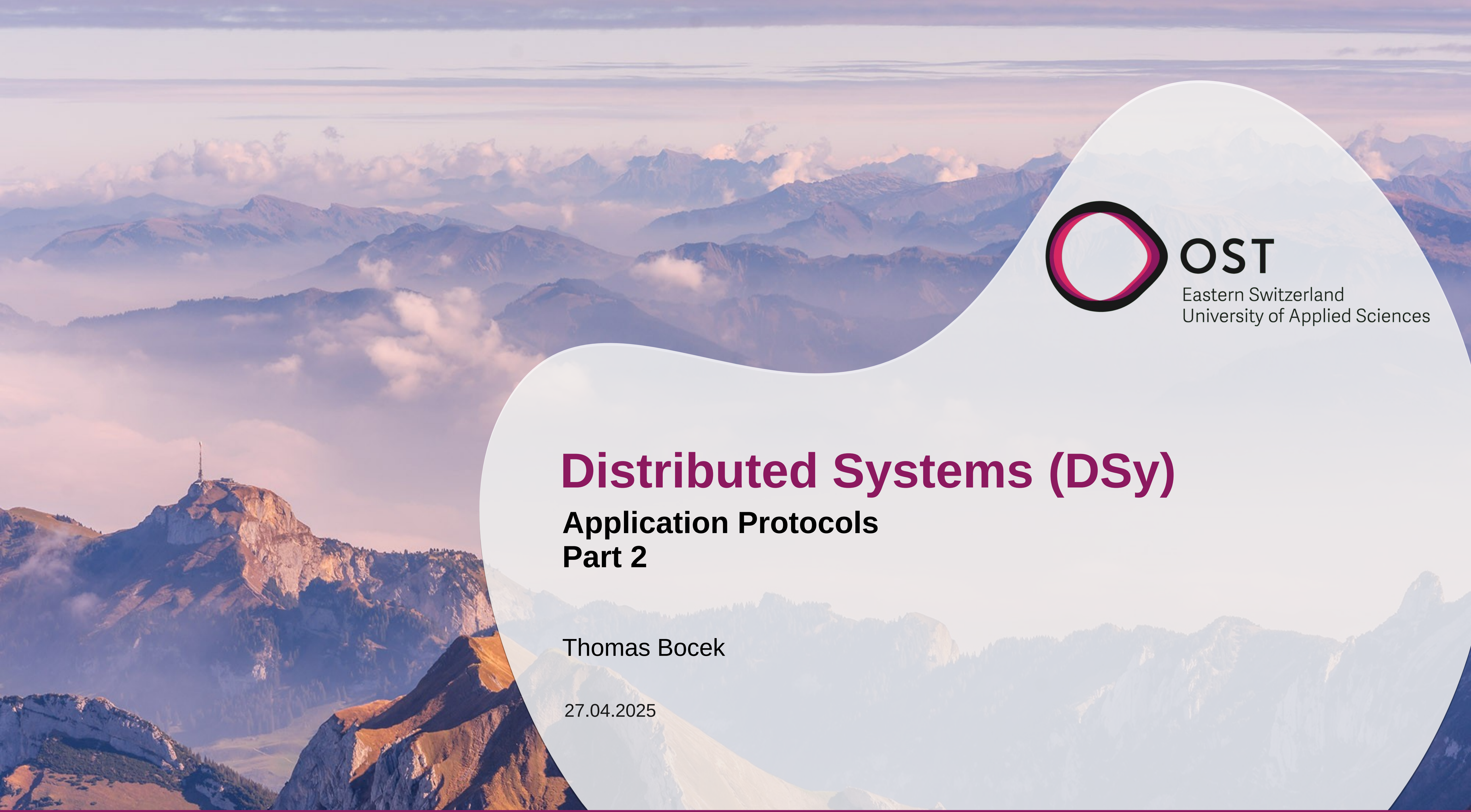


Eastern Switzerland
University of Applied Sciences

Distributed Systems (DSy)

Application Protocols
Part 2

Thomas Bocek

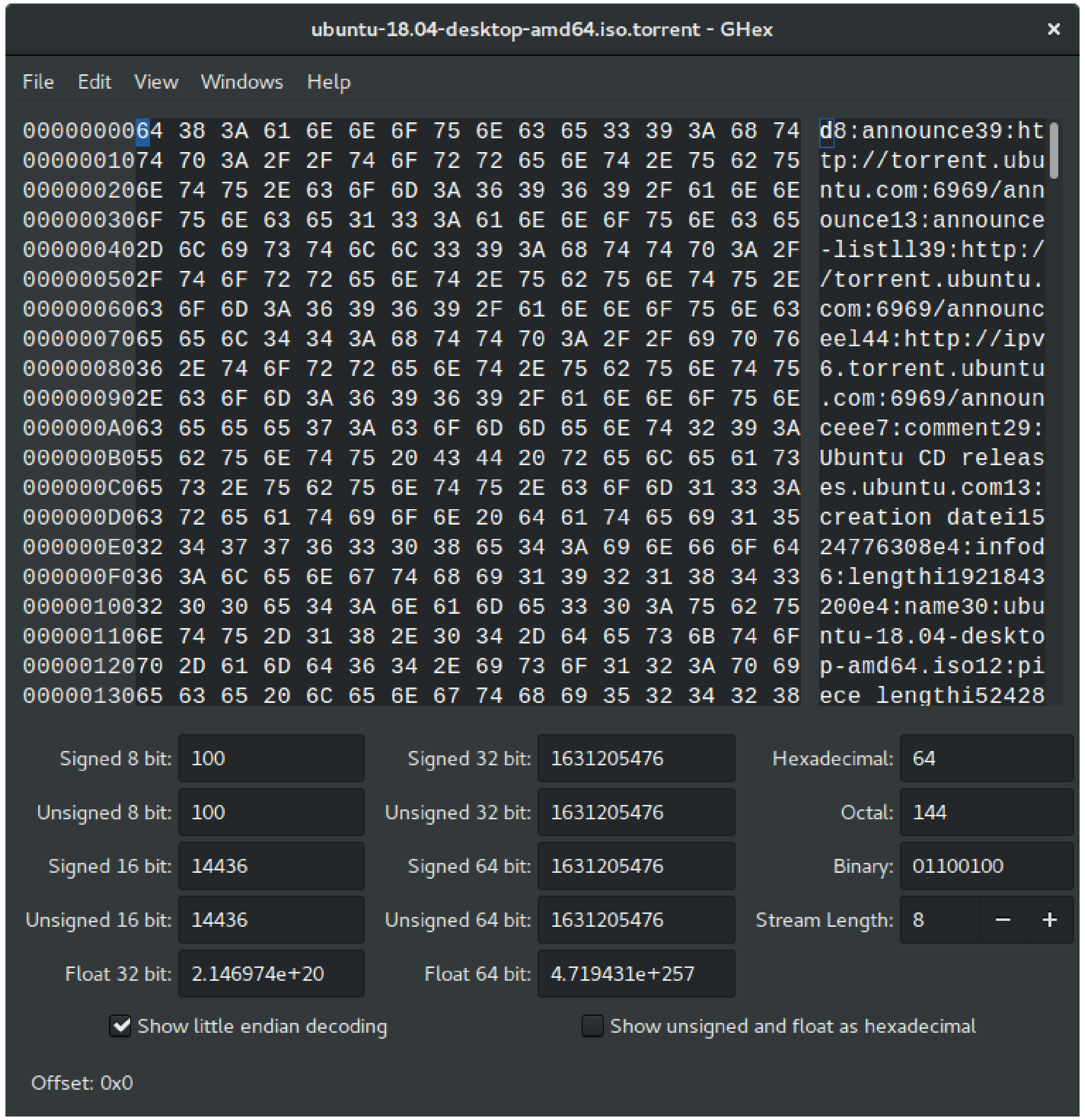
27.04.2025

Learning Goals

- Lecture 10 (Application Protocols)
  - Important protocols on layer 9
  - Part 1: custom protocols, RPC, HTTP, JSON, WebSockets, Server Sent Events
  - Part 2: Bencoding, WebRTC, DNS

Protocols Bencoding and Others

- Benconding
  - Integers: i42e, Byte string: 4:test, list: l4:testi42ee
  - Map/dictionary: d4:test3:ost3:tomi42ee
- Use: BitTorrent
- UBJSON
- Cap’n Proto , FlatBuffers
  - Do not serialize, just copy, little-endian
- Apache Arrow
  - Do not serialize, copy, and optimally layout for memory access
- … and many others
- Benchmarks, benchmarks, …



```
0000000064 38 3A 61 6E 6E 6F 75 6E 63 65 33 39 3A 68 74 d8:announce39:ht
0000001074 70 3A 2F 2F 74 6F 72 72 65 6E 74 2E 75 62 75 tp://torrent.ubu
000000206E 74 75 2E 63 6F 6D 3A 36 39 36 39 2F 61 6E 6E ntu.com:6969/ann
000000306F 75 6E 63 65 31 33 3A 61 6E 6E 6F 75 6E 63 65 ounce13:announce
000000402D 6C 69 73 74 6C 6C 33 39 3A 68 74 74 70 3A 2F -listll39:http:/
000000502F 74 6F 72 72 65 6E 74 2E 75 62 75 6E 74 75 2E /torrent.ubuntu.
0000006063 6F 6D 3A 36 39 36 39 2F 61 6E 6E 6F 75 6E 63 com:6969/announc
0000007065 65 6C 34 34 3A 68 74 74 70 3A 2F 2F 69 70 76 eel44:http://ipv
0000008036 2E 74 6F 72 72 65 6E 74 2E 75 62 75 6E 74 75 6.torrent.ubuntu
000000902E 63 6F 6D 3A 36 39 36 39 2F 61 6E 6E 6F 75 6E .com:6969/announ
000000A063 65 65 65 37 3A 63 6F 6D 6D 65 6E 74 32 39 3A ceee7:comment29:
000000B055 62 75 6E 74 75 20 43 44 20 72 65 6C 65 61 73 Ubuntu CD releas
000000C065 73 2E 75 62 75 6E 74 75 2E 63 6F 6D 31 33 3A es.ubuntu.com13:
000000D063 72 65 61 74 69 6F 6E 20 64 61 74 65 69 31 35 creation datei15
000000E032 34 37 37 36 33 30 38 65 34 3A 69 6E 66 6F 64 24776308e4:infod
000000F036 3A 6C 65 6E 67 74 68 69 31 39 32 31 38 34 33 6:lengthi1921843
0000010032 30 30 65 34 3A 6E 61 6D 65 33 30 3A 75 62 75 200e4:name30:ubu
000001106E 74 75 2D 31 38 2E 30 34 2D 64 65 73 6B 74 6F ntu-18.04-deskto
0000012070 2D 61 6D 64 36 34 2E 69 73 6F 31 32 3A 70 69 p-amd64.iso12:pi
0000013065 63 65 20 6C 65 6E 67 74 68 69 35 32 34 32 38 ece lengthi52428
```

Signed 8 bit:	100	Signed 32 bit:	1631205476	Hexadecimal:	64
Unsigned 8 bit:	100	Unsigned 32 bit:	1631205476	Octal:	144
Signed 16 bit:	14436	Signed 64 bit:	1631205476	Binary:	01100100
Unsigned 16 bit:	14436	Unsigned 64 bit:	1631205476	Stream Length:	8
Float 32 bit:	2.146974e+20	Float 64 bit:	4.719431e+257		

[✓] Show little endian decoding   [ ] Show unsigned and float as hexadecimal

Offset: 0x0

WebRTC – Introduction

- WebRTC for browser to browser communication
  - P2P, no server involved (~mostly)
  - Real time communication (RTC) via API
  - Main 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
- Supported initially by Chrome, Firefox (now many others)
  - First cross-browser call in February 2013

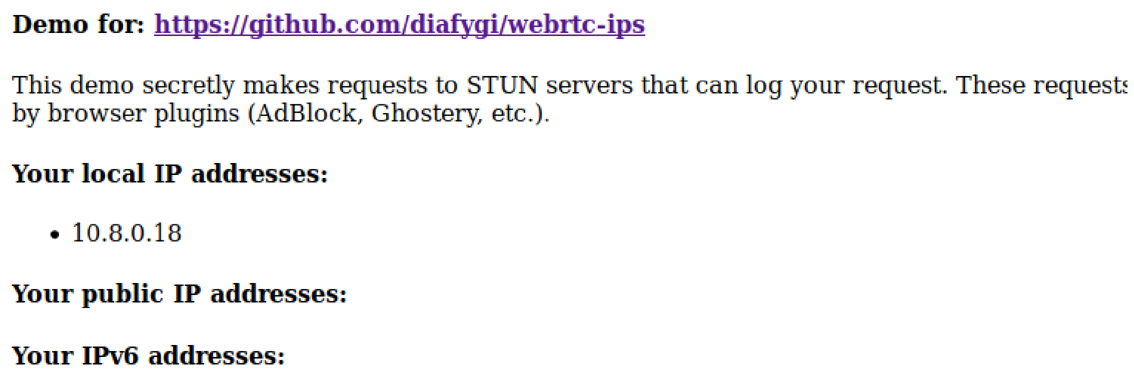
- Compatibility, 97%
- PeerJS
  - Not all browser work 100% compatbile with each other → PeerJS
  - «PeerJS provides a complete, configurable, and easy-to-use peer-to-peer API built on top of WebRTC»

WebRTC – Introduction

- Filling gap in the Web-Experience
  - Video Chat → Google Hangouts Plugin, Flash, Java
  - Multimedia / Conferences → Expensive, proprietary 3rd party apps
  - Customer Service → Chat only, 3rd party plugins/apps
- WebRTC widely deployed, no client necessary!
- Used in WhatsApp, Facebook Messenger, whereby.com
- Standard finalized: 26.01.2021
  - W3C Recommendation (W3C process flow)
  - «The RTCPeerConnection API has endured three design iterations on this topic over the years. As a result, each browser today implements a snapshot from a different point in the timeline of an evolving spec.» (source) (other updates) (large messages, fixed)

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, SRTP
  - Complexity - SCTP over DTLS over UDP



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

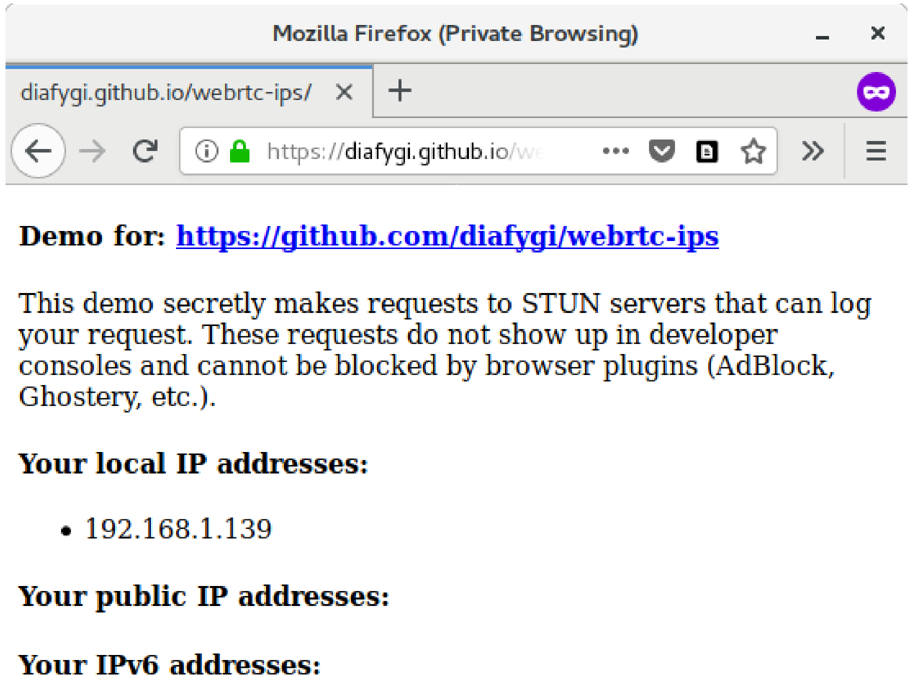
This demo secretly makes requests to STUN servers that can log your request. These request
by browser plugins (AdBlock, Ghostery, etc.).

Your local IP addresses:

- 10.8.0.18

Your public IP addresses:

Your IPv6 addresses:


Mozilla Firefox (Private Browsing)
diafygi.github.io/webrtc-ips/
https://diafygi.github.io/w

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:

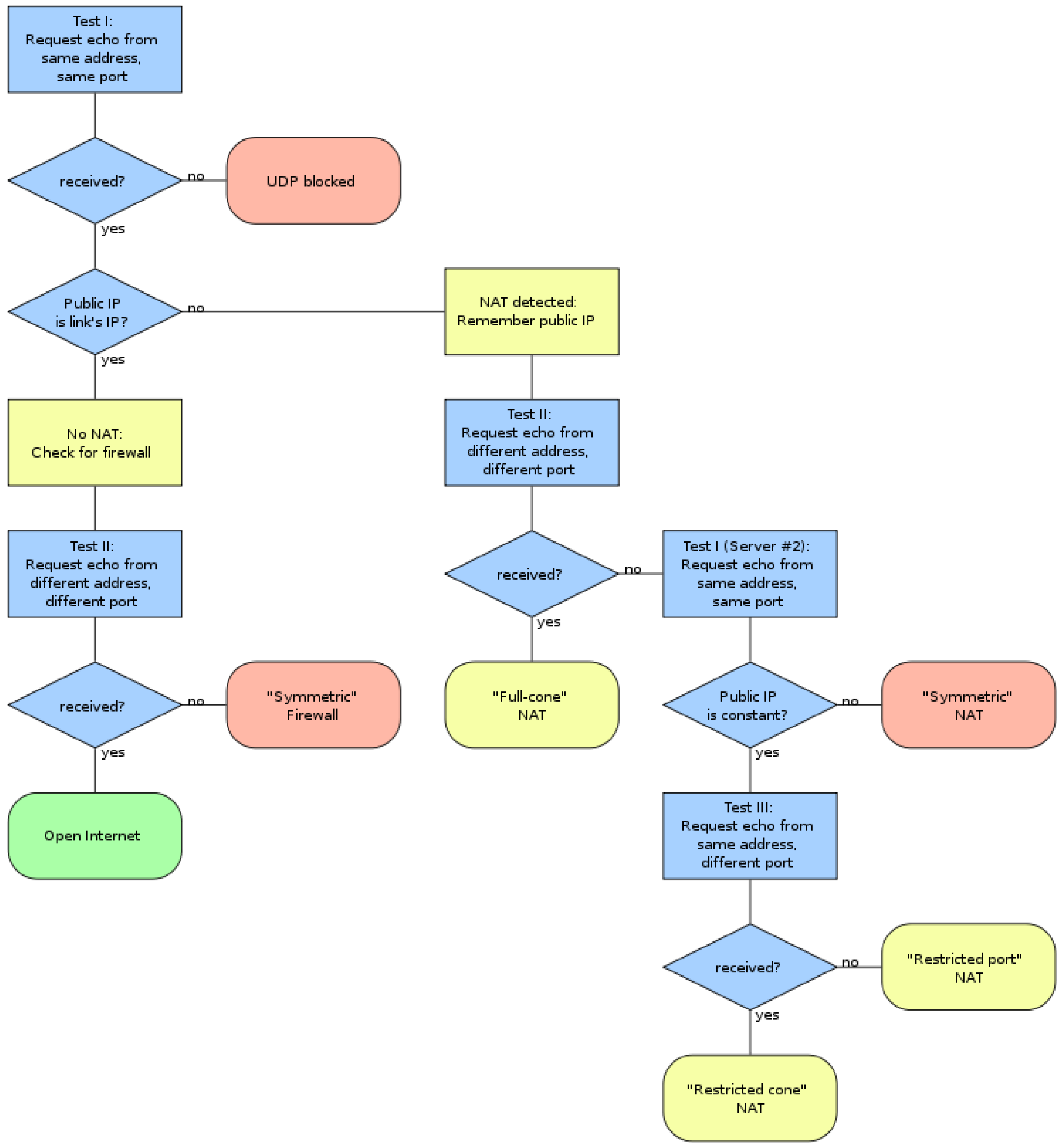
- 192.168.1.139

Your public IP addresses:

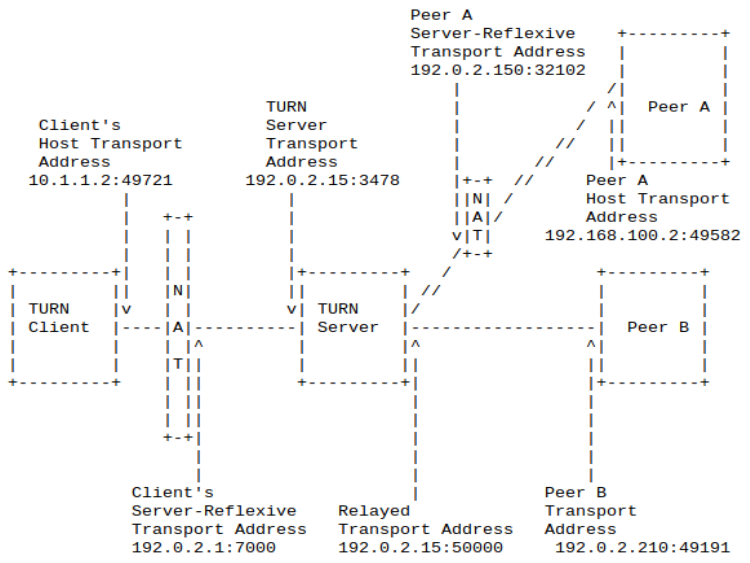
Your IPv6 addresses:

WebRTC – Introduction

- On the bright side: developer does not need to care about NAT
  - 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: **t**raversal **u**sing **r**elays around **N**AT



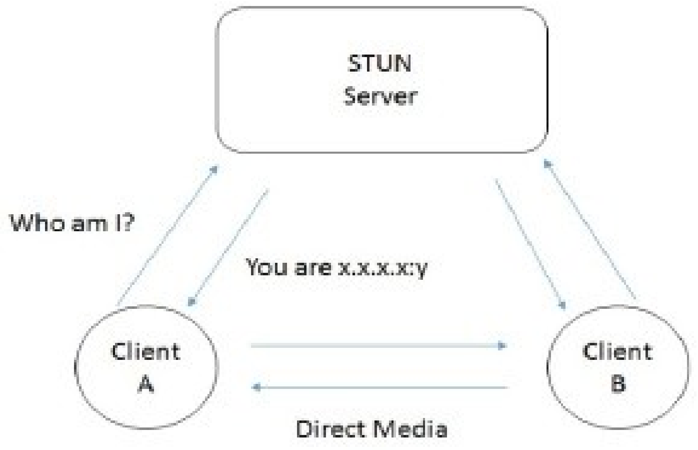
Test I: Request echo from same address, same port
received?
no
UDP blocked
yes
Public IP is link's IP?
no
NAT detected: Remember public IP
yes
No NAT: Check for firewall
Test II: Request echo from different address, different port
received?
no
"Symmetric" Firewall
yes
Open Internet
Test II: Request echo from different address, different port
received?
yes
"Full-cone" NAT
no
Test I (Server #2): Request echo from same address, same port
Public IP is constant?
no
"Symmetric" NAT
yes
Test III: Request echo from same address, different port
received?
no
"Restricted port" NAT
yes
"Restricted cone" NAT

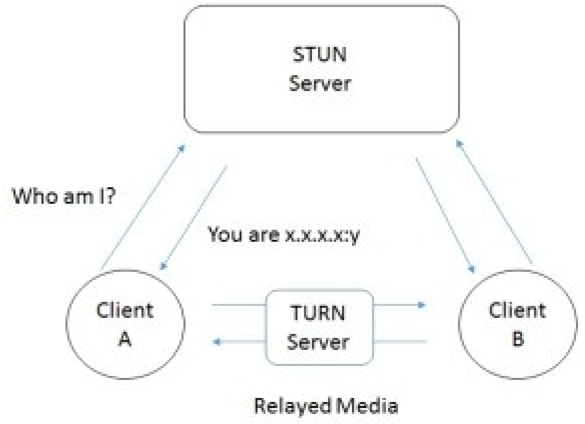


Peer A Server-Reflexive Transport Address 192.0.2.150:32102
Peer A
Client's Host Transport Address 10.1.1.2:49721
TURN Server Transport Address 192.0.2.15:3478
NAT
Peer A Host Transport Address 192.168.100.2:49582
TURN Client
NAT
TURN Server
Peer B
Client's Server-Reflexive Transport Address 192.0.2.1:7000
Relayed Transport Address 192.0.2.15:50000
Peer B Transport Address 192.0.2.210:49191

WebRTC – Introduction

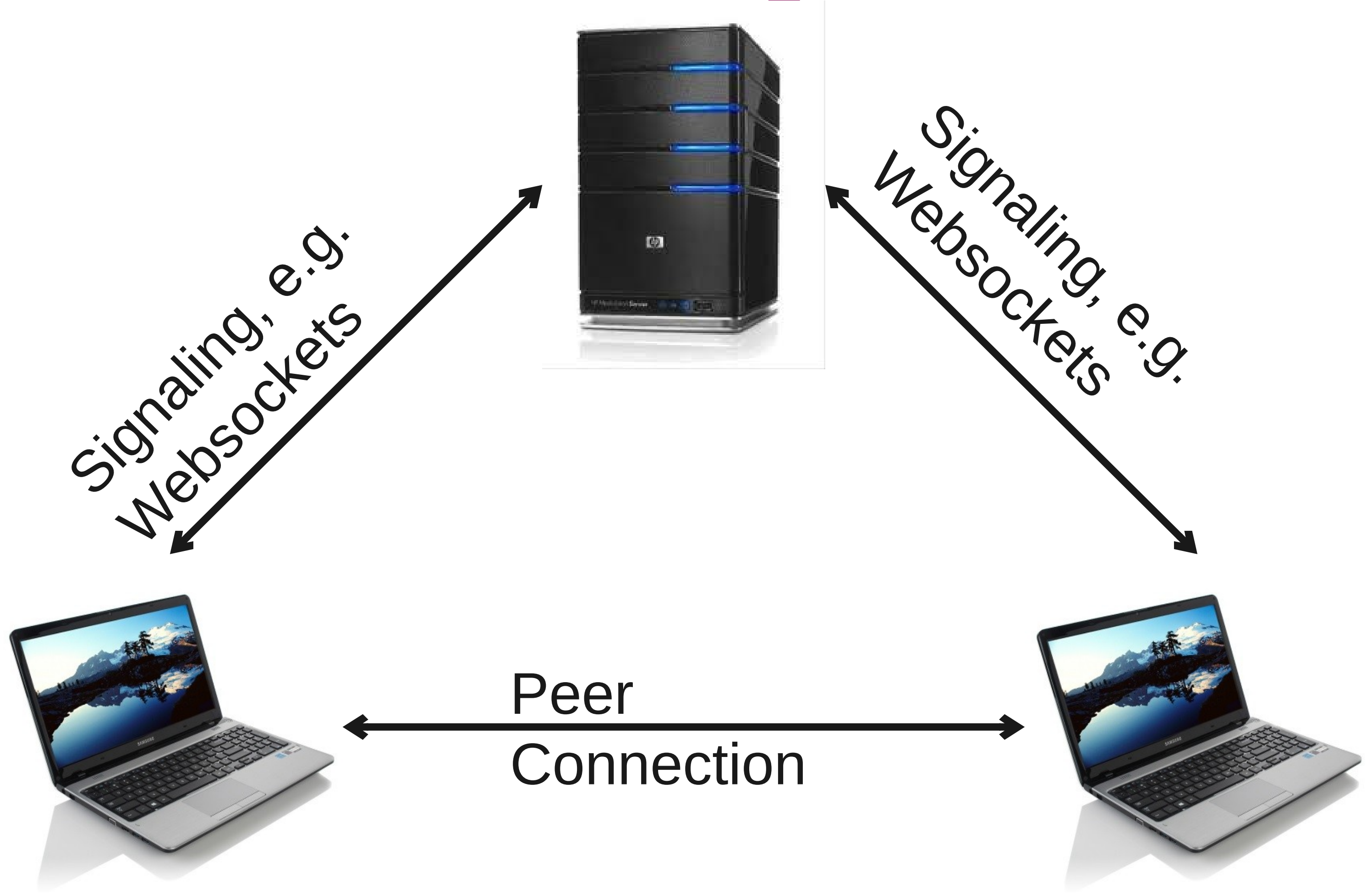
- TURN
  - TURN client/server uses UDP, TCP/TLS
    - Some firewalls block UDP entirely
  - UPnP / NAT-PMP setup by browser optional?
    - Bugzilla@Mozilla – Bug 860045
- ICE - Interactive Connectivity Establishment
  - RFC 8445 a protocol for NAT traversal
  - "ICE works by exchanging a multiplicity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





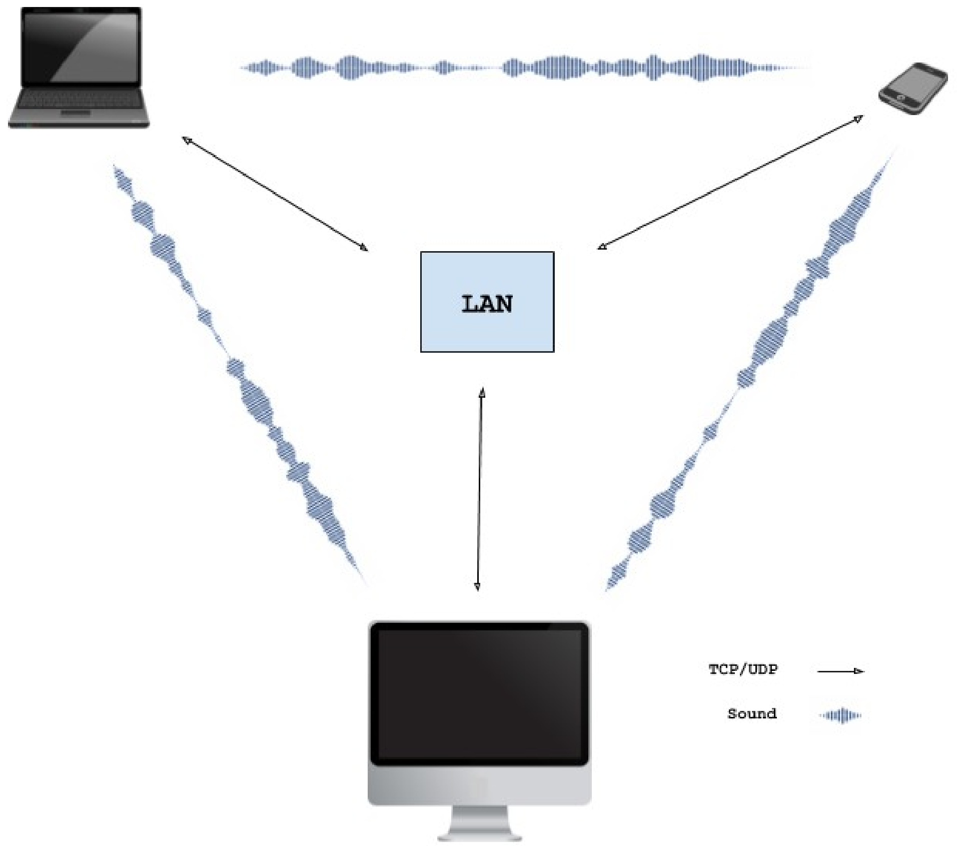
source

WebRTC Architecture - Triangle



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



WebRTC - Demo

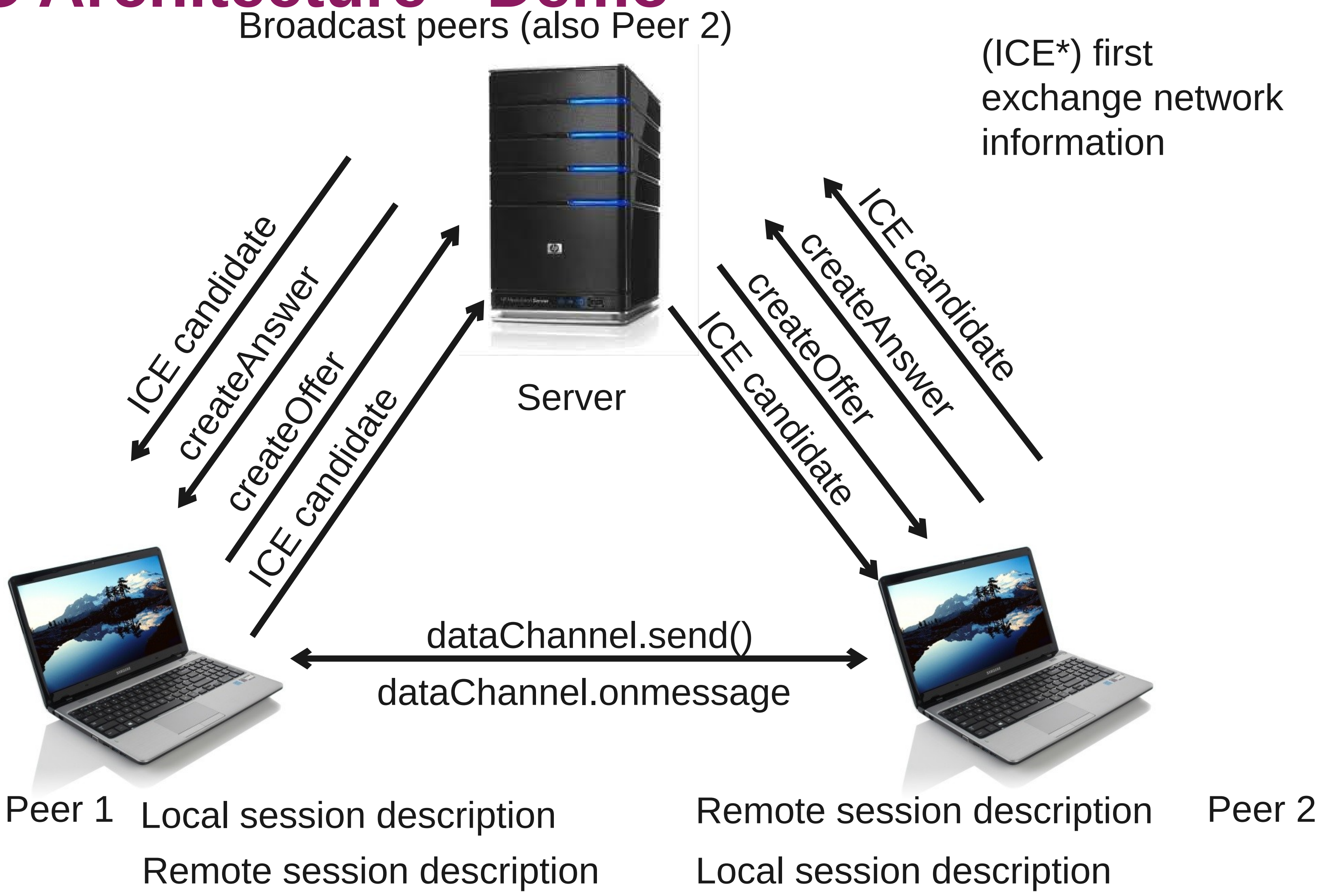
- Server - server.js
  - Node.js server
  - Serves files (index.html / express)
  - Listens to incoming WS messages and broadcast messages to all connected clients
  - 26 loc
- npm i / node server.js
- Minimal example!

```
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.');
```

Example: https://github.com/tbocek/DSy/webrtc

WebRTC Architecture - Demo

(ICE*) first
exchange network
information

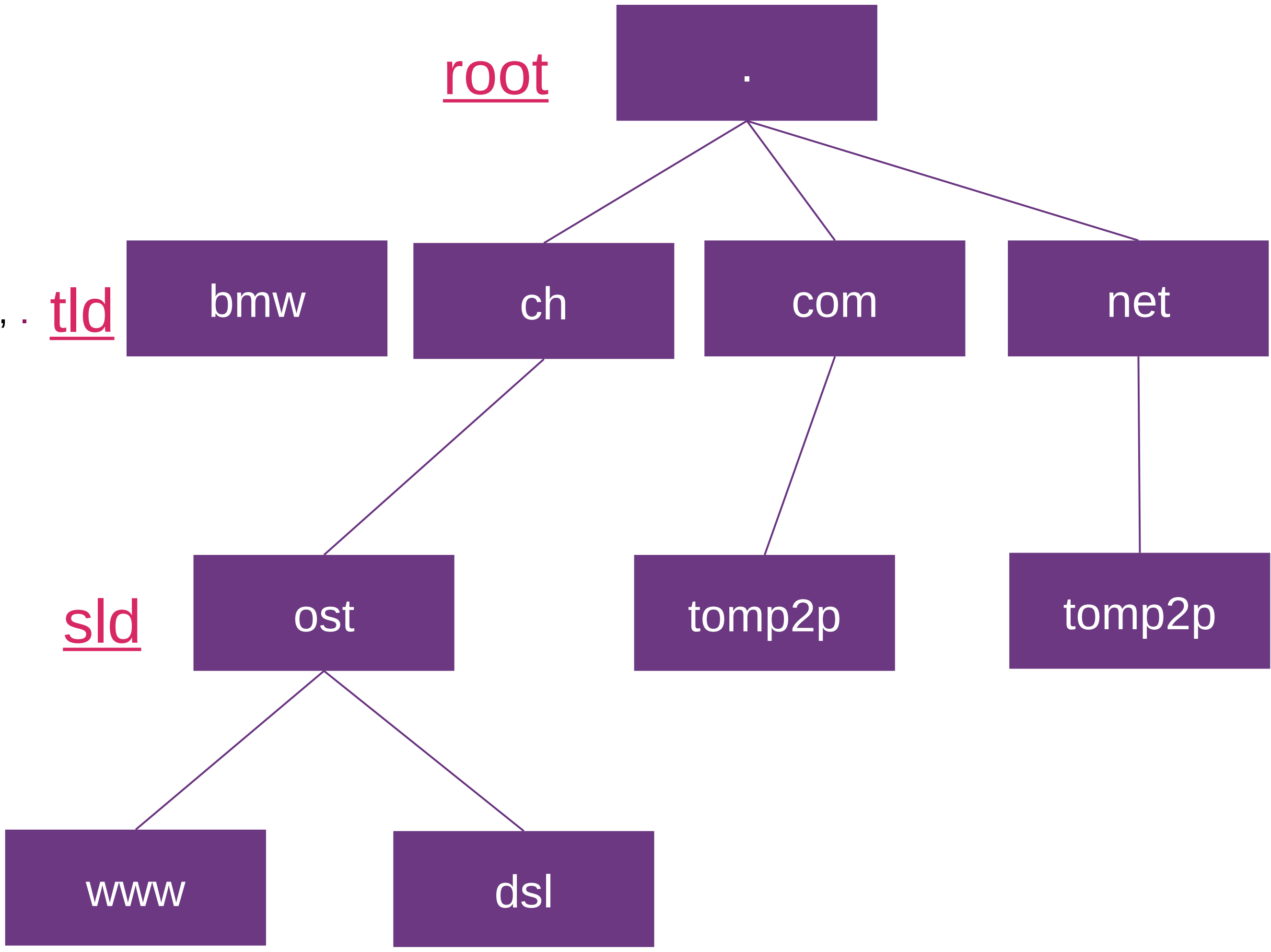


WebRTC – Outlook

- Strong focus on VoIP
  - Skype competitor? MS Teams?
  - Microsoft / IE / Edge and WebRTC?
- Fewer plugins (flash, java), fewer registrations
- Mandatory Codecs: VP8, H264
- Web-based P2P frameworks
  - http://peerjs.com - make the API simpler, is it complex?

Application Protocol: DNS

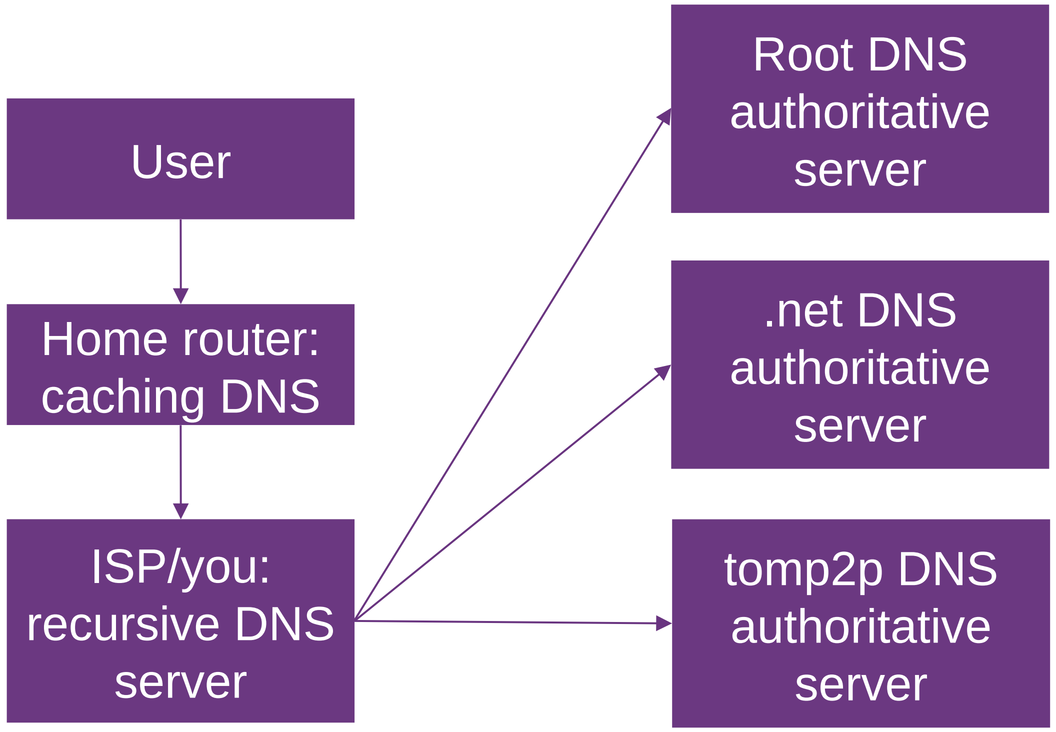
- Translates human readable domain names to IP addresses
  “phonebook of the Internet”
  - Delegate authority over sub-domains to other name servers
- Lots of new TLD: .zuerich, .bmw, .americanexpress, .youtube, .გე (application fee 185k USD) - not widely used
  - No special characters: ASCII (no UTF)
  - But, 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: exchange of hosts.txt
  - Does not scale



Application Protocol: DNS

- Primary + secondary DNS in case of failure
  - Secondary DNS gets data from primary
- 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., tomp2p.net (e.g., bind/nsd)
    - Authoritative DNS service 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, ~1000 root servers around the world

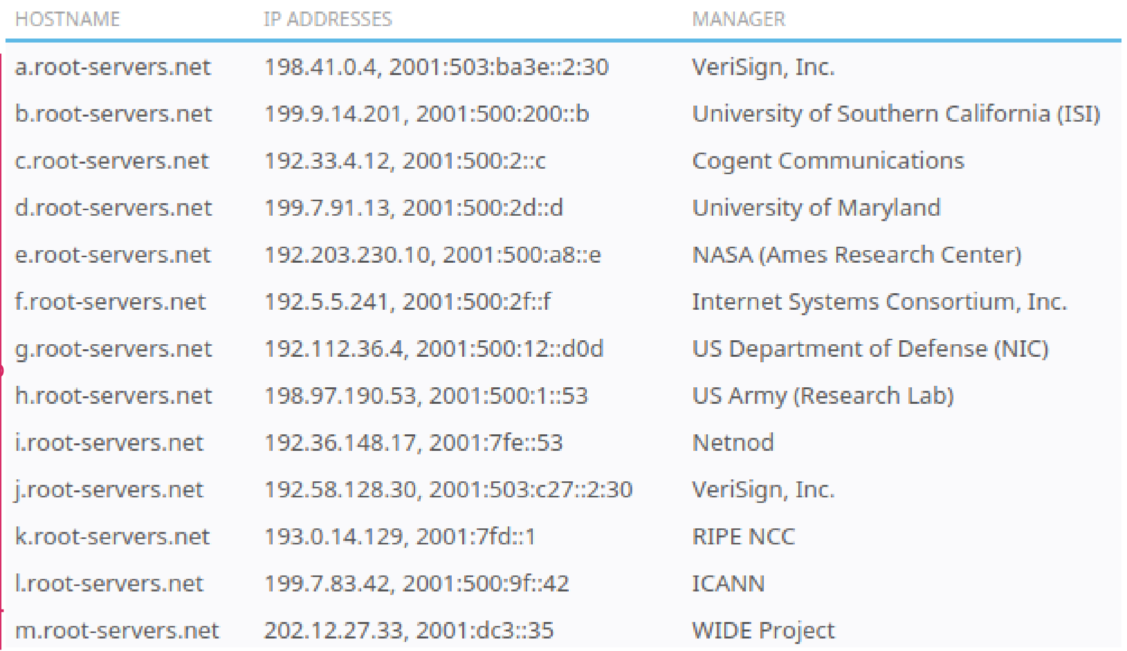
- E.g. BIND


User
Home router: caching DNS
ISP/you: recursive DNS server
Root DNS authoritative server
.net DNS authoritative server
tomp2p DNS authoritative server

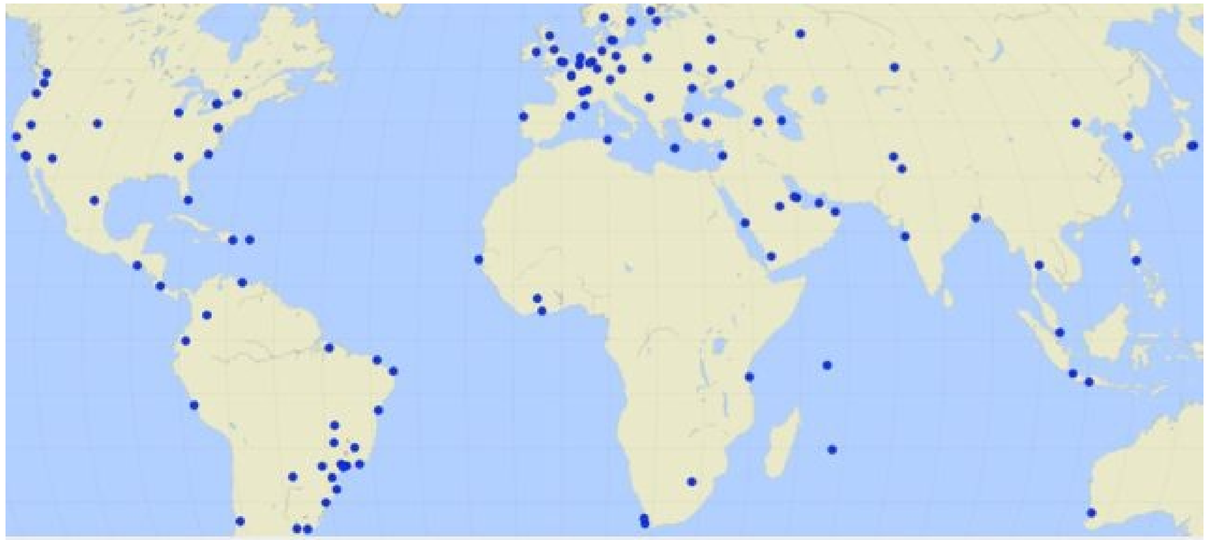
Application Protocol: DNS

- l.root-servers.net , 1 root IP with anycast mirrored in 138 locations
  - All root servers
- 2015: Internet DNS servers withstand huge DDoS attack [link]
  - 5m requests/s – some DNS could handle it
- Root zone is controlled by the United States Department of Commerce, operations by ICANN
- 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".

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 tomp2p.net.
@ SOA ns.nope.ch. root.nope.ch. (2018030404 8H
2H 4W 3H)
                NS              ns.nope.ch.
                NS              ns.jos.li.
                MX      10      mail.nope.ch.
                A               188.40.119.115
                TXT             "v=spf1 mx
-all“
www             A               188.40.119.115
bootstrap       A               188.40.119.115
$INCLUDE "/etc/opendkim/keys/mail.txt“
$INCLUDE "/etc/bind/dmarc.txt"
```

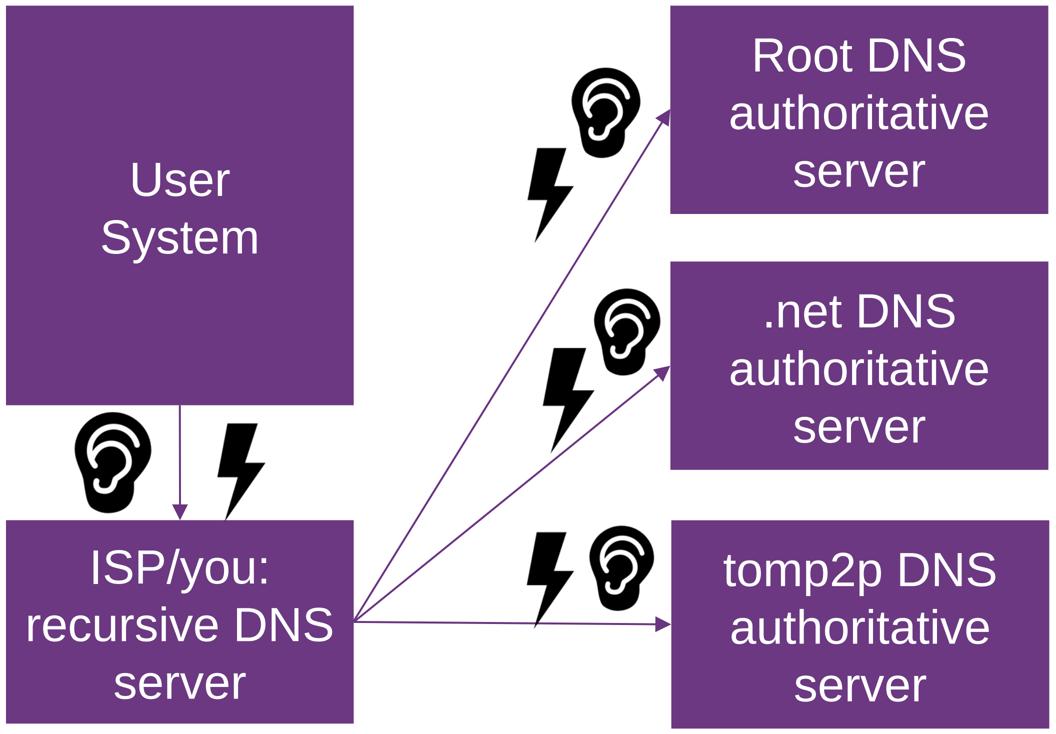
Application Protocol: DNS

- 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 tomp2p.net

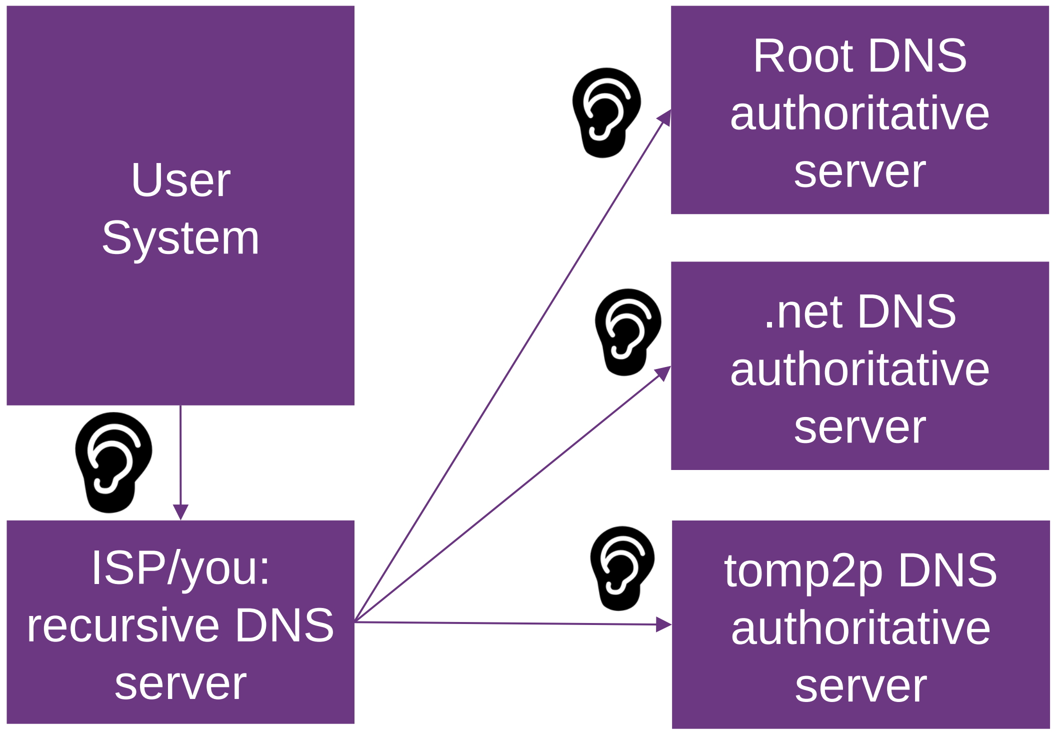
- New record types: RRSIG, DNSKEY, DS, …
  - RRSIG, sign all resource sets
  - DS (delegation signer) record in the parent zone
    - dig DS tomp2p.net
  - ZSK to sign RRset
    - How to validate ZSK?
  - KSK to sign ZSK pub key
    - With 2 keys, its easier to change ZSK

Application Protocol: DNS

- DNS

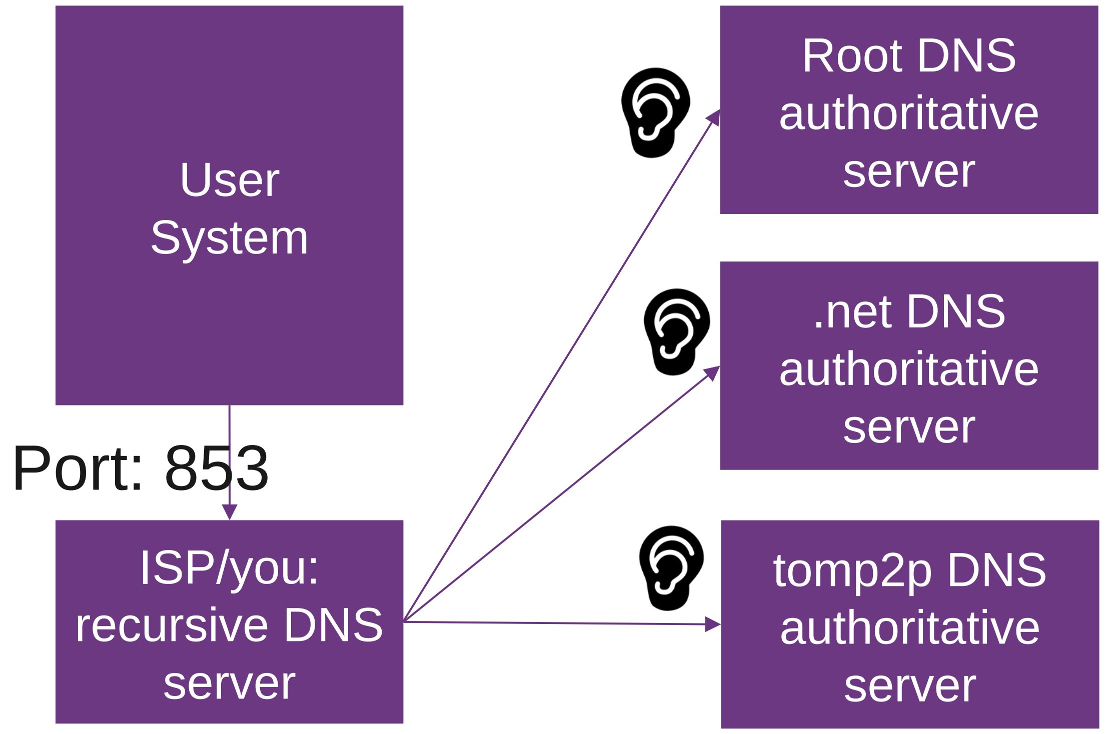

User System
ISP/you: recursive DNS server
Root DNS authoritative server
.net DNS authoritative server
tomp2p DNS authoritative server

- DNSSEC

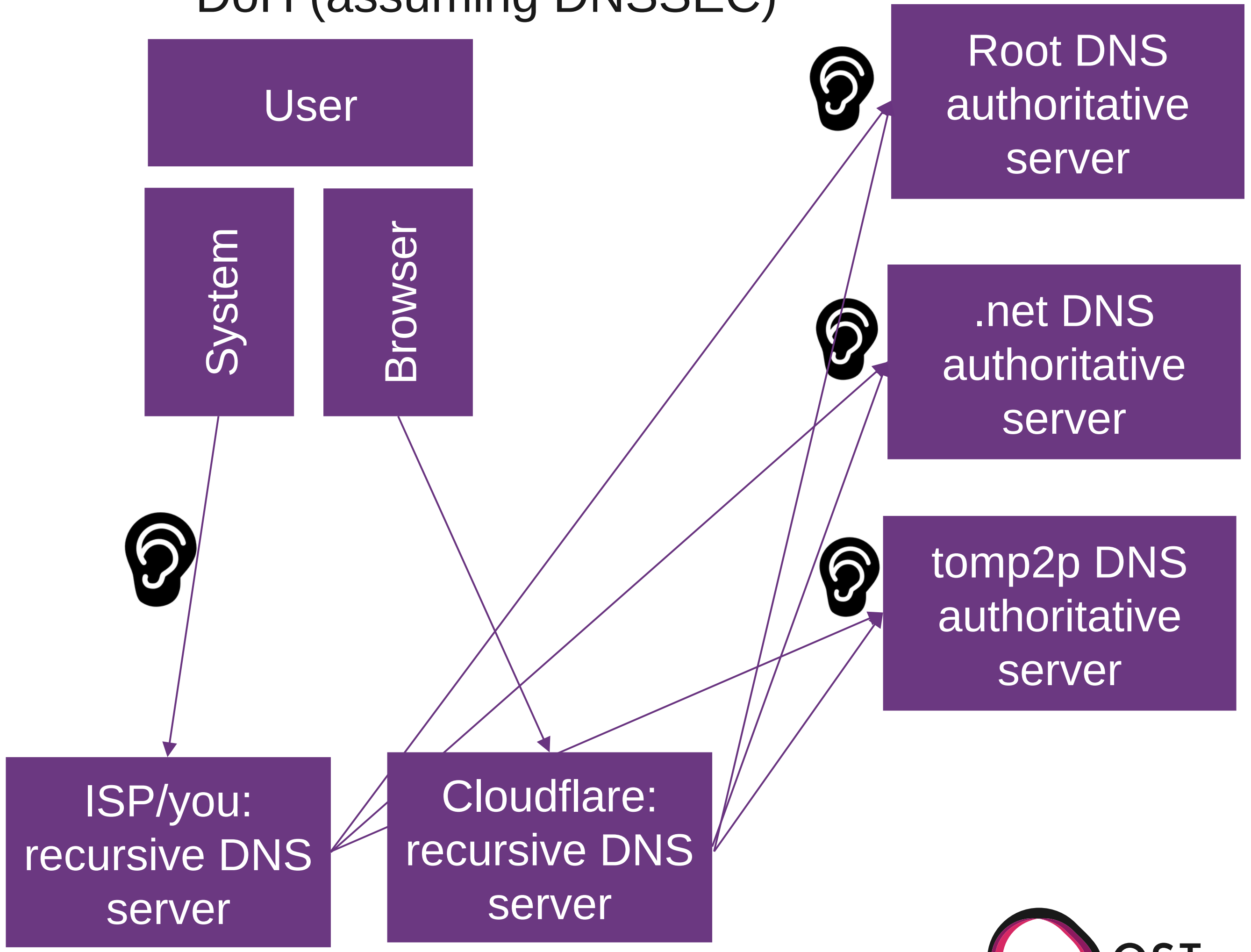

User System
ISP/you: recursive DNS server
Root DNS authoritative server
.net DNS authoritative server
tomp2p DNS authoritative server

Application Protocol: DNS

- DoT (assuming DNSSEC)


User System
Port: 853
ISP/you: recursive DNS server
Root DNS authoritative server
.net DNS authoritative server
tomp2p DNS authoritative server

- DoH (assuming DNSSEC)


User
System
Browser
ISP/you: recursive DNS server
Cloudflare: recursive DNS server
Root DNS authoritative server
.net DNS authoritative server
tomp2p DNS authoritative server

DoH vs. DoT

DoH

- provides confidentiality of lookups in transit
- Uses standard HTTP/2, on the standard 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 issuing 5m certificates per day
- Certificates or domain-validation certificates. Cannot compete with traditional CA (identity checks)
- Certs have automated renewal
  - ACME protocol – challenge response
    - Automated Certificate Management Environment
  - Query Web servers or DNS servers (wildcard)

- Certbot – client for ACME
  - `certbot certonly --webroot -w /tmp -d ost.tomp2p.net --debug-challenges`
  - Copy the challenge where Let’s encrypt server can find it (in my case /var/www/html)
  - Nginx config

```
server_name ost.tomp2p.net;
  ssl_certificate
/etc/letsencrypt/live/ost.tomp2p.net/fullchain.pem;
  ssl_certificate_key
/etc/letsencrypt/live/ost.tomp2p.net/privkey.pem;
```

  - This needs to be automated!

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

  - Caddy and Traefik already implement ACME