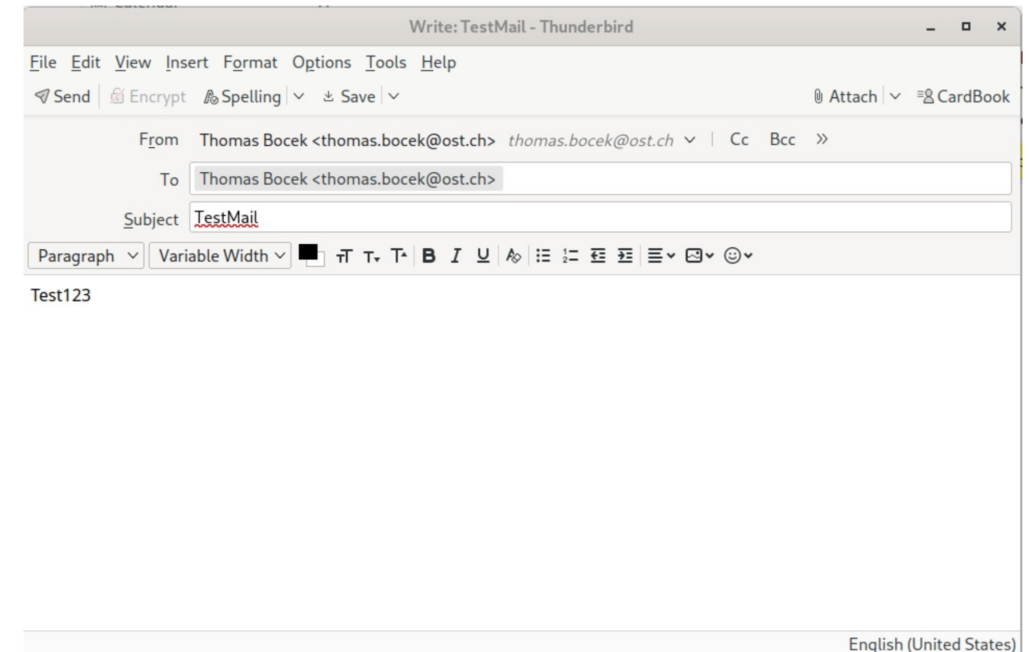
```
43 6 * * *   root   certbot renew --post-hook "systemctl
reload nginx"
```

  - Caddy and Traefik already implement ACME

OST

Mail Protocols - Introduction

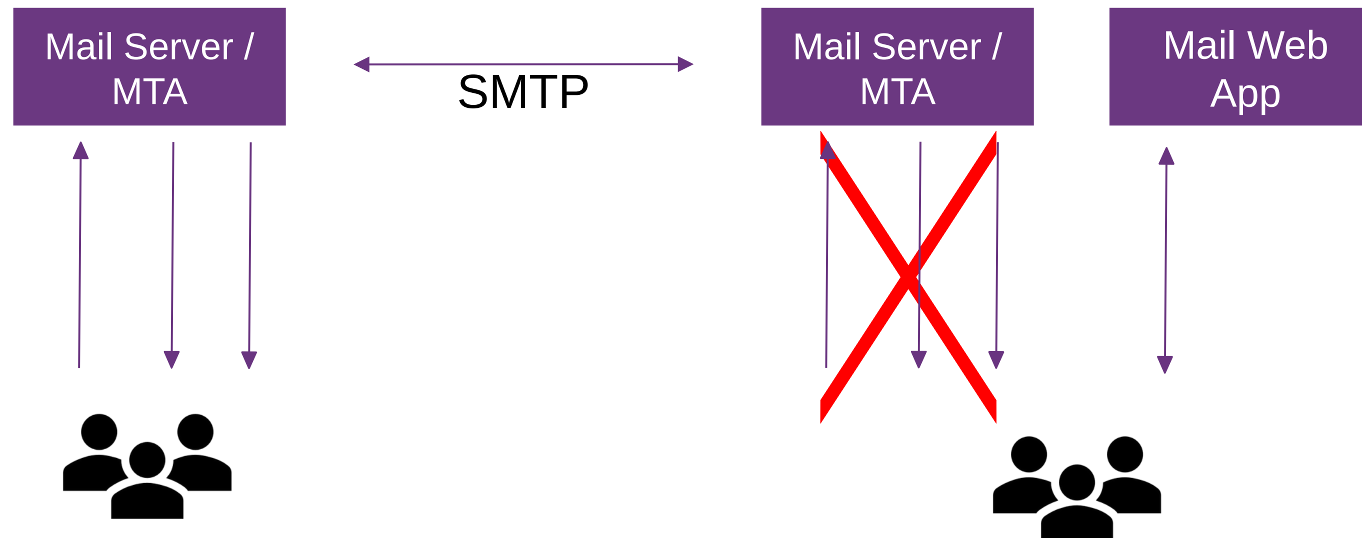
- Understanding how mail works
  - Basics of Email, SPF, DMARC, DKIM, and their importance in email security.
- Email: backbone of personal, academic, and professional correspondence worldwide
  - Alternatives growing: Slack, Discord, Teams, Telegram, WhatsApp, Treema, Matrix, ...
- Core protocols:
  - SMTP specified in 1982
  - POP specified in 1984
  - IMAP specified in 1988
  - All protocols improved over time



OST

POP / SMTP / IMAP

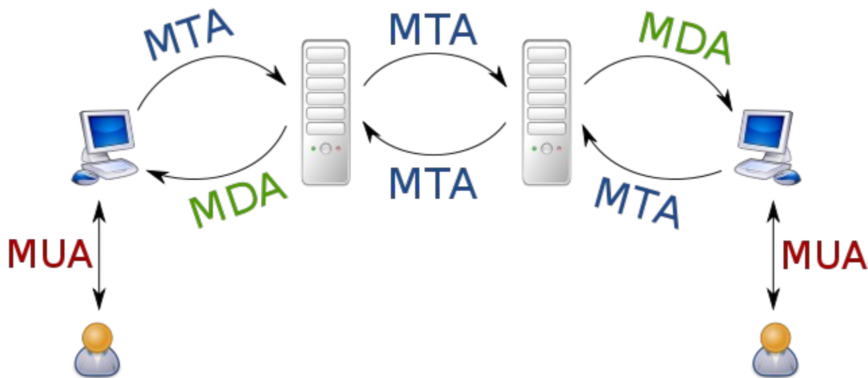
- Traditional mail not that popular anymore
  - Thunderbird now ~0.17%
- SMTP → Simple Mail Transfer Protocol
  - Standard protocol for sending/receiving emails
- Access to email mostly via Web – easier
  - Mailserver to mailserver still SMTP
  - JMAP to the rescue?
    - Combind IMAP / CardDav / CalDav
    - Integrated calendar / contact functionality



OST

SMTP

- IMAP / POP is dead? (I’m still using it :)
- Port Numbers
  - Port 25 for non-encrypted communication,
  - Port 587 for encrypted submission from email
  - Port 465 (deprecated but still in use) for SMTPS (SMTP over SSL)



https://commons.wikimedia.org/wiki/File:MTA-MDA-MUA_relationship.svg

- Mail flow: MUA (Mail User Agent) → MTA (Mail Transfer Agent) → MDA (Mail Delivery Agent) → recipient's mailbox
- Routing: domain == local → MDA; external → forward to recipient's MTA
- DNS MX records for mail server discovery

Secure SMTP / Spam

- STARTTLS (port 25/587): upgrades plain text to TLS on same port
  - (+) Backward compatible, recommended standard
  - (−) Can fall back to unencrypted → vulnerable to downgrade attacks
- SMTPS (port 465): entire session encrypted from start (SSL/TLS)
  - (+) Guaranteed encryption from the beginning
  - (−) Port confusion due to deprecation/revival history

- Greylisting: temporarily rejects emails from unknown senders
  - First email from unknown sender → temporary rejection ("try again later")
  - Legitimate servers retry after delay (per SMTP standard)
  - On retry → sender recognized, email accepted
  - Subsequent emails pass without delay
- (+) Reduces spam, low resource usage, simple to implement
- (−) Delays legitimate emails initially; spammers may adapt

Spam Prevention

- SURBL (Spam URI Real-time Blocklists): checks URLs in emails against blacklists
- DNS Blocklists (DNSBL): lists of IP addresses known to send spam
  - E.g., UCEPROTECT
  - Mail servers query DNSBLs in real-time to accept/reject
- Bayesian analysis: machine learning to classify emails by content

- Training data: junk/not-junk emails
- Word probability: "discount" → likely spam; "meeting" → likely not
- E.g., Thunderbird adaptive junk filter

Junk Settings

Selection

☑ Enable adaptive junk mail controls for this account

If enabled, you must first train Thunderbird to identify junk mail by using the Junk toolbar button to mark messages as junk or not. You need to identify both junk and non junk messages. After that Thunderbird will be able to mark junk automatically.

OST

Spam Prevention

- SPF (Sender Policy Framework): domain owner specifies allowed sending servers (DNS TXT record)
  - Receivers verify sender IP against SPF record → prevents spoofing
- DKIM (DomainKeys Identified Mail): digital signature linked to domain's DNS
  - Verifies email integrity and sender authenticity
- DMARC (Domain-based Message Authentication, Reporting and Conformance):
  - Builds on SPF + DKIM, adds policy (reject/quarantine/allow) and reporting
- BIMI (Brand Indicators for Message Identification):
  - Publish logo (SVG) in DNS TXT record; requires VMC in most cases