Internet scale routing with NixOS

Yifei Sun

Inria, ENS de Lyon, Université Grenoble Alpes

NixCon 2025 · September 6, 2025

Why?

(O_O)

Routing daemon

Software that implements **routing protocols** to **exchange routing information** with other routers and **maintain routing tables**.

BIRD: BIRD Internet Routing Daemon

Problem

NixOS module option `services.bird.config` is text only

- Maintenance burden
  - Hard to read
  - Hard to write
  - Hard to debug
- Hard to compose and reuse

Solution

Parameterizing BIRD configuration with NixOS options

Prerequisites

- Nix and NixOS

- Some netwoking knowledge
  - systemd-networkd
  - nftables[1]

- BIRD
- Tailscale[2]

[1] Linux Kernel packet classification framework
[2] Open source mesh VPN software

How the internet works

Resource acquisition

Routing security

Finding an upstream

Pingable!

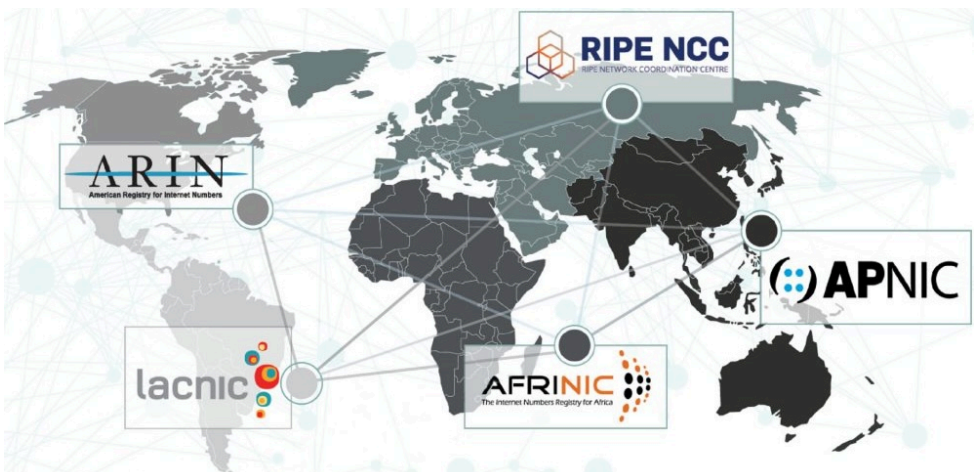
Acquisition[3]

ASN:

- RIR[4] direct assignment or LIR sponsorship

IP address:

- Direct assignment
- Lease/purchase from third-party



[3] Or dn42, ...

[4] nro.net/about/rirs

Routing security

RPKI: signed authorization objects hosted by registries

ROA

- Which AS can announce the prefix under some max length
- Only registries can host ROAs
- X.509 cert signed objects

IRR: database of routing policies, hosted by registries and other entities

IRR objects

- Mostly who can announce the prefix, who is customer, who is provider, etc.
- Queryable
- Aside from registries, known entities can also host IRR (NTT, RADB, etc.)

Add objects to for your ASN & prefix

ROA

ROUTE/ROUTE6, AS-SET, etc.

Getting connected

- Physical presence in datacenter
  - Hurricane Electric
  - Cogent
  - Equinix
  - …
- Virtual presence at cloud provider that provide IP transit
  - Vultr, V.PS, Neptune Networks, etc.
  - https://bgp.services
- Any VPS + virtual IX or transit providers
  - Cloudflare Magic Transit
  - https://bgp.exchange
  - https://route64.org

Going live

- Setting up BGP session
  - Get routes from upstream
    - Default route (0.0.0.0/0, ::/0)
    - Full table (usually not necessary)
      - BIRD: 250MB+ for ~1M IPv4 routes + ~230K IPv6 routes)
      - May be smaller or larger depending on routing daemon
- Routing policies
  - Import/export
  - Filtering
  - …
- Add address within announced prefix to interface

Parameterizing BIRD configuration

RPKI and filters

Export routes to kernel

Static routes and BGP sessions

Internal routing with Tailscale + bonus ;)

RPKI and filters

RPKI

- BIRD `rpki` protocol and `roa` table
- Defining `options.router.rpki`
  - v4/v6 table and filter names
  - retry, refresh, expiration times
  - list of validators

Pre-defined filters

- Renamable ROA filters based on route RPIKI status
- Can be referred later in import/export filters
- Will support arbitrary filters in the future

```
router.rpki.validators = [{
  id = 0;
  remote = "rtr.rpki.cloudflare.com";
  port = 8282;
}];
```

RPKI and filters

```
services.bird.config = lib.mkOrder <int> ''
  ${lib.concatMapStringsSep
  "\n\n"
  (validator: ''
    protocol rpki rpki${toString validator.id} {
      remote "${validator.remote}" port ${toString validator.port};

      retry keep ${toString cfg.router.rpki.retry};
      refresh keep ${toString cfg.router.rpki.refresh};
      expire ${toString cfg.router.rpki.expire};
    }'')
  cfg.router.rpki.validators}
'';
```

RPKI and filters

```
roa6 table trpki6;

protocol rpki rpki0 {
  roa6 { table trpki6; };
  remote "rtr.rpki.cloudflare.com" port 8282;
  retry ...; refresh ...; expire ...;
}

protocol rpki rpki1 {
  roa6 { table trpki6; };
  remote "r3k.zrh2.v.rpki.daknob.net" port 3323;
  retry ...; refresh ...; expire ...;
}

filter validated6 {
  if (roa_check(trpki6, net, bgp_path.last) = ROA_INVALID) then {
    print "Ignore RPKI invalid ", net, " for ASN ", bgp_path.last;
    reject;
  }
  accept;
}
```

Kernel protocol

Export routes to kernel

- Default route: most common
- Full table: 250MB+ from BIRD, and another copy in kernel
- Manual: don't export routes but set `networking.*` options

```
options.router.kernel = {
  ipv4 = {
    import = lib.mkOption { ... };
    export = lib.mkOption { ... };
  };
  ipv6 = {
    import = lib.mkOption { ... };
    export = lib.mkOption { ... };
  };
};
```

Static routes

Manually configured routes

```
options.router.static.ipv4.routes = lib.mkOption {
  type = lib.types.listOf (lib.types.submodule {
    options = {
      prefix = ...;
      option = ...;
    };
  });
};
```

```
router.static = {
  ipv4.routes = [
    { prefix = "0.0.0.0/0";
      option = "via 198.51.100.1"; }
    { prefix = "203.0.113.0/24";
      option = "blackhole"; }
  ];
  ipv6.routes = [{ prefix = "2001:db8::/32";
                   option = "reject"; }];
};

# config.services.bird.config
lib.concatMapStringsSep
  "\n  "
  (r: "route ${r.prefix} ${r.option};")
  cfg.router.static.ipv4.routes
```

BPG sessions

- The usual: 1 session per protocol
- MP-BGP: 1 session for both v4 and v6

```
options.router.sessions = lib.mkOption {
  type = lib.types.listOf (lib.types.submodule {
    options = {
      name = ...;
      password = ...; # null for don't validate
      type = ...; # disable, direct, multihop
      mp = ...; # null, v4 over v6, v6 over v4
      neighbor = ...; # ASN, IPv4, IPv6
      import = ...; # IPv4/IPv6 import filter
      export = ...; # IPv4/IPv6 export filter
      ...
    };
  });
};
```

```
# example
router.sessions = [{
  name = "somebody";
  password = null;
  type.ipv4 = "disabled";
  type.ipv6 = "multihop";
  mp = "v4 over v6";
  neighbor = {
    asn = 65536;
    ipv4 = null;
    ipv6 = "2001:db8::1";
  };
  import.ipv4 = "import none;";
  import.ipv6 = "import none;";
  export.ipv4 = "export all;";
  export.ipv6 = "export all;";
}];
```

Adding announced prefixes to interfaces

- Enable forwarding
- Use `dummy` interface (or use the main interface or whatever)
- Disable `ManageForeignRoutes` in systemd-networkd (will delete routes exported by BIRD)

```
boot.kernelModules = [ "dummy" ];

boot.kernel.sysctl = {
  "net.ipv4.conf.default.forwarding" = 1;
  "net.ipv6.conf.default.forwarding" = 1;
};

systemd.network.config = {
  networkConfig.ManageForeignRoutes = false;
};
```

Adding announced prefixes to interfaces

- Use `systemd.network.netdevs` to configure virtual interfaces
- Use `systemd.network.networks` to configure addresses and routing policies
  - `routingPolicyRules` is only needed when the outbound gateway of the announced prefixes is different from the default gateway of the main interface

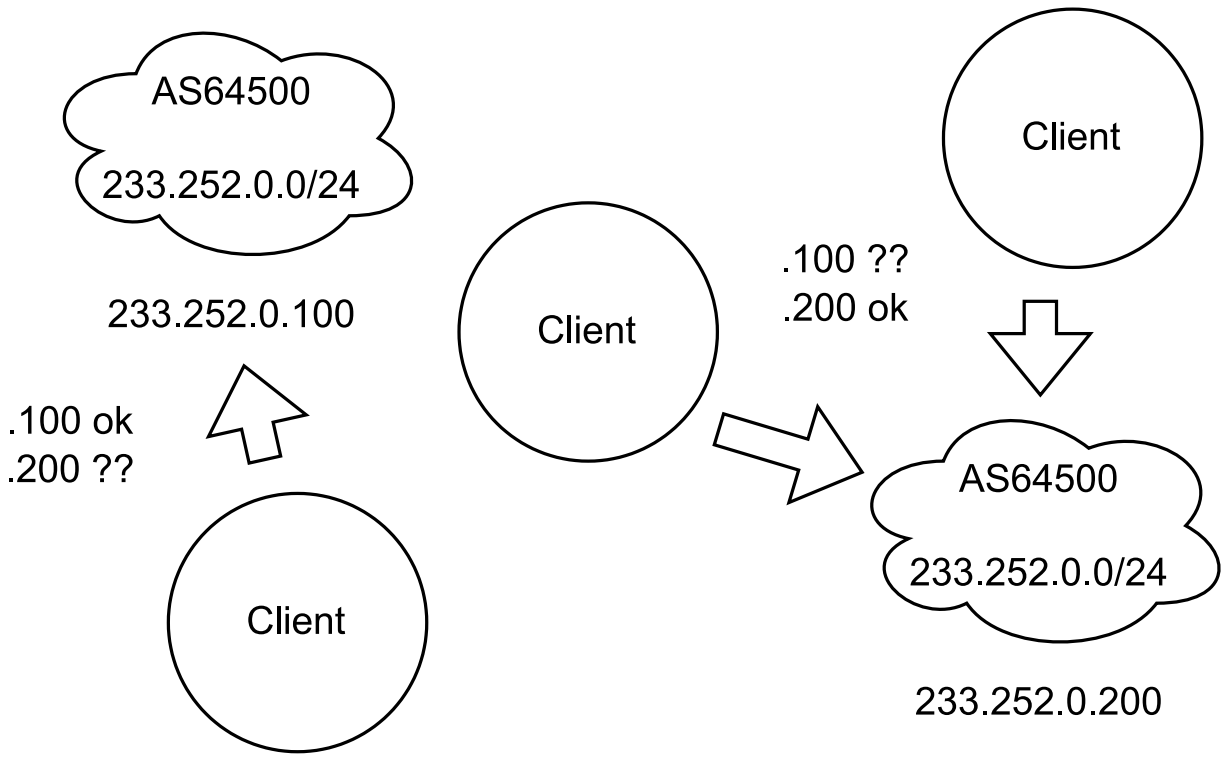
```
systemd.network.netdevs = {
    "40-dummy0".netdevConfig = {
    Kind = "dummy";
    Name = "dummy0";
  };
};

systemd.network.networks = {
    "40-dummy0" = {
    name = "dummy0";
    address = ipv4.addresses ++ ipv6.addresses;
    routingPolicyRules = ...;
  };
};
```

Immediate benefit

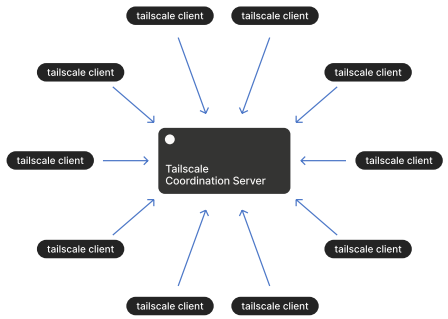
Parametericity

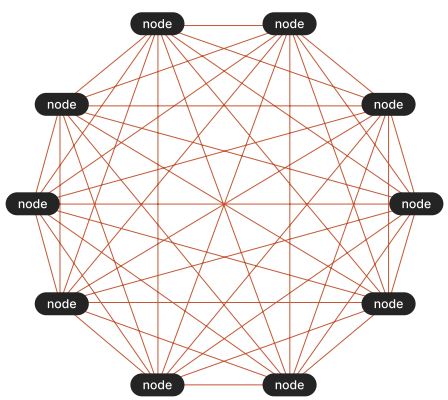
```
config = lib.mkIf
  (options?router
  &&
  lib.elem
    config.a.b.c.bind.v4
    config.services.router.local.ipv4.addresses
  &&
  lib.elem
    config.a.b.c.bind.v6
    config.services.router.local.ipv6.addresses)
  { ... };
```

Multiple upstreams?



Internal routing with Tailscale[5]



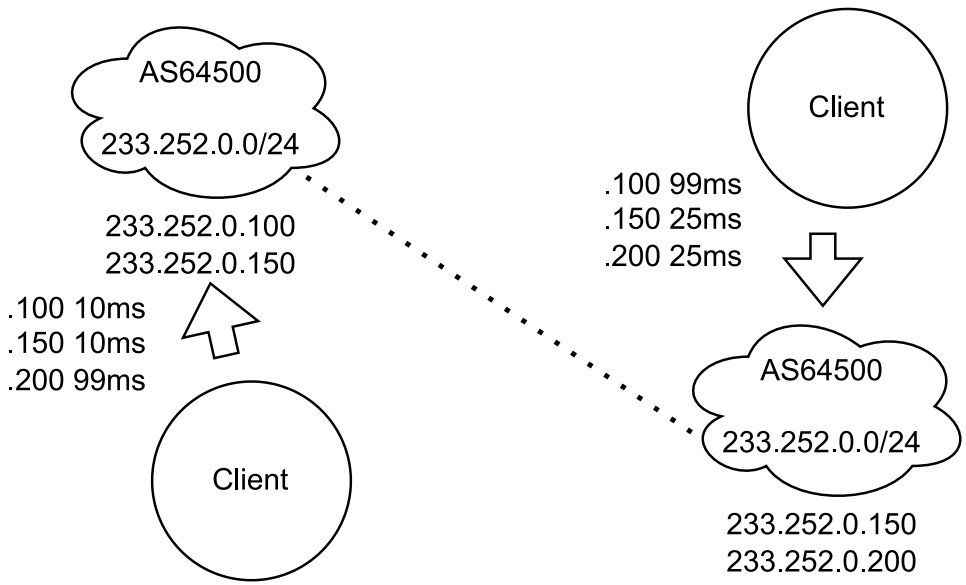



```
services.tailscale.extraSetFlags =
let
  v4s = lib.concatStringsSep "," ipv4.addresses;
  v6s = lib.concatStringsSep "," ipv6.addresses;
  addresses =
    if v4s == "" then v6s
    else if v6s == "" then v4s
    else v4s + "," + v6s;
in
[ "--accept-routes"
  "--advertise-routes=${addresses}"
  "--snat-subnet-routes=false" ];
```

*Other tunneling protocols will also work

[5] tailscale.com/blog/how-tailscale-works

Anycast



Announcing the same prefixes, and use same address(es) on multiple machines. This is what Cloudflare, GitHub Pages, and many other providers do.

Lessons learned

- Use IPv6 when possible
- Host a looking glass
  - `bird-lg` or other looking glass tools
  - Feed full table to BGP.Tools, NLNOG, etc.
- `tcpdump`, `mtr`, `ping`, `traceroute`, `dig`, etc.[6] are your friends
- Tailscale is good but…
  - Eat the entirety of CGNAT address space (100.64.0.0/10)
  - `nodeAttrs` -> `ipPool` is half-baked, outstanding issue since Feb 2021 (tailscale#1381)
  - My workaround: set lookup rules with very specific priorities

[6] Also ping.sx is pretty cool

Closing remarks

- Possible to run your own validator (ROA based and IRR based)
- Plan to write a more generic module to replace `services.bird` and upstream it
  - Proof of concept: `github:stepbrobd/router#nixosModules.alpha`
  - Idea from: `github:NuschtOS/bird.nix`
  - Support all protocols
  - Integrate flow exporter for better observability

Questions?

ysun@hey.com · `github:stepbrobd/router`

Special thanks to Nick Cao (`github:NickCao`)

CLOUDFLARE Project Alexandria