IPv6-only im Heimnetz

... und was dabei alles kaputtgeht

s3lph

s3lph@s3lph.me

@s3lph@chaos.social

@s3lph:kabelsalat.ch

CoSin 2025

Über mich

- s3lph (he/him)
- Linux Systems Engineer
- Macht `$dinge` im CCC Basel
  - Ansible
  - Python
  - E-Mail
- www.s3lph.me
- **Kein Netzwerker!**

Was bisher geschah?

1995 IPv6 in RFC 1883 standardisiert
2011 IANA exhausted
2012 World IPv6 Launch Day
2015 ARIN exhausted
2019 RIPE exhausted

IPv6 in einer Slide

- 128-bit Adressen, Hexadezimal-Notation
  - $8 \cdot 16$ bit *Hextets*, mit `:` getrennt: `2001:0db8:0ccc:0000:0000:0000:0000:0042`
  - Führende Nullen in Hextets werden weggekürzt: `2001:db8:ccc:0:0:0:0:42`
  - Längste Kette an Nullen wird durch `::` ersetzt: `2001:db8:ccc::42`
- Nur Präfixlänge (z.B. /64), keine Subnetzmasken
- Adressvergabe: *Stateless Address Autoconfiguration* (SLAAC)
  - Router schickt ICMPv6 *Router Advertisement* mit Präfix
  - Client vergibt sich Adresse innerhalb des Präfix selbst
  - DHCPv6 etc. gibt es auch, hier aber nicht relevant
- Wichtige Präfixe

`::1/128`	localhost	`fe80::/64`	Link Local, analog `169.254.0.0/16`
`::/0`	Defaultroute	`fd00::/8`	Unique Local, analog RFC 1918
`2000::/3`	Globally Unique	`ff00::/8`	Multicast

IPv6 in einer Slide

- 128-bit Adressen, Hexadezimal-Notation
  - $8 \cdot 16$ bit *Hextets*, mit : getrennt: `2001:0db8:0ccc:0000:0000:0000:0000:0042`
  - Führende Nullen in Hextets werden weggekürzt: `2001:db8:ccc:0:0:0:0:42`
  - Längste Kette an Nullen wird durch `::` ersetzt: `2001:db8:ccc::42`
- Nur Präfixlänge (z.B. /64), keine Subnetzmasken
- Adressvergabe: *Stateless Address Autoconfiguration* (SLAAC)
  - Router schickt ICMPv6 *Router Advertisement* mit Präfix
  - Client vergibt sich Adresse innerhalb des Präfix selbst
  - DHCPv6 etc. gibt es auch, hier aber nicht relevant
- Wichtige Präfixe

`::1/128`	localhost	`fe80::/64`	Link Local, analog `169.254.0.0/16`
`::/0`	Defaultroute	`fd00::/8`	Unique Local, analog RFC 1918
`2000::/3`	Globally Unique	`ff00::/8`	Multicast

IPv6 in einer Slide

- 128-bit Adressen, Hexadezimal-Notation
  - $8 \cdot 16$ bit *Hextets*, mit : getrennt: `2001:0db8:0ccc:0000:0000:0000:0000:0042`
  - Führende Nullen in Hextets werden weggekürzt: `2001:db8:ccc:0:0:0:0:42`
  - Längste Kette an Nullen wird durch `::` ersetzt: `2001:db8:ccc::42`
- Nur Präfixlänge (z.B. /64), keine Subnetzmasken
- Adressvergabe: *Stateless Address Autoconfiguration* (SLAAC)
  - Router schickt ICMPv6 *Router Advertisement* mit Präfix
  - Client vergibt sich Adresse innerhalb des Präfix selbst
  - DHCPv6 etc. gibt es auch, hier aber nicht relevant
- Wichtige Präfixe

`::1/128`	localhost	`fe80::/64`	Link Local, analog `169.254.0.0/16`
`::/0`	Defaultroute	`fd00::/8`	Unique Local, analog RFC 1918
`2000::/3`	Globally Unique	`ff00::/8`	Multicast

IPv6 in einer Slide

- 128-bit Adressen, Hexadezimal-Notation
  - $8 \cdot 16$ bit *Hextets*, mit : getrennt: `2001:0db8:0ccc:0000:0000:0000:0000:0042`
  - Führende Nullen in Hextets werden weggekürzt: `2001:db8:ccc:0:0:0:0:42`
  - Längste Kette an Nullen wird durch `::` ersetzt: `2001:db8:ccc::42`
- Nur Präfixlänge (z.B. /64), keine Subnetzmasken
- Adressvergabe: *Stateless Address Autoconfiguration* (SLAAC)
  - Router schickt ICMPv6 *Router Advertisement* mit Präfix
  - Client vergibt sich Adresse innerhalb des Präfix selbst
  - DHCPv6 etc. gibt es auch, hier aber nicht relevant
- Wichtige Präfixe

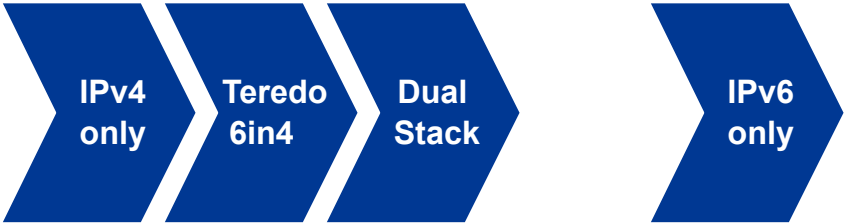
`::1/128`	localhost	`fe80::/64`	Link Local, analog `169.254.0.0/16`
`::/0`	Defaultroute	`fd00::/8`	Unique Local, analog RFC 1918
`2000::/3`	Globally Unique	`ff00::/8`	Multicast

Von IPv4 zu IPv6

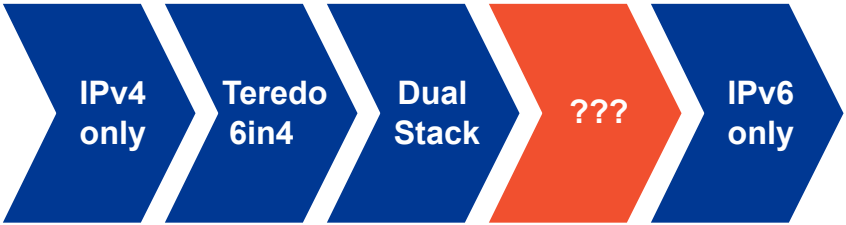
Von IPv4 zu IPv6

Von IPv4 zu IPv6

Von IPv4 zu IPv6



Von IPv4 zu IPv6



Motivation & Ziel

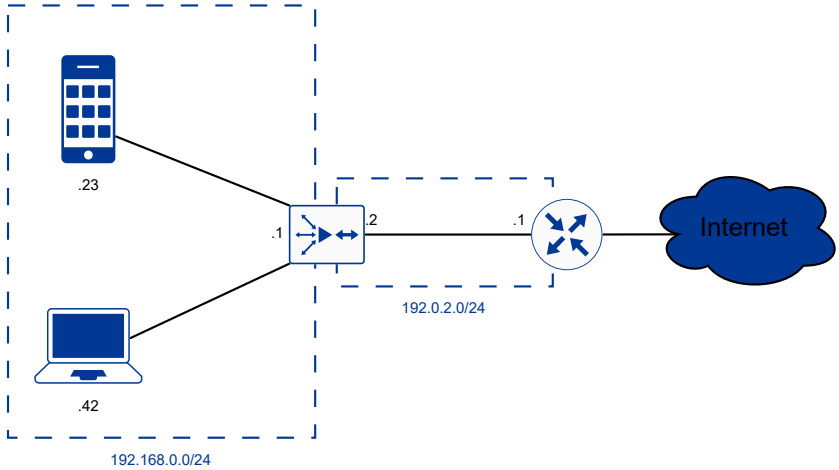
- Motivation: Übergangstechnologien verstehen
- Dual Stack im WAN
- IPv6-only im LAN
  - IPv4 nur noch für Geräte, die es unbedingt brauchen
- Trotzdem IPv4-only Dienste im Internet erreichen können
  - Beispiel: `github.com`

Mein Heimnetz

- Dual-Stack Uplink
  - 1 dynamische IPv4-Adresse
  - Statisches /48 IPv6-Prefix
- Gateway: Debian auf APU2
  - nftables
  - unbound
  - radvd
  - isc-dhcp-server
- Clients: Linux, Android

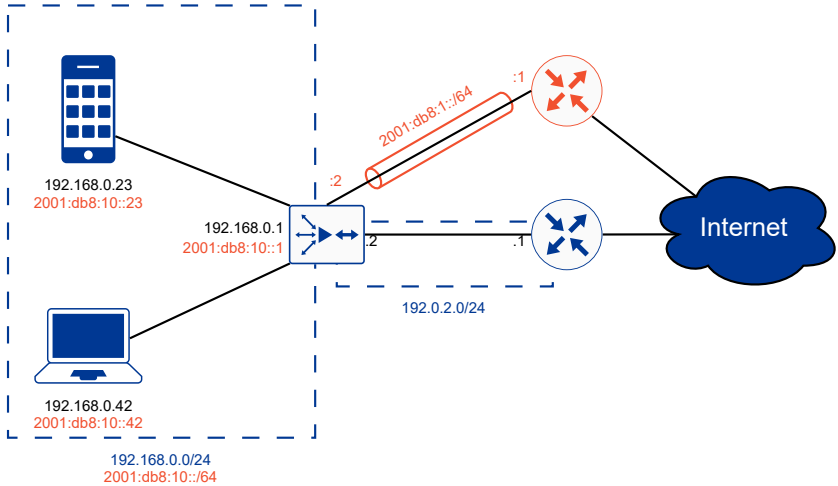
Von IPv4 zu IPv6

IPv4 only



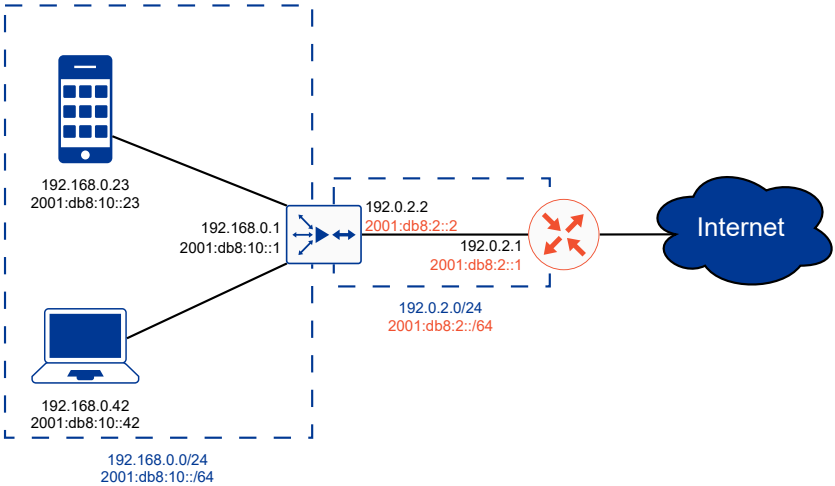
Von IPv4 zu IPv6

IPv6 via Tunnelprovider

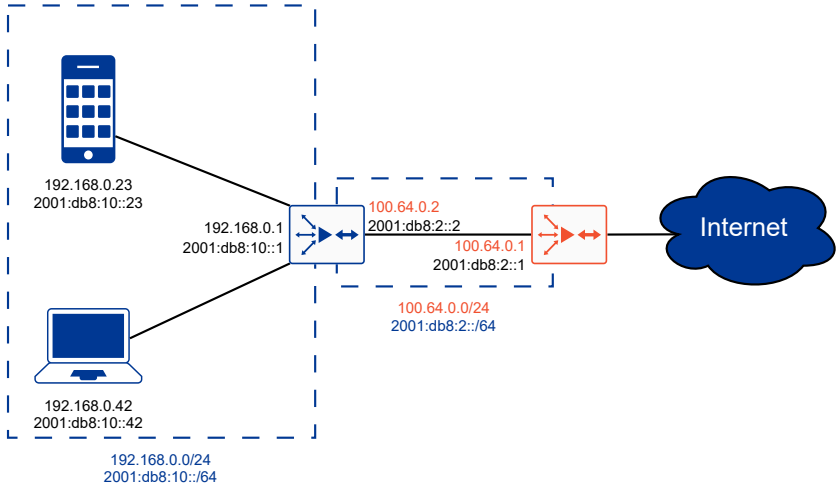


Von IPv4 zu IPv6

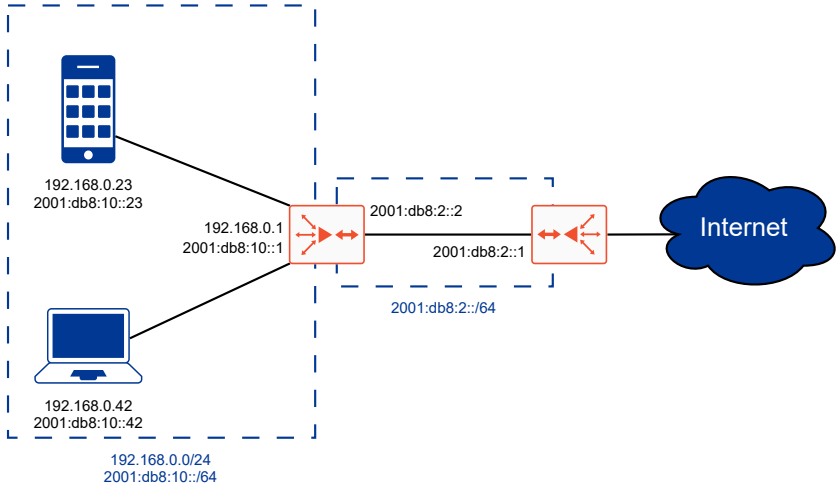
Dual Stack


192.168.0.23
2001:db8:10::23
192.168.0.1
2001:db8:10::1
192.0.2.2
2001:db8:2::2
192.0.2.1
2001:db8:2::1
Internet
192.0.2.0/24
2001:db8:2::/64
192.168.0.42
2001:db8:10::42
192.168.0.0/24
2001:db8:10::/64

DS-Lite


192.168.0.23
2001:db8:10::23
192.168.0.1
2001:db8:10::1
100.64.0.2
2001:db8:2::2
100.64.0.1
2001:db8:2::1
Internet
100.64.0.0/24
2001:db8:2::/64
192.168.0.42
2001:db8:10::42
192.168.0.0/24
2001:db8:10::/64

464XLAT


192.168.0.23
2001:db8:10::23
192.168.0.1
2001:db8:10::1
2001:db8:2::2
2001:db8:2::1
Internet
2001:db8:2::/64
192.168.0.42
2001:db8:10::42
192.168.0.0/24
2001:db8:10::/64

464XLAT

- RFC 6877, 2013
- 4-to-6-to-4 Translation
  - CLAT: **C**ustomer-side trans**LAT**ion, stateless IPv4 zu IPv6 NAT
  - PLAT: **P**rovider-side trans**LAT**ion, stateful IPv6 zu IPv4 NAT

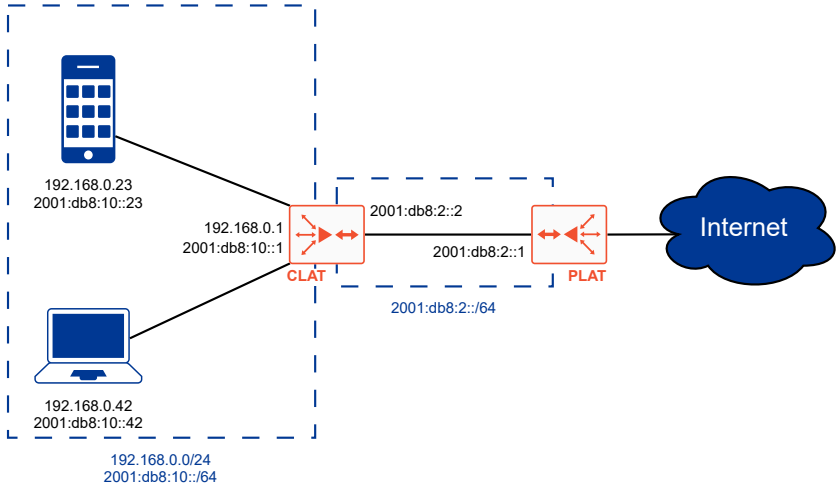
CLAT: SIIT

- RFC 6145, 2011
- ***S**tateless **I**P/**I**CMP **T**ranslation Algorithm*
- IPv4-Adressraum wird auf ein IPv6-Präfix abgebildet
  - Üblicherweise `64:ff9b::/96`
- Beispiel:
  - Vor CLAT: `192.168.0.42` →`140.82.121.3` (`github.com`)
  - Nach CLAT (mapped IPv4): `2001:db8::192.168.0.42` →`64:ff9b::140.82.121.3`
  - Nach CLAT (IPv6): `2001:db8::c0a8:2a` →`64:ff9b::8c52:7903`

PLAT: NAT64

- RFC 6146, 2011
- IPv4-embedded IPv6-Adresse wird zurück in IPv4 übersetzt
- Üblicherweise stateful; IPv4-Adresse des NAT64-Gateway als Source
- Beispiel:
  - Vor PLAT: `2001:db8::c0a8:2a` →`64:ff9b::8c52:7903`
  - Nach PLAT: `192.0.2.1` →`140.82.121.3`

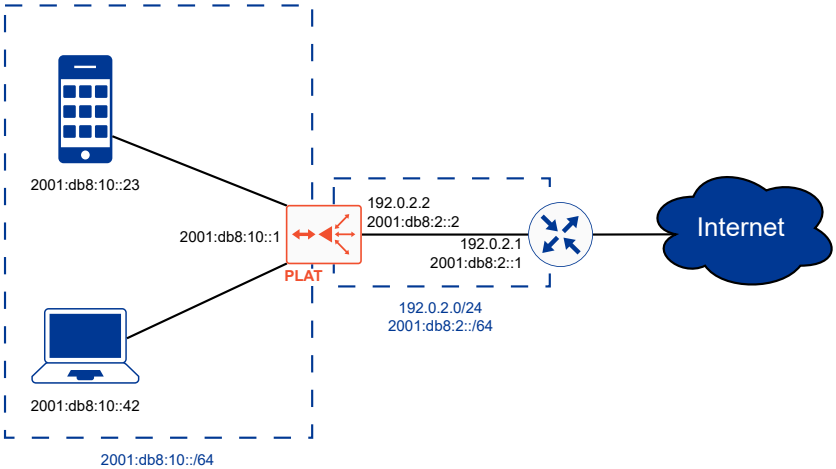
464XLAT


192.168.0.23
2001:db8:10::23
192.168.0.1
2001:db8:10::1
CLAT
2001:db8:2::2
2001:db8:2::1
PLAT
Internet
2001:db8:2::/64
192.168.0.42
2001:db8:10::42
192.168.0.0/24
2001:db8:10::/64

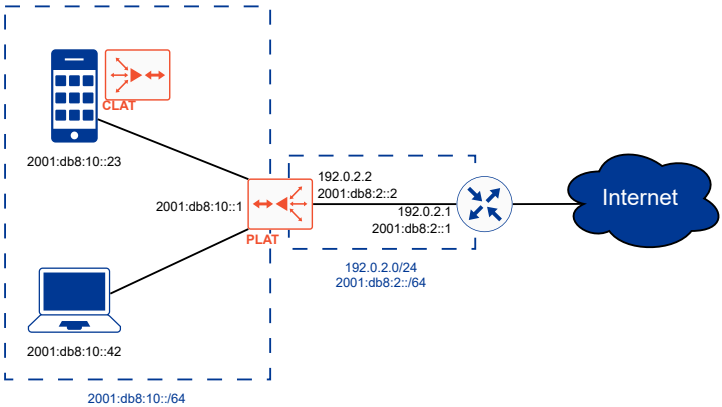
464XLAT

- Verbindet 2 IPv4-Netze via IPv6 miteinander
- Vielerorts anstelle DS-Lite verwendet
- Insbesondere in Mobilfunknetzen verbreitet

Ziel


2001:db8:10::23
2001:db8:10::1
PLAT
192.0.2.2
2001:db8:2::2
192.0.2.1
2001:db8:2::1
Internet
192.0.2.0/24
2001:db8:2::/64
2001:db8:10::42
2001:db8:10::/64

Option 1: CLAT im Endgerät



Option 1: CLAT im Endgerät

- Client gibt sich lokale IPv4-Adresse auf virtuellem Interface
- Applikationen «sehen» IPv4-Verbindung
  - Wird aber als IPv6-Paket ins Netzwerk gesendet
- Unterstützt in:
  - Android >= 4.3 (2013)
  - iOS >= 12.0 (2018)
  - macOS >= 13 (2022)
  - Windows >= 10 (2017, nur Mobilfunk)
  - NetworkManager: MR !2107, 2025

«IPv6-mostly»

- Unterstützte Geräte geben sich nur eine IPv6-Adresse
- Andere Geräte können weiterhin IPv4-Adressen verwenden
- Vorhandensein von NAT64 muss signalisiert werden

Signalisation «IPv6-mostly»

- *PREF64*
  - RFC 8781, 2020
  - IPv6 RA Option, um das NAT64-Prefix mitzuteilen
- DHCPv4 Option 108: «IPv6-Only Preferred»
  - RFC 8925, 2020
  - DHCPv4-Client holt sich kein Lease, falls IPv6 vorhanden
- Bei manchen Clients erforderlich, um CLAT zu aktivieren

Option 2: Endgeräte anlügen

- DNS64
- RFC 6147, 2011
- Umsetzung im DNS-Resolver
- Synthetisierter AAAA-Record mit NAT64-Prefix
  - Tut nichts, falls ein «echter» AAAA-Record existiert
  - Gibt auch A-Record zurück

```
github.com.   IN   AAAA   64:ff9b::8c52:7904
github.com.   IN   A      140.82.121.4
```

NAT64

- Für Linux gibt es 2 Implementationen
- Tayga
  - Im Userspace via tun-Device
- Jool
  - Im Kernelspace via netfilter
- Tayga schafft auf der APU2 nur etwa 200 Mbps
- Auswahl fällt auf Jool

NAT64

/etc/jool/jool.conf:

```
{
  "instance": "wan",
  "framework": "netfilter",
  "global": {
    "pool6": "64:ff9b::/96"
  }
}
```

```
systemctl restart jool.service
```

PREF64

/etc/radvd.conf:

```
interface lan {
  # ...
  nat64prefix 64:ff9b::/96 {
    AdvValidLifetime 1800;
  };
};
```

```
systemctl restart radvd.service
```

DHCP Option 108

/etc/dhcp/dhcpd.conf:

```
option ipv6-only-preferred code 108 = unsigned integer 32;

subnet 10.10.10.0 netmask 255.255.255.0 {
  # ...
  option ipv6-only-preferred 3600;
}
```

```
systemctl restart isc-dhcp-server.service
```

DHCP Option 108

/etc/dnsmasq.conf:

```
dhcp-option=108,3600i
```

```
systemctl restart dnsmasq.service
```

DNS64

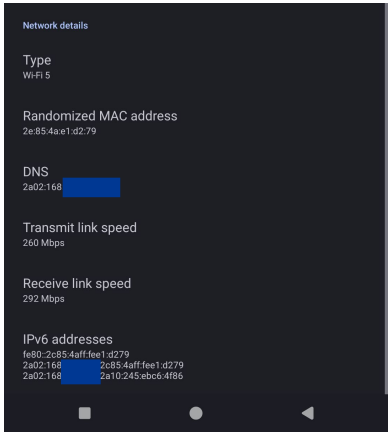
/etc/unbound/unbound.conf.d/server.conf:

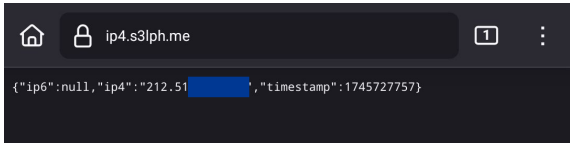
```
server:
  # ...
  module-config: "dns64 validator iterator"
```

```
systemctl restart unbound.service
```

Was funktioniert?

- Android mit Client-CLAT: Alles!





Was geht kaputt?

- Android & DHCPv4 Option 108
  - Einige Android-Vendor setzen einen DHCPv4 Lease voraus, um ein WLAN als «verbunden» zu betrachten
  - Android versteht die ipv6-only-preferred Option und holt sich keinen IPv4-Lease
  - ... und trennt dann die Verbindung, weil es ja keine Adresse bekommen hat
- `https://issuetracker.google.com/issues/291235541`

Was geht kaputt?

- Android & Router Advertisements
  - DNS-Resolver werden nur bei ausreichend langer RA-Lifetime übernommen
  - SLAAC-IP wird unabhängig von Lifetime gesetzt
  - Mit Dual Stack leicht zu übersehen
  - Ohne DNS-Resolver trennt Android die Verbindung

Was funktioniert?

- Mit DNS64:

```
$ host github.com
github.com has address 140.82.121.4
github.com has IPv6 address 64:ff9b::8c52:7904
```

Was funktioniert?

- DHCPv4 Option ipv6-only-preferred & NetworkManager
- Erst seit NetworkManager 1.52.0 (Februar 2025)
- Muss via nmcli aktiviert werden, nicht via GUI/TUI konfigurierbar:

```
$ nmcli con modify id <name> ipv4.dhcp-ipv6-only-preferred 1
NetworkManager: <info> dhcp4 (wlp2s0): received option
"ipv6-only-preferred": stopping DHCPv4 for 3600 seconds
```

DNS64: Was geht kaputt?

- Erstaunlich viel
- Teilweise auf sehr subtile Art

Was geht kaputt?

- Software, die nur IPv4 spricht
  - IPv4-Adressen hardcoded
  - Verwendung von `AF_INET` Sockets
  - Eigene DNS-Auflösung, die nur A-Records versteht
- Beispiel: Steam
  - Login geht, Spiele können gestartet werden
  - Sync zwischen Library und Store kaputt
  - Cloud Sync kaputt
- Kein Workaround

Was geht kaputt?

- Software, die kein *Happy Eyeballs* kann
- Beispiel: Gradle

```
> Could not GET 'https://repo.maven.apache.org/...'.
    > Got socket exception during request.
        > Network is unreachable
```

- Kein Workaround

Was geht kaputt?

- Hosts mit AAAA-Record, aber kaputtem IPv6
  - IPv6-only Client = kein Happy Eyeballs
  - AAAA-Record = kein synthetischer Record
- DNS64-Workaround in unbound:

```
server:
  # ...
  module-config: "dns64 validator iterator"
  dns64-ignore-aaaa: git.coredump.ch
```

Was geht kaputt?

- Unmodifizierte Antworten von zweitem DNS-Resolver
  - Beispiel: VPN
  - DNS-Query wird an alle Resolver gestellt
  - VPN-Resolver gibt nur IPv4-Adresse zurück
  - «Address unreachable»
- Workaround: systemd-resolved DNS Query Routing

systemd-resolved DNS Query Routing

- DNS Search Domains
  - bspw. `lan.example.org`
  - `foo` → `foo.lan.example.org`
- systemd-resolved fragt Upstream-Resolver nur nach Namen unterhalb der Search Domains
- Ausnahme: Interface mit der Defaultroute
- `resolvectl` gibt genaue Infos zur aktuellen Konfiguration
- `/etc/resolv.conf` auf `127.0.0.53` zeigen lassen

Was geht kaputt?

- systemd-resolved: DNS64 & DNSSEC
- systemd-resolved macht eigene DNSSEC-Validierung
- Synthetisierter AAAA-Record ist nicht signiert, *bogus*
- Workaround: `127.0.0.54` in /etc/resolv.conf eintragen
  - «proxy-only» DNS-Resolver ohne eigene Validierung

Fazit

- IPv6-only mit Client-CLAT sehr problemlos und einfach umsetzbar
- IPv6-only ohne Client-CLAT macht sehr viele Sachen kaputt
  - DNS64 verträgt sich nicht mit DNSSEC
  - Es gibt leider immer noch genug Software ohne IPv6-Support
- Die Zeit für IPv6-only ist noch nicht gekommen
  - Hoffnung auf CLAT-Support in NetworkManager
  - NAT64, PREF64 & DHCP Option 108 weiterhin aktiv
  - IPv6-mostly als Zwischenlösung

Literaturhinweise

- RIPE Labs: Deploying IPv6-Mostly Access Networks
- Jool: 464XLAT
- Jool: Node-Based Translation

Fragen?

s3lph@s3lph.me

@s3lph@chaos.social

@s3lph:kabelsalat.ch