RIPE NCC
RIPE NETWORK COORDINATION CENTRE

IPv6-mostly on OpenWRT

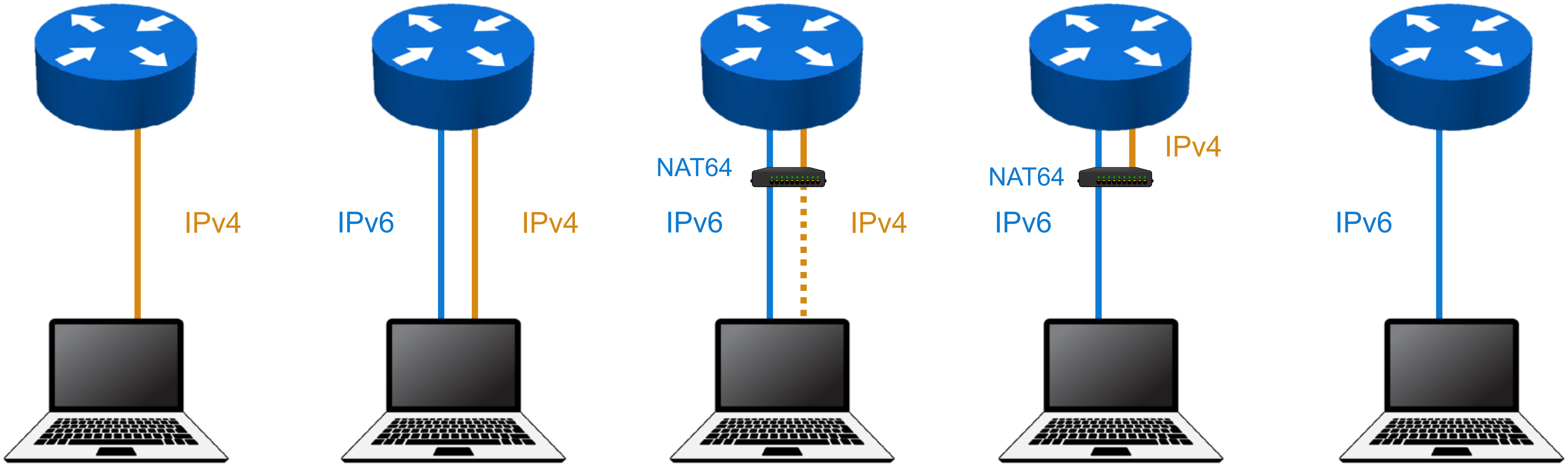
Running NAT64 / PREF64 / DNS64 / DHCP108 at home

Ondřej Caletka | RIPE 87 | 27 November 2023

What and why?

- You **have IPv4 and IPv6** at home
- Everything is **dual-stack**
- You would like to *gradually* get rid of IPv4
- You want to **see things break** so you can help fixing them
  - **spoiler alert**: you will not see any big breakages

Phased IPv6 transition



Prerequisites

- A dual-stack upstream connectivity with **delegated IPv6 space**
- A CPE capable of running OpenWRT, **preferably v23.05.2**
- Hardware tips:

Turris routers run TurrisOS which is based on somewhat older OpenWRT

GL-iNet routers come with firmware based on OpenWRT, can be easily replaced with vanilla OpenWRT release

What we are going to do

- Add an extra **IPv6-only network**
- Set up NAT64 using **Jool**
- Configure native **PREF64** support in OpenWRT
- Configure DHCP server to offer “**IPv6-only preferred**”
- Set up **DNS64** using Public DNS/Unbound/Knot Resolver
- Use **Ansible** to automate everything

IPv6-only Network

IPv6-only network

- Let’s keep the default network lan **dual-stack**
- We create another network lan6 without any IPv4 config
- We allocate a /60 IPv6 to that interface
  - first /64 would be used for directly connected devices
  - the rest will be available via DHCP-PD for **downstream routers**

```
config device
    option type 'bridge'
    option name 'br-lan6'
    option bridge_empty '1'
    list ports 'lan2'
```

```
config interface 'lan6'
    option proto 'static'
    option device 'br-lan6'
    option ip6assign '60'
    option ip6hint '60'
```

```
config dhcp 'lan6'
    option interface 'lan6'
    option ignore '1'
    option ra 'server'
    option dhcpv6 'server'
```

```
config zone
    option name ‘lan'
    …
    list network 'lan6'
```

What we have now

- Dual-stack network: business as usual
- IPv6-only network: no IPv4 support
  - ideal future Internet
  - a lot of things **work already**
  - but a lot of things also **do not work**

IPv6-only

PSK: tutorial

IPv6-mostly

PSK: tutorial

NAT64

Pretending everything is reachable over IPv6

NAT64

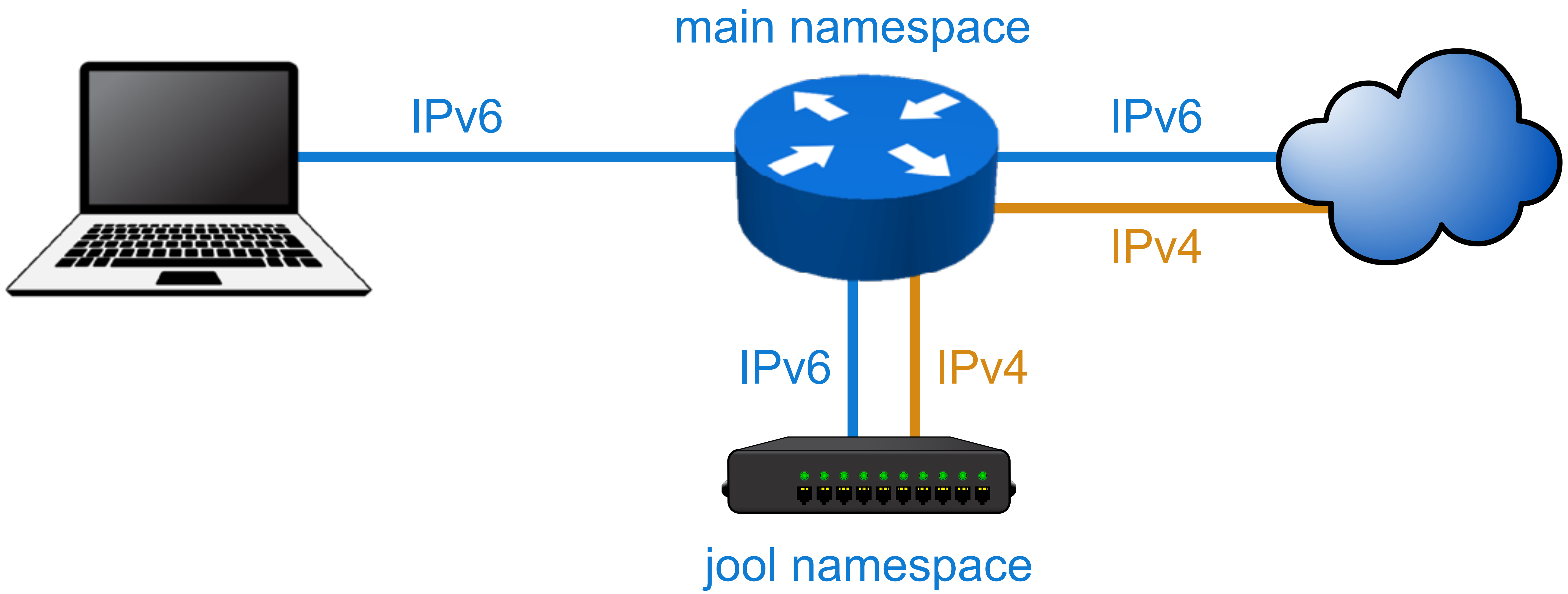
- A packet translator between IPv6 and IPv4
- Stateless or **stateful**
  - stateless is mostly useful for **providing IPv6 services** to IPv4-only clients
  - stateful is mostly useful to enable **IPv6-only clients** to reach **IPv4 services**
- Uses **Well-Known** or **Network-Specific Prefix**
  - No **private IPv4 addresses** allowed in Well-Known Prefix

Jool

- A Linux kernel-space implementation of NAT64
- Available in OpenWRT
- Not integrated into OpenWRT configuration system
- *Stealing* packets in the **PREROUTING**, injecting translated packets into **POSTROUTING**
  - Hard to enforce firewall rules
  - Translation not available for locally generated traffic

Jool in a network namespace

- Use veth pair to interconnect main and jool namespace
- No issues with firewall/locally generated content



Let’s set it up

Necessary packages

- `kmod-veth`
- `ip-full`
- `kmod-jool-netfilter`
- `jool-tools-netfilter`

```
#!/bin/sh
ip link add jool type veth peer openwrt
ip netns add jool
ip link set dev openwrt netns jool

ip netns exec jool sh <<EOF
        sysctl -w net.ipv4.conf.all.forwarding=1
        sysctl -w net.ipv6.conf.all.forwarding=1
        sysctl -w net.ipv6.conf.openwrt.accept_ra=2
        sysctl -w net.ipv4.ip_local_port_range="32768 32999"
        ip link set dev lo up
        ip link set dev openwrt up
        ip addr add dev openwrt 192.168.164.2/24
        ip addr add dev openwrt fe80::64
        ip route add default via 192.168.164.1
        modprobe jool
        jool instance add --netfilter --pool6 64:ff9b::/96
        jool global update lowest-ipv6-mtu 1500
        jool pool4 add 192.168.164.2 33000-65535 --tcp
        jool pool4 add 192.168.164.2 33000-65535 --udp
        jool pool4 add 192.168.164.2 33000-65535 --icmp
EOF
```

OpenWRT side

- We use IPv4 subnet 192.168.164.1/24
- We allocate one IPv6 /64 with SLAAC
- We route NAT64 prefix to fe80::64
- We put this interface to LAN firewall zone

```
config dhcp 'jool'
    option interface 'jool'
    option ra 'server'
    option ra_default '2'
    option ignore '1'
```

```
config interface 'jool'
   option device 'jool'
   option proto 'static'
   option ip6assign '64'
   option ip6hint '64'
   list ipaddr '192.168.164.1/24'

config route6 'nat64'
   option interface 'jool'
   option target '64:ff9b::/96'
   option gateway 'fe80::64'
```

Testing it

- `ping/traceroute 64:ff9b::<your favourite IPv4>`
- Make sure it works also from the connected devices
  - otherwise it might be a routing/firewall issue

PREF64

Letting everybody know that NAT64 is in place

PREF64

- Option in **Router Advertisement** messages carrying the **NAT64 prefix** the network is using
- Hosts can therefore send traffic there instead of native IPv4
  - Usually by means of **CLAT** - Customer-side translator between IPv4 and IPv6
- PREF64 is a **new feature** of OpenWRT v23.05.0

```
config dhcp 'lan6'
    option interface 'lan6'
    …
    option ra_pref64 '64:ff9b::/96'
```

What we have now

- Dual-stack network: business as usual
- IPv6-only network:
  - works **normally** with Android (IPv4 goes via CLAT)
  - works **normally** with iOS/macOS (IPv4 goes via CLAT, ~~except for Safari et al~~)
  - works **barely** with other OSs (no CLAT, no PREF64 support, IPv4 is dropped)

IPv6-only

PSK: tutorial

IPv6-mostly

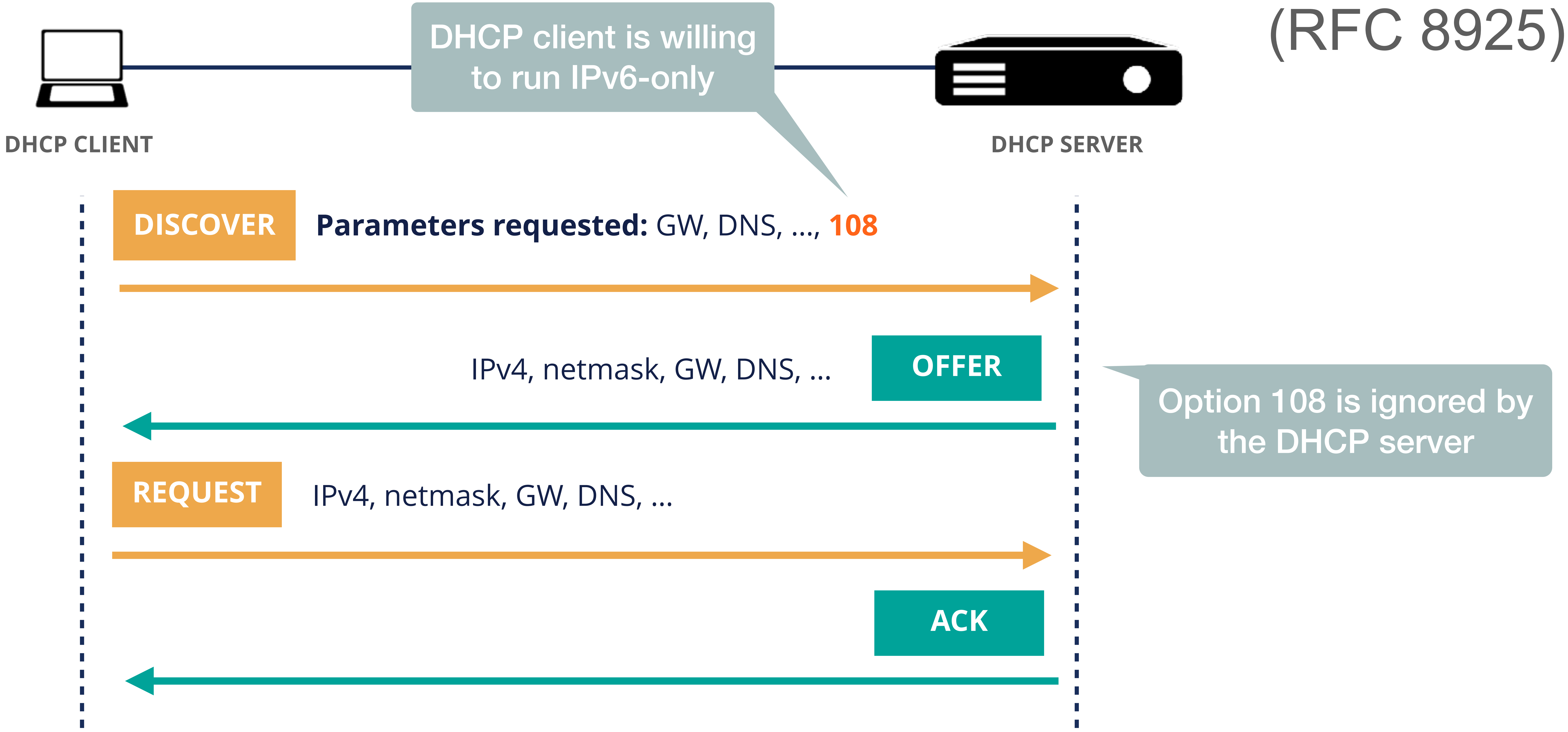
PSK: tutorial

IPv6-only-preferred

DHCP option to turn off IPv4

IPv6-only Preferred option of DHCP

(RFC 8925)



DHCP client is willing to run IPv6-only

Parameters requested: GW, DNS, …, 108

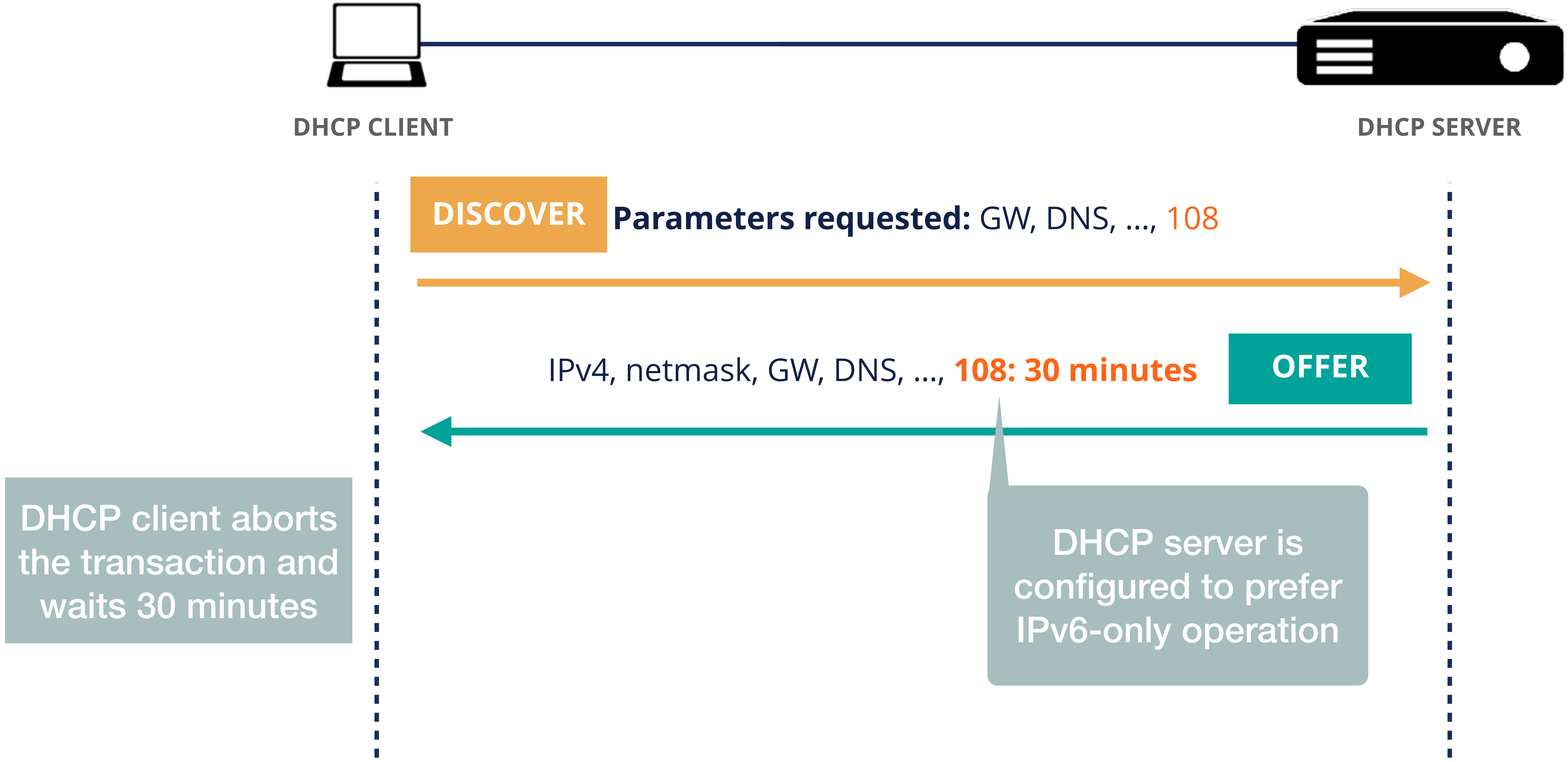
IPv4, netmask, GW, DNS, …

Option 108 is ignored by the DHCP server

IPv4, netmask, GW, DNS, …

Using DHCP to turn IPv4 off

(RFC 8925)



Parameters requested: GW, DNS, …, 108

IPv4, netmask, GW, DNS, …, **108: 30 minutes**

DHCP server is configured to prefer IPv6-only operation

DHCP client aborts the transaction and waits 30 minutes

Setting up DHCP Option 108

- No **special treatment needed** in the DHCP server
- We just need to encode the value ourselves
  - 30 minutes = 1800 seconds = 0x708 seconds

```
config dhcp 'lan'
    option interface 'lan'
    list dhcp_option '108,0:0:7:8'
    …
```

What we have now

- IPv6-only network: no change
- ~~Dual-stack~~ IPv6-mostly network:
  - **no change** for Windows, Linux
  - **same as IPv6-only** for Android, iOS and macOS

IPv6-only

PSK: tutorial

IPv6-mostly

PSK: tutorial

DNS64

Faking DNS with
good intentions

DNS64

- A easy trick to make legacy hosts use NAT64
- Native IPv6 is **unaffected**
- Queries for IPv4-only resources receive a **synthesised IPv6 answer** pointing to NAT64 space
- A legacy host thinks **every domain name has an IPv6 address**
- Works pretty well, but has some issues:
  - IPv4 literals
  - DNSSEC validation
  - Legacy IPv4-only socket API

Do we really need DNS64?

- Eventually, it will likely be **superseded by PREF64** and in-host translation
- Android can work well with **just NAT64/PREF64**
- ~~Native iOS/macOS apps require DNS64 to access IPv4 resources~~
- DNS64 makes legacy OSs use more NAT64 in place of native IPv4
  - good for IPv6-only network
  - **not so good** for an IPv6-mostly network, where legacy OSs run dual-stack

The easy option: Public DNS64

- Google Public DNS64
- Cloudflare Public DNS64
- Only if you use **Well-Known Prefix** for NAT64

```
config dhcp 'lan'
    option interface 'lan'
    list dns '2001:4860:4860::64'
    list dns '2606:4700:4700::64'
    …
```

Easy solution on TurrisOS

- TurrisOS uses **Knot DNS Resolver** by default
- Knot DNS Resolver has *decent* support for DNS64

```
modules = { 'dns64', 'view' }

-- Custom prefix example
-- dns64.config({ prefix = '64:ff9b:face:b00c::' })

-- Disable dns64 for IPv4 clients
view:addr('0.0.0.0/0', policy.all(policy.FLAGS('DNS64_DISABLE')))

-- Reenable it for a specific prefix:
view:addr(‘127.0.0.0/8’, policy.all(policy.FLAGS(nil, 'DNS64_DISABLE')))
```

Replacing dnsmasq with Unbound

- We need to turn off the DNS resolver function of dnsmasq while keeping the DHCP function
- Turning DNS off will stop offering DNS option in DHCP
- Some people find it really bad if DHCP hostnames do not appear in local DNS

```
config dnsmasq
    option port '0'
    …

config dhcp 'lan'
    list dhcp_option '6,0.0.0.0'
    …
```

```
config unbound ‘ub_main'
    …
    option dns64 '1'
    option dns64_prefix '64:ff9b::/96'
    …
    option validator '1'
    list iface_lan 'lan'
    list iface_lan 'lan6'
```

Unbound Recursive DNS server with UCI

What we have now

- IPv6-only network:
  - works **without issues** on Android, iOS and macOS
  - works with slight issues on other OSs
- ~~Dual-stack~~ IPv6-mostly network:
  - works exactly like IPv6-only network for Android, iOS and macOS
  - some IPv4 traffic goes via **DNS64** instead of native IPv4 for Windows, Linux

IPv6-only

PSK: tutorial

IPv6-mostly

PSK: tutorial

Using Ansible

Automating the setup

Pitfalls

- No Python in OpenWRT by default
- No native Ansible support for UCI configuration system

Both are resolved with role `gekmihesg.openwrt`

My roles collection

- openwrt-lan6
- openwrt-jool
- openwrt-pref64
- openwrt-dhcp108
- openwrt-unbound

Feel free to use and share:

https://github.com/oskar456/ansible-openwrt-ipv6-mostly

Questions ?

Ondrej.Caletka@ripe.net
@ripencc