IPv6-mostly Field Report from RubyKaigi

Sorah Fukumori https://sorah.jp/
@ Internet Week 2025

- ✦ **Sorah Fukumori** (そらは) https://sorah.jp/
  - ✦ RubyKaigi Organizer (2018-)
  - ✦ RubyKaigi NOC Lead (2017-)
  - ✦ AS59128 京大マイコンクラブ (2017-)
- ✦ 本業: **Principal Engineer for IVRy (2025/4-)**
- ✦ 趣味: ストリーマー/**VTuber**おたく, オンゲキ, **FFXIV**

- ✦ 本スライドは以下で公開してます (写真撮る必要ないです)
- ✦ 本プログラム内でもう話されていて重複する内容はスキップしていきます
- ✦ 結構な速度で話すので、→ おすすめです

https://speakerdeck.com/sorah/iw2025-rubykaigi-v6mostly

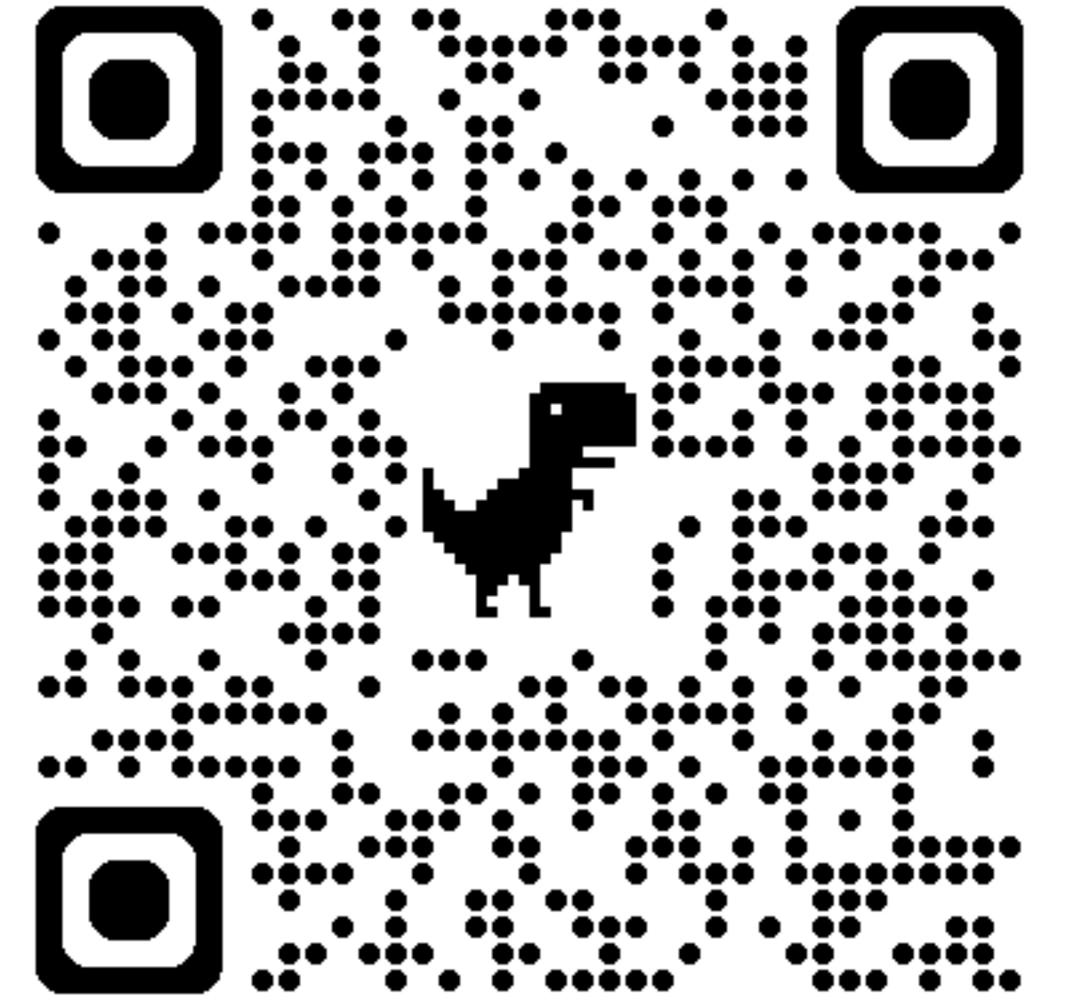
- RubyKaigi は、プログラミング言語 Ruby に関する世界最大級の国際カンファレンスです。 最先端の技術セッションの数々が披露されるプログラマーズカンファレンスとして、また Ruby の処理系自体の開発者を集めて議論を促進するエンジンとして、さらに、英語・日本語を公用語とした世界的な Ruby コミュニティの交流のハブとしても、大きな役割を担っています。
- **日本各地を転々**と年1開催する国際会議
  - 1,200-1,500 人規模, Assocs は人数 * 1.1 くらい, 200-400Mbps
  - https://rubykaigi.org/2025/

- RubyKaigi NOC では Kaigi on Rails の Wi-Fi 提供も実施
  - 同じ法人格を共有するが主催・運営方針は異なる、
    年1、東京で開催される国内の技術カンファレンス
  - Kaigi on Rails は 「初学者から上級者までが楽しめるWeb系の技術カンファレンス」をコンセプトに、技術カンファレンス参加の敷居を低くすることを意図して企画されています。一方、それと同時にベテランの技術者も満足させるような高度な発表も提供していくことで、上記のコンセプトを現実化させていきたいと考えています。
- https://kaigionrails.org/

RubyKaigi NOC

- ✦ **少数精鋭、省力。**固定メンバー + 募集で毎年ちょっとずつ入れ替わり
- ✦ その中のさらに少数が L3〜L4 以上を触っている
  - ✦ 最近は最新の RFC を実装したり
    (DNS over TLS/HTTPS/QUIC etc)
- ✦ Transit/PA を AS59128 KMC から渡しているので KMC メンバー多め

Openness



- 成果物はだいたい全て公開
  - https://github.com/ruby-no-kai/rubykaigi-net (rknet)
  - Issue や Grafana Dashboard もパブリック
- 今回の話に関連する諸々もあるはず
- その他ブログ等の write-up も README からリンクされています

Agenda

- ✦ IPv6-mostly とは何か
- ✦ IPv6 Transition Technologies Recap
- ✦ IPv6-mostly のモチベーション
- ✦ Actual deployment and issues at RubyKaigi

本プログラム中の他のセッションと重なる部分もありますが、
企業内ネットワークやイベントネットワーク管理者のコンテキストでお伝えします

IPv6-mostly とは何か?

- IPv6-mostly とは?
  - Native IPv4 は必要な端末にだけ提供するネットワーク
  - IPv6-mostly 対応の端末は IPv4 を NAT64 越しに使う
- 特徴:
  - **同じ L2 ドメインで共存・コントロールしつつ移行できる**
  - 不要な端末の DHCPv4 のアドレス割当を省略できる
  - IPv4接続は端末が直接IPv6移行技術を使いサポート

昔からある物を含め複数の RFC を利用して実現しています。 draft-ietf-v6ops-6mops-04 が “IPv6-mostly” の規定になる予定

- 具体的には端末が IPv4 Lease をオプトアウト可能で、かつ NAT64 があり、それに必要な情報を取得できれば良い
- DHCPv4 Option 108 とPref64::/n のディスカバリーや通知
  - RFC 8925 (Option 108) — IPv4 Lease なくてもいいよの意思表示
  - RFC 8781 (RA PREF64) — IPv6 RA で NAT64 prefix を通知
  - RFC 7050 (DNS-based Discovery) — RFC 9872 で非推奨*に

* normative な NOT RECOMMENDED ではなく、RFC 8781 が使えるならその方が良いと問題点付きで議論されています

- ✦ **NAT64 を利用するが DNS64 ではない点に注意**
- ✦ DNS64 は問題点が多い
  - ✦ 提供されているNAT64に合わせたDNSリゾルバを利用する必要(Public DNSとか各種VPNで破綻)
  - ✦ DNSSEC 非互換
  - ✦ IPv4を前提としたAPIに依存したソフトウェアとの互換性(AF_INETとかIPアドレス直接続とか)

IPv6 Transition Technologies
おさらい

- 話を進める前に IPv6 移行技術 (IPv4aaS) のおさらい
  - ※ 6rd のように IPv4 ネットワークで IPv6 を提供する技術ではなく、DS-Lite や MAP-E といった IPv6 ネットワークで IPv4 接続性を提供する手段の方
- 本プログラム1つ目のセッションでだいたい触れられているのでサッと

- ✦ まずパケットの取り扱いで 2 方式
- ✦ **4in6 encapsulation** ※所謂トンネル

  IPv4パケットをIPv6パケットとしてカプセル化
- ✦ **4-6-4 translation**

  IPv4ヘッダをIPv6ヘッダに変換してIPv4ヘッダに戻す

あわせて読みたい: **RFC 9313** Pros and Cons of IPv6 Transition Technologies for IPv4-as-a-Service (IPv4aaS)

- ✦ つぎに IPv4 NAT の扱いで 2 方式
- ✦ **(Carrier Grade) NAT**

  IPv4aaS側設備で IPv4 NAPT を行う

- ✦ **A+P (Address plus Port)**

  IPv4aaSからCustomer Edge側に予めIPv4アドレスとポート範囲を割当, CE側でNAPTしておいてもらう

*必ずしも Carrier Grade である必要はありません

✦ 実際の移行技術にあてはめると、こう:

	CGNAT	A+P
4in6	DS-Lite	MAP-E
4-6-4	464XLAT DNS64	MAP-T

* DNS64は4-6-4というか6-4だけど便宜的に4-6-4のところに書いてます

✦ 4-6-4 translation ではどちらも NAT64 を利用

	CGNAT	A+P
4in6	DS-Lite	MAP-E
4-6-4	**464XLAT** **DNS64**	**MAP-T**

* MAP-T は CE で NAPT44 をするため PE は Stateless NAT64

これまでの技術はわりと ISP/Carrier 向け

- DS-Lite は単なるトンネルではなく **IPv6** ヘッダの **IPv6 source address も含めた** NAPT テーブルを持つ必要がある
  - 単にdecapしてNAPT処理をすればいいわけではなく、NAPTとdecapを同時にできる設備が必要
- A+P方式もMAP-Eなどそれぞれに対応した設備が必要

- NAT64, 4-6-4 translation はその分じつは手軽である
  - もちろん大規模になると NAPT 設備のスケール問題は発生する
  - DNS64 は厳しかったが、それ以外で活かす方法が少なかった
- ネットワーク設備としては、下記があれば良い
  - Customer 側: NAT64 prefix (Pref64::/n) への経路
  - NAT64 設備: Customer への IPv6 経路と IPv4 接続性
  - 上記の間の Backbone は純粋な IPv6 ネットワーク

NAT64 は手軽

- NAT64, 4-6-4 translation はその分じつは手軽である
  - もちろん大規模になると NAPT 設備のスケール問題は発生する
  - **DNS64 は厳しかった**が、それ以外で活かす方法が少なかった
- ネットワーク設備としては、下記があれば良い
  - Customer 側: NAT64 prefix (Pref64::/n) への経路
  - NAT64 設備: Customer への IPv6 経路と IPv4 接続性
  - 上記の間の Backbone は純粋な IPv6 ネットワーク

- **RFC 6877 (464XLAT)**
- 端末で “CLAT” として IPv4 <> IPv6 ヘッダの translation (RFC 7915 aka SIIT) をして提供されている NAT64 prefix を利用する方法
  - 端末の中で local 192.0.0.2, first hop 192.0.0.1 (例) のようなネットワークが設定されるため、IPv4 の API とも互換性がある
- “PLAT” であるネットワークにある Stateful NAT64 機能は RFC 6146 そのまま
- 2013-2014 頃から US T-Mobile で運用されている枯れた手法

- 464XLAT は 2013-2014 頃から US T-Mobile で運用されている, 枯れた手法 (というか T-Mobile 社員発案(のはず)
  - Android 4.3〜, iOS 12 頃〜
- 端末の CLAT は事前に設定された mobile network や **IPv6 only** で NAT64(DNS64) がある Wi-Fi/Wired network に限っての起動だった
  - Wi-Fi 等でも有効になるようになったのは iOS 17 頃, Android 5.1 のよう

Recap

- ✦ IPv4aaS のうち DS-Lite, MAP-E は ISP 向けだし、CPE もネットワーク機器への実装が中心
- ✦ NAT64 は設備側がステートフルだが実はお手軽
- ✦ DNS64 は厳しかったが 464XLAT が battle-tested
- ✦ Apple 端末や Android では 464XLAT の端末側機能 CLAT が NAT64 がある IPv6-only network では有効になっている

Re: IPv6-mostly とは何か?

- IPv4v6 dualstack, **IPv6 (native) + IPv4 (NAT64),** IPv4 only を端末によって使い分けできるネットワークのこと
- DHCPv4 Option 108 (IPv6-only preferred) が鍵
    - サーバーはこれを受信したら **IPv4 lease** を省略して良い
      (Option 108 を送り返す必要はある)
    - 端末は 108 が送り返されてきたら Native IPv4 をオフにして、
      NAT64 があれば CLAT を起動する

* 仕様上は IPv4 完全に不要かもしれないし、現実的には NAT64 (464XLAT) だけど DNS64 かもしれないしそれ以外の移行方法かもしれない。ここは DHCPv4 オプションの RFC としては定めていない

- IPv4v6 dualstack, **IPv6 (native) + IPv4 (NAT64),** IPv4 only を端末によって使い分けできるネットワークのこと
- DHCPv4 Option 108 (IPv6-only preferred) が鍵
  - サーバーはこれを受信したら **IPv4 lease を省略して良い**
    (Option 108 を送り返す必要はある)
  - 端末は 108 が送り返されてきたら Native IPv4 をオフにして、NAT64 があれば CLAT を起動する

* 仕様上は IPv4 完全に不要かもしれないし、現実的には NAT64 (464XLAT) だけど DNS64 かもしれないしそれ以外の移行方法かもしれない。ここは DHCPv4 オプションの RFC としては定めていない

DHCPv4 Option 108 (IPv6-only preferred) が鍵

- ✦ DHCPv4 設備は受信した Option 108 に対して処理を決められる。例えば:
  - ✦ 一定の割合で Option 108 を返す
  - ✦ 端末によって Option 108 を返す
  - ✦ Option 108 を一部の端末では返さない
- ✦ **つまり、サーバー側から柔軟に IPv6 + IPv4aaS への移行を操作できる**

段階的移行のメリット

- ユーザーにネットワークを選択させたり、頑張って **Dynamic VLAN** 等で **IPv6-only network** へ誘導するのは現実的ではない
  - ユーザーはわざわざ選択しないし、ダメなら黙って戻ってしまう
- **Dualstack** な同一の **L2** ドメイン **1** つで **DHCPv4** オプションで **IPv4 usage** を制御できるのは強み
  - 段階的に **IPv4** プールや **NAPT44** 設備をスケールダウンできる
  - ユーザーに接続先を意識させたりしなくて良い

- RFC 8781 (RA PREF64) によって ICMPv6 Router Advertisement に Pref64::/n 情報を含めることが可能になった
- これまでの RFC 7050 (DNS-based discovery) はセキュリティや性能面、ネットワークに対応したリゾルバに依存するといった問題があった (RFC 9872 で議論)
- RFC 8781 によって RFC 7050 は不要に

なお、RFC 7050 は DNS64 をフル実装する必要はなく ipv4only.arpa の AAAA レコードだけ実装すれば良い

Recap

- ✦ IPv6-mostly は既存の dualstack な L2 ドメインで端末を IPv6 native + NAT64 に寄せるための仕組み
- ✦ 礎の 464XLAT はモバイルネットワークで大規模な運用実績
- ✦ DHCPv4 オプションを利用した柔軟な制御
- ✦ Windows 以外のサポートは順調
    - ✦ Windows 11 も Private Preview の告知が出たところ

IPv6-mostly のモチベーション

- 業界内でもプレイヤーの立場によってモチベーションが異なる
  (ISP, ネットワーク機器ベンダー, OSベンダー, 企業内の管理者, etc)
- 社内ネットワークやイベントネットワークの管理者としての私見

- (特に国内では) MAP-E や DS-Lite が十分枯れた今、選択肢として追加する強いモチベーションはないかも? 想像……。
- CPEがOption 108とRA PREF64に対応するモチベーションは?
  - 消費者向けに提供するには技術サポートコストが増すだけかも? LAN 内の IPv4 通信も制約がかかる
  - ※モバイルネットワークでは既に大規模な実績がある
- ISP視点だとDS-Lite同様ステートフルNAPTの運用コストもかかる

アプリケーション開発者視点

- モチベーションというか **464XLAT** の利点
- **NAT64+DNS64** と違い **CLAT** だと **IPv4** 依存の **API** が使え互換性が高く、アプリケーション開発者は意識する必要がないのが大きい
  - **Apple App Store** は **NAT64** 利用しても良いので **IPv6-only** 環境での動作を審査要件に含めているが、**CLAT** 前提だと負担がだいぶ減るはず
  - **Wi-Fi** や **Wired** の **LAN** でも **464XLAT** に出来るのは良いこと

- ✦ IPv4 singlestack → IPv4/IPv6 dualstack になったとしても, IPv4 への依存がなかなか無くならない
  - ✦ IPv6 singlestack へちょっとずつ持っていくための仕組みが必要

- ここまで解説してきたように、**NAT64** のアクセスネットワークでの柔軟な有効化という点で、こちらが主な用途かな**?** と思っている
    - **CPE** ではなく端末から直接つながる特徴が活きる
- 自社内で管理する **IPv4** プールや設備の規模をちょっとずつ縮小できるのは強み
    - 経路表を増やさずに **Pref64::/n** の使い分けで **IPv4** 出口を使い分けるのも可能
    - ネットワーク内の**1**つのオーバーレイ機能として**IPv4**を捉えられる

- RubyKaigi に限らず, 各種NOGやRIR、またIETF等標準化団体のイベントで採用が進んでいる
- これはどちらかと言えばテンポラリなネットワークだから新しい仕様を試せるとか、新しい仕様を試すのは楽しいという側面がある
- 同様に IPv4 運用を縮小できたら嬉しいという側面も否定はしない

- ✦ IPv4 only, IPv4v6 dualstack に加えて IPv6 only + NAT64 という接続形態が出来る以上、短期的な運用負担は低くはない
  - ✦ (後で触れるが) Apple device と Android が majority な環境であればNAPT44 設備の需要は即座に低下するため、それに気を配る必要がなくなるという点では負担は変わらないかも
  - ✦ NAT64 設備の運用経験が浅いとかそういう観点の短期的負担はあるかもしれません

- 経済面だと、NAT44 と NAT64 で設備を分離する場合は IPv4 アドレスコストは増加する
  - NAPT44 プール需要が移動するはずなのでアドレスを NAT64 に移すことはできるだろうけど
- NAT64 設備がポピュラーではないという難点もある
- 一方、DNS64 設備を捨てることは出来るようになった

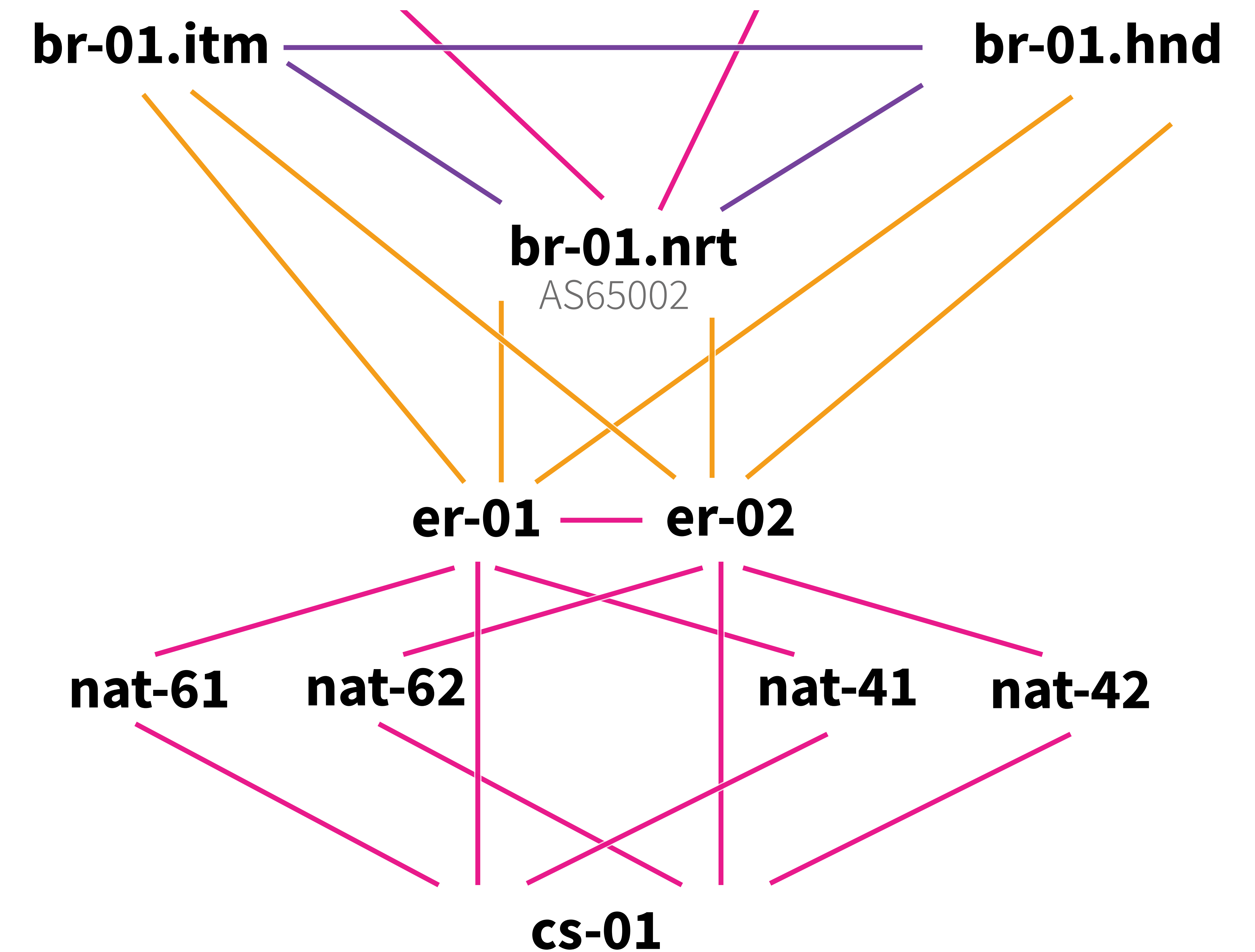
- IPv6-mostly を only にしていく IPv4 環境の縮退は課題になるかも
  - Backbone を v6 only にしたくても DHCPv4 Relay してるから… みたいな問題は残りそう

RubyKaigi 2025 の IPv6-mostly

- おしながき
    - NAT64
    - Pref64::/n
    - DHCPv4 Option 108
- トラブル集

NAT64

- 既存の battle-tested な NAPT44 設備の横に NAT64 設備を追加
  - 既存の機材が NAT64 非サポート
  - トラブル時に実績ある確実な setup に fallback するため
- コアL3スイッチ → NAT{44,64} → 会場エッジルーター



- NAT64 機材は Linux が入ったミニ PC (N100)
- Linux mainline には Stateful NAT64 実装がない (!)
  - Jool など外部の kernel module + userspace 実装がポピュラー
- 採用した構成は Linux netfilter + 自前実装 + bird2
  - NAPT66: Linux netfilter (nftables)
  - NAT64 (SIIT): https://github.com/sorah/xlat
  - Routing (BGP): bird2

- RubyKaigi なので? Ruby で実装できるのでは
  - kazuho/rat (Ruby userspace の NAPT44 実装) という先行事例
- できた → https://github.com/sorah/xlat

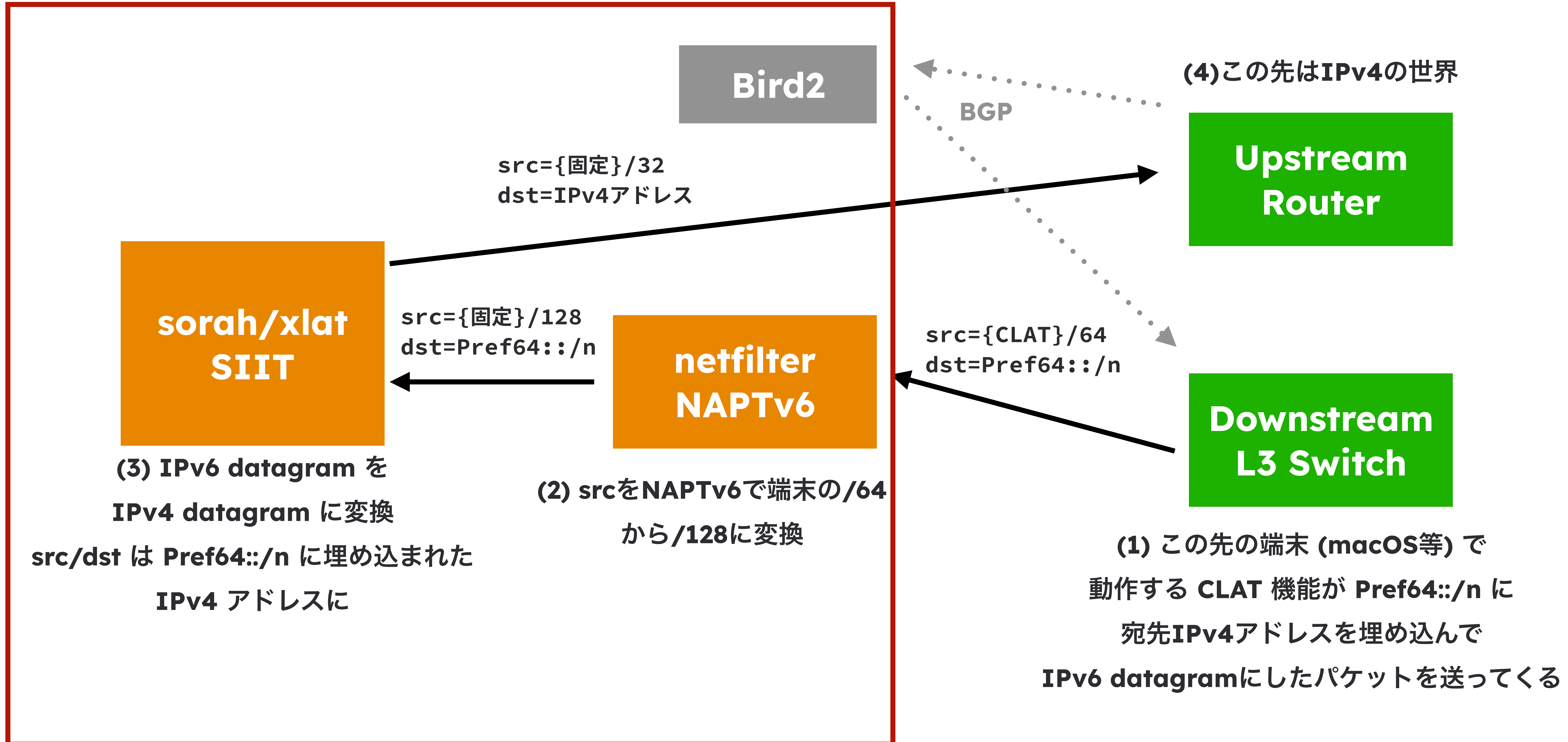
- Acknowledgement: @hanazuki が高速化をやってくれました

  https://blog.kmc.gr.jp/entry/2025/05/13/130055
  RubyKaigi 2025で使用したSIITのRuby実装について

- ✦ Linux mainline に実装がない、ステートレスな IPv4/v6 ヘッダと ICMPv4/v6 パケット変換 (SIIT*) だけ実装
- ✦ NAPT は Linux netfilter の NAPT66 実装を利用
  - ✦ Netfilter の NAPT44 は過去に RubyKaigi で運用し実績がある
  - ✦ kernel で IPv6 addresses → IPv6 address の N:1 な NAPT SIIT は IPv6 address ←→ IPv4 address の NAT をするだけ
- ✦ = NAPT66 + Stateless NAT64 の合成で Stateful NAT64 に

* RFC 7915: Stateless IP/ICMP Translation, RFC 6052: IPv6 Addressing of IPv4/IPv6 Translators

ユーザ側からインターネットへの経路



- 経路制御は BGP
  - Pref64::/n への経路を downstream (client側) に IPv4 outer の /32 を upstream (internet側) に広報している
  - 受け取るのは downstream から配下の経路, upstream から 0/0

経路制御

- Downstream の経路表では ECMP で冗長 + ロードバランス
  - Hash key を IPv6 saddr に絞って stickiness を実装
  - 障害/復旧時は経路切替で既存の NAPT セッションは全滅する
- Upstream の経路表は IPv4 outer address /32
  - ふつう

経路制御

- BIRD で Pref64::/n 宛の static route を recursive next hop, v4-via-v6 で定義している → upstream からの v4 default route が途切れたら downstream への経路広報を停止する
- Kernel FIBには install しない BGP 広報用の経路データ

```
protocol static static_bgp6 {
  …
  igp table bgp4;
  route 2001:df0:8500:ca64::/64 recursive 192.0.0.0 {
    bgp_community.add((C_SELF,C_CTL_ALLOW_INSIDE));
    bgp_community.add((C_SELF,C_CTL_PREVENT_KERNEL));
  };
}
```

https://github.com/ruby-no-kai/rubykaigi-net/blob/master/itamae/roles/plat/templates/etc/bird/bird.conf.d/plat.conf

Pref64::/n

- 464XLAT (RFC 6146) では RFC 7915 (SIIT) で利用する v4/v6 アドレス変換アルゴリズムは RFC 6052 を利用すると定義
- RFC 6052 § 3.1. で IPv4 private address で well-known prefix の利用が禁止されている*ため、自分で prefix を選ぶ必要がある
- また § 4.1. に選び方の1つとして checksum neutral の紹介がある
  - Pref64::/n の 1 の補数和が $ffff_{16}$ ないし 0000_{16} にしておく
    → アドレス変換時にIPヘッダのチェックサムの再計算が不要に

* この制限を緩和する I-D draft-ietf-v6ops-nat64-wkp-1918 がちょうど今日 WG adopt されたよう

- **RA Option (RFC 8781) は Junos 22.4R1 から設定可能に** (ラッキー)

```
set protocols router-advertisement interface … nat-prefix …
```

- RFC 7050 (ipv4only.arpa AAAA レコード) も実装していたが、本番ではほぼ RA オプションを見る端末が占めていた(はず)
- DHCPv4サーバーと比較してルータ機器でサポートされない場合はデプロイが困難な状況なのは辛い
  - 設定で任意のオプションとデータ列を追加できて欲しい… RDNSSの事を思い出す

DHCP

- RubyKaigi では長年 ISC Kea を DHCPv4 サーバーとして利用
- DHCPv4 Option 108 (RFC 8985: IPv6-only Preferred) は Kea 2.7.1 からサポートが入った
    - 返した場合に DHCP リースを省略する
    - それ以前のバージョンでも任意のオプションを追加できるため リースの省略はできないが送信自体は可能

- **“Legacy” SSID では Option 108 をオプトアウトしたい**
- DHCP Relay がつける Option 82 で switchport や SSID を取れる
- ただ、Subnet で定義した option-data を client class で上書き出来ない。そのため設定はオプトイン方式で行う

```
{
  name: 'main_ssid',
  test:     "(split(relay4[2].hex, ':', 2) == 'RubyKaigi 2025')" + // SSID
          " or (relay4[1].hex == 'cs-01-venue:ge-0/0/0.0:usr')",     // switchport
  'only-if-required': true,
  'option-data': [{ name: 'v6-only-preferred', data: '1800' }],
}
```

https://github.com/ruby-no-kai/rubykaigi-net/blob/master/k8s/dhcp/config/class_main_ssid.libsonnet

RubyKaigi 2025
Experiment Results

- 2025/4 の RubyKaigi 2025 で試したところ、いくつかトラブルがあったので事例を紹介
    - Majority が Apple デバイスのため iOS, macOS 関連の問題
- 2025/9 の Kaigi on Rails 2025 では Apple 側の修正等により、認識している問題の殆どは解決
    - macOS 15.5, iOS 15.5 (5月) 以降か macOS 26, iOS 26 なら OK
    - https://blog.sorah.jp/2025/11/04/mystery-of-clat-on-apple-devices にもまとまっています

Unicast Neighbor Solicitation への応答がない

- 本番の際、Apple device は接続して 30 分程度で IPv4 Internet 接続性を失うことが分かった
  - CLATは起動しているがパケットが帰ってこない
- 本番中は分からず、後日調査。開発時は WLC を通していなかったため、WLC 予備機を利用して環境をさらに近付けて追試
- 調査の結果 First hop router で CLAT が持つ IPv6 アドレスの Neighbor Discovery に失敗していることが分かった

- Wi-Fi 環境では AP, Controller で ARP/ND の代理応答がありがち
  - Multicast パケットを減らして Wi-Fi エアタイムを削減するため
- WLC の近隣テーブルで reachable な間は WLC が unicast NS に代返しているが、タイマが切れると unicast NS に答えなくなる

これは 2 つの Workaround が考えられる. RubyKaigi では #1 を採用

1. **近隣テーブルの Lifetime を延長する:**
   最大のベクライアント数や滞在時間、機器の上限を考えながら設定
   RubyKaigi では 18 時間に

2. **Multicast NS パケットを Wi-Fi クライアントへ転送する:**
   Wi-Fi のエアタイムを消費するのでおすすめしない
   また、IPv6 で stateful firewall がないため、インターネットから
   届く様々なパケットによって NS パケットが生成されてしまう問題も

DNS で起動した CLAT が無効になる

DNS-triggered CLAT disabled after resuming from sleep

- ipv4only.arpa の AAAA レコードを利用して Pref64::/n を検出する方法も検証時はテストしていた
- 再現条件はハッキリしていないがスリープなどから戻ってくると CLAT が止まってしまうのか IPv4 接続性が失われることが
  - その場合でも引き続き Option 108 は有効で native IPv4 に fallback しない
- 再現が時間かかる・難しいため macOS 26 での追試はできてない
  - RA PREF64 を使いましょう

補足: DNS based Pref64::/n discover is not encouraged

- Well known な A レコード ipv4only.arpa (RFC 8880) を定義して、DNS64 が動いている環境だと AAAA レコードが生成されて NAT64 prefix を計算することができるという仕組み
- ただ、RFC 9872 で RFC 8781, RA オプションでの Pref64::/n シグナリングが推奨されている

DNS-based Discovery の主な問題点 (from RFC 9872)

- NAT64 prefix に対応したリゾルバに依存して確実性がない
  - カスタムリゾルバを使われると対応不可。マルチホームでも問題になりうる
- DNS Spoofing 等で IPv4 トラフィックを誘導されてしまうリスク
- 解決した ipv4only.arpa レコードの TTL が切れるまで更新できない
  - RAなら再送すると更新をプッシュすることが可能
    L2ドメイン毎に変更するのも容易
- Discovery が終わって CLAT が起動するまでのレイテンシが高い
  - IPv6 autoconfiguration の後 DNS クエリをして待つので…
    RDNSS無い場合DHCPv6を待つことになりさらに時間かかる

CLAT Internal Address Leakage

- Apple の CLAT は local 192.0.0.1, peer 192.0.0.2 とした link が作成されて peer (192.0.0.2) が IPv4/v6 パケット変換を行う
  - 192/29 は RFC 7335 IPv4 Service Continuity Prefix
- Src 192.0.0.1 なパケットが MDNS, ICMP 等で underlay に漏れてくる問題。同じアドレスを持つ端末が複数いるように見える
- Wi-Fi 環境だとセキュリティ機能で IP Theft と判断されて exclusion されたりすることがある

- この問題は APRICOT 2024 Wi-Fi の writeup 等で報告されていたため、RubyKaigi 2025 の時点で事前に対応済.
- Cisco WLC の場合:
  - IP Theft と Reuse を Client Exclusion の対象外に設定する

    ```
    config wps client-exclusion ip-theft disable
    ```
- たぶん最新の macOS/iOS では直ってる…

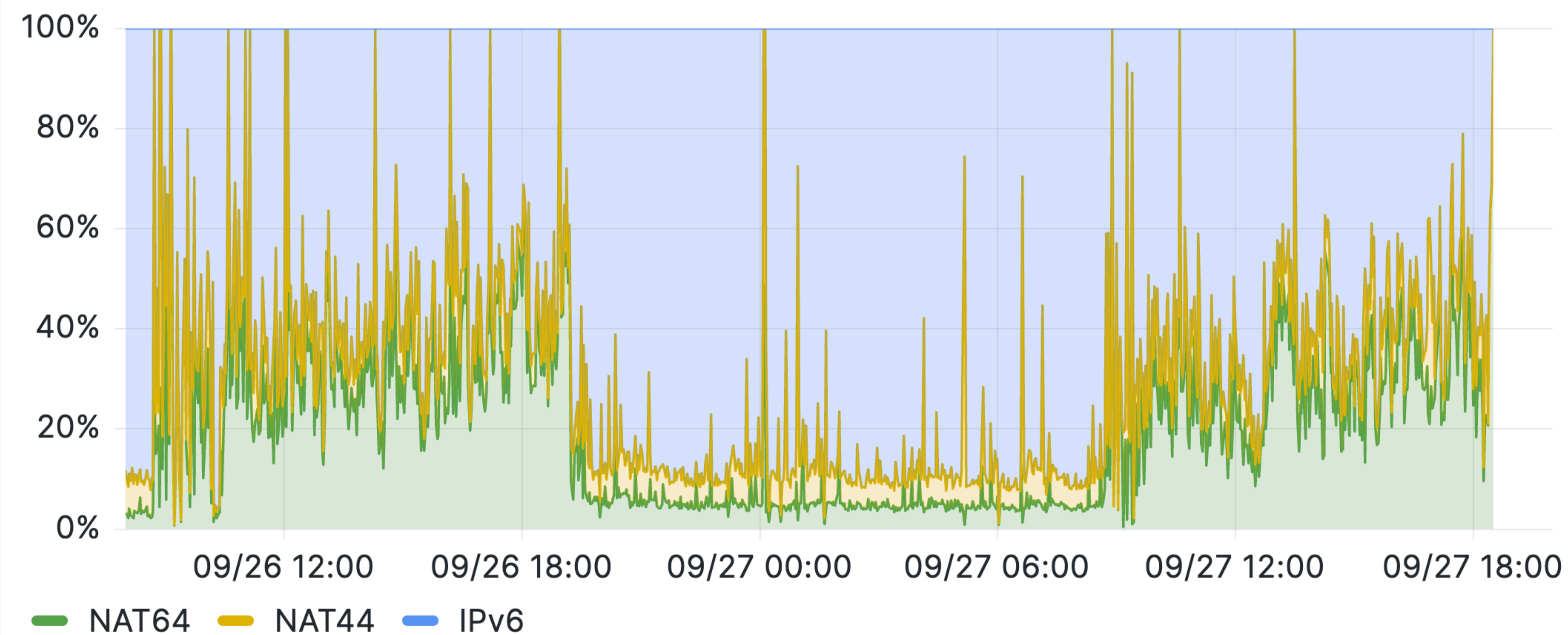
https://blog.afrinic.net/apricot-ipv6-only#:~:text=Remaining%20issues%20were%20minor

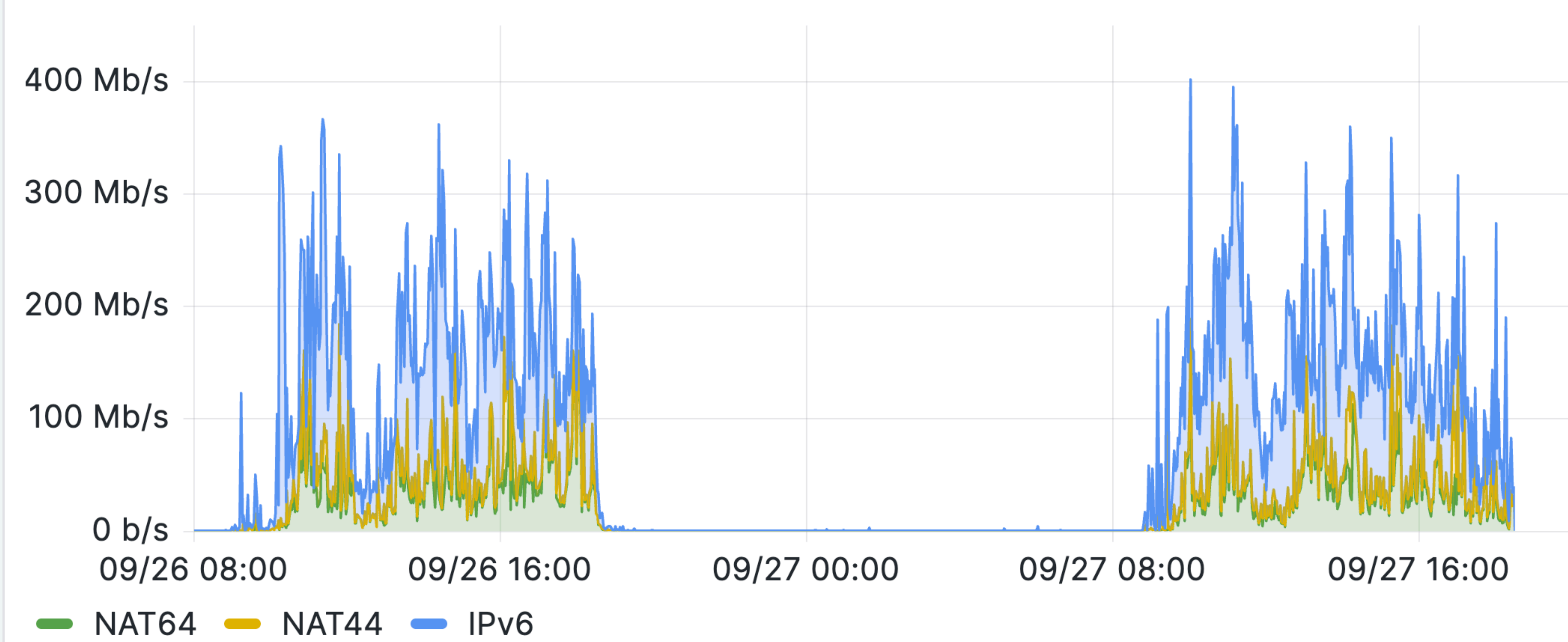
Retry at Kaigi on Rails 2025

- 2025/9 の Kaigi on Rails 2025 でここまでの内容を踏まえて再試行
- 結果、概ね問題なし
  - Wi-Fi クライアントに IPv6 RA が届かず接続が失われる問題はあったが本件と関連あるかは不明…

- 以下の問題は macOS 15.5, iOS 18.5 頃,
  遅くとも macOS/iOS 26 では修正されている
  - No response to Unicast Neighbor Solicitations
  - CLAT internal address leak
  - No response to ICMPv6 Echo Requests
- IPv4 アドレスで traceroute ができない問題はまだのこっている
  - IPv6 アドレスに変換すれば当然可能だけど

- 900 clients
- IPv4 の 80% 程度が NAT64 を利用
- 全体の 50-60% はそもそも IPv6 Native (!)





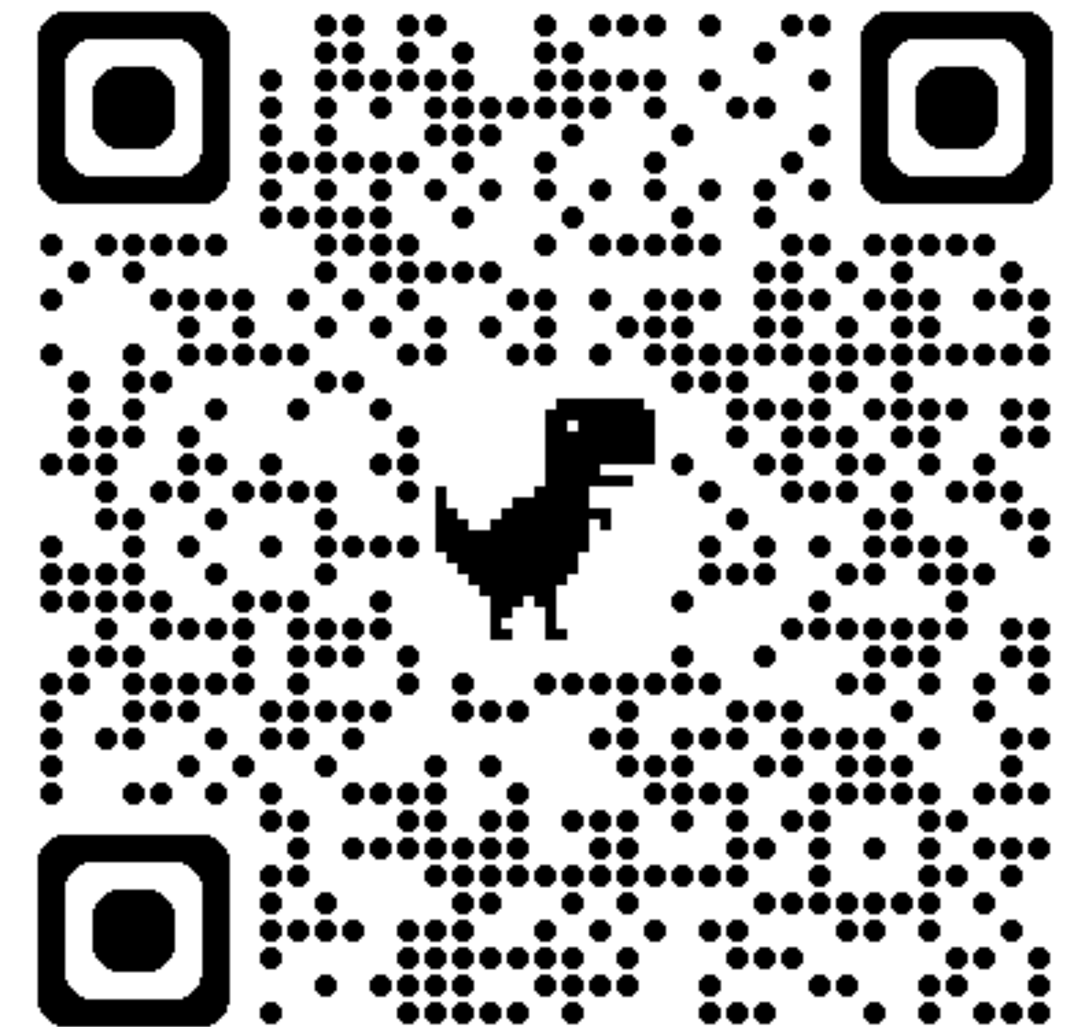
Kaigi on Rails は RubyKaigi 2025 に比べると小規模 (RubyKaigi は 1,200-1,500 clients)

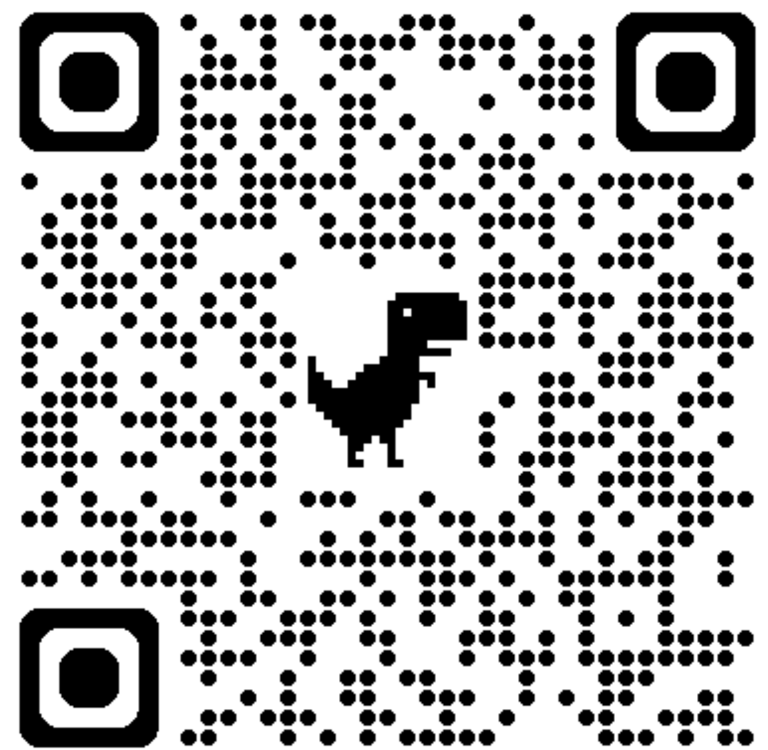
- draft-ietf-v6ops-6mops-04 § 7.2. での推奨もあり、対応は入れているけど問題があったかないか分かってないのは下記
  - IP Fragment
    - QUIC は MSS Clamping できないよ 😎
  - ESP packet (IPsec)
- 特に ESP は需要がないか、問題が報告されていないかは分からない

- v4→v6 のパケット処理の時だけヘッダサイズが増えるため、v4 Internet → CLAT の向きだけ IPv4 MTU -20 となる
  - 逆はヘッダが縮むので考えなくて良い
- RubyKaigi では対外接続はトンネルで MTU が低めに設定されている。v4 Internet 側には CLAT 側の MTU に近いパケットが来ることはない
  - 既にFragmentされてるかFragmentation Neededを上流で返してる
- したがって、sorah/xlat ではやってきたパケットを 2 つ以上に分割はしない。Fragment Header の翻訳だけ実装した

- ✦ sorah.jp が以下をお届けしました
  - ✦ IPv6-mostly と IPv6 移行技術 Recap
  - ✦ RubyKaigi 2025, Kaigi on Rails 2025 での運用実績
- ✦ Many thanks: RubyKaigi NOC team members

Slides ↓





←コード+
ブログ等リンク集

https://github.com/ruby-no-kai/rubykaigi-net

https://speakerdeck.com/sorah/iw2025-rubykaigi-v6mostly