Faster SRv6 D-plane with XDP

Ryoga Saito

自己紹介

- 齋藤遼河
- 国立木更津工業高等専門学校 情報工学科 5年
- LINE株式会社
  - ITサービスセンター Verda室 ネットワーク開発チーム

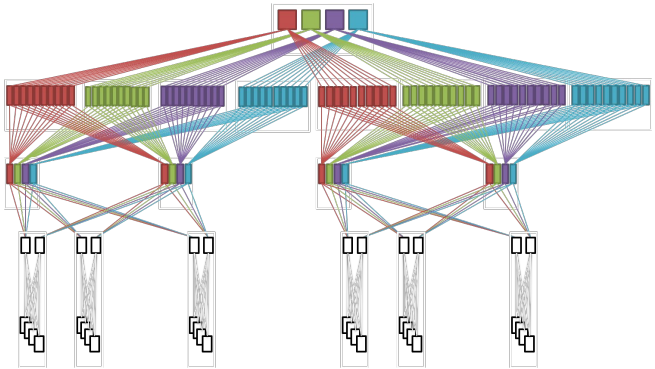
JANOG44 “Next Data Center Networking with SRv6” : https://www.janog.gr.jp/meeting/janog44/program/srv6

Background

背景

Full L3 CLOS Network

- シングルテナントなネットワーク
- LINEのメッセンジャーやファミリーサービスなどが稼働



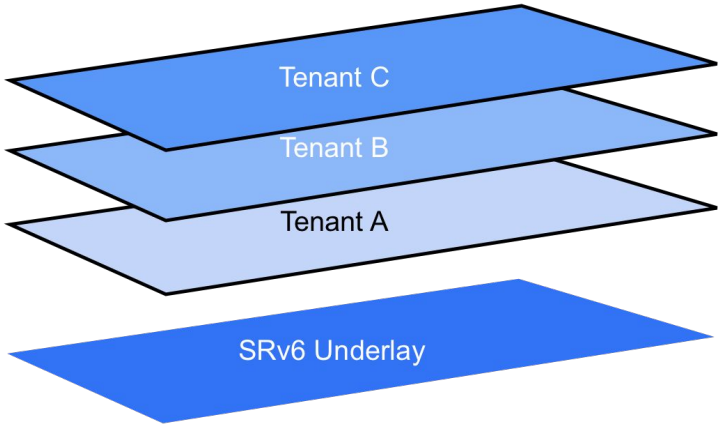
特定サービス専用のネットワーク

- サービス固有の要件を持つサービスが稼働
- サービス毎にネットワークを構築

アンダーレイネットワークの断片化
設計・構築にかかる時間の増加
運用コストの増加

背景

- アンダーレイネットワークを共通化し、運用負荷を軽減する
- オーバレイネットワークでテナント間分離し、それぞれのポリシーを実現する

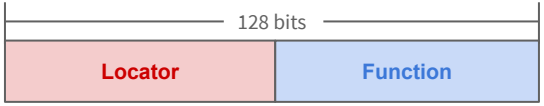


柔軟にスケールするオーバレイ
テナント毎に分離・独立したセキュリティ
将来的なサービスチェイニング

シンプルなL3のアンダーレイ

SRv6の概要

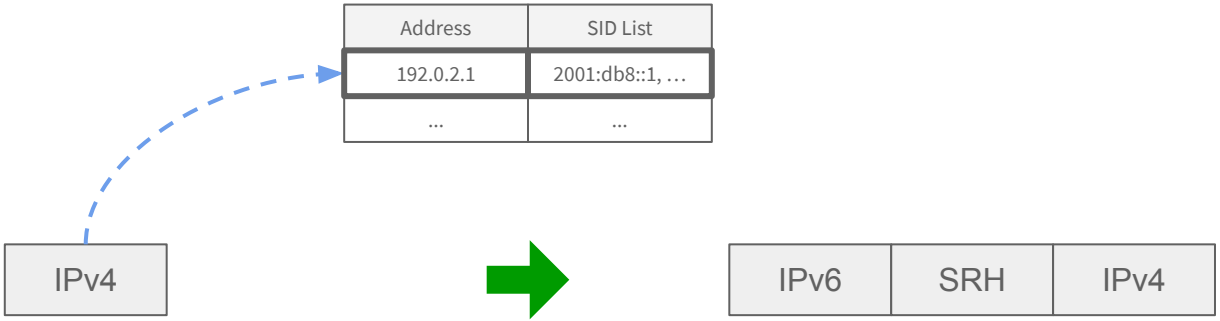
- Segment ID（SID）
  - Segment（パケットに対する指示）を表す ID
    - Locator: Functionが実行されるノードを示す情報
    - Function: Functionを示す情報



- Segment Routing Header（SRH）
  - IPv6のルーティング拡張ヘッダ
  - Segment List（通るべきSIDのリスト）やSegments Left（Segment List内の現在のSIDのオフセット）などが含まれている

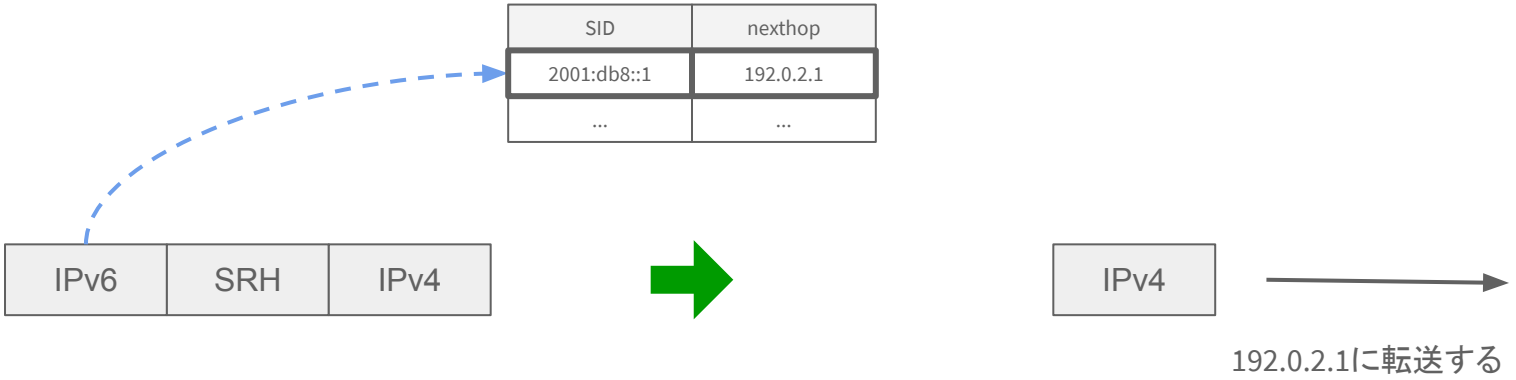
SRv6の概要

- Well-known Functions
  - H.Encaps (SR Headend with Encapsulation in an SRv6 Policy):
    IPv6ヘッダ, SRHを用いてカプセル化し転送する



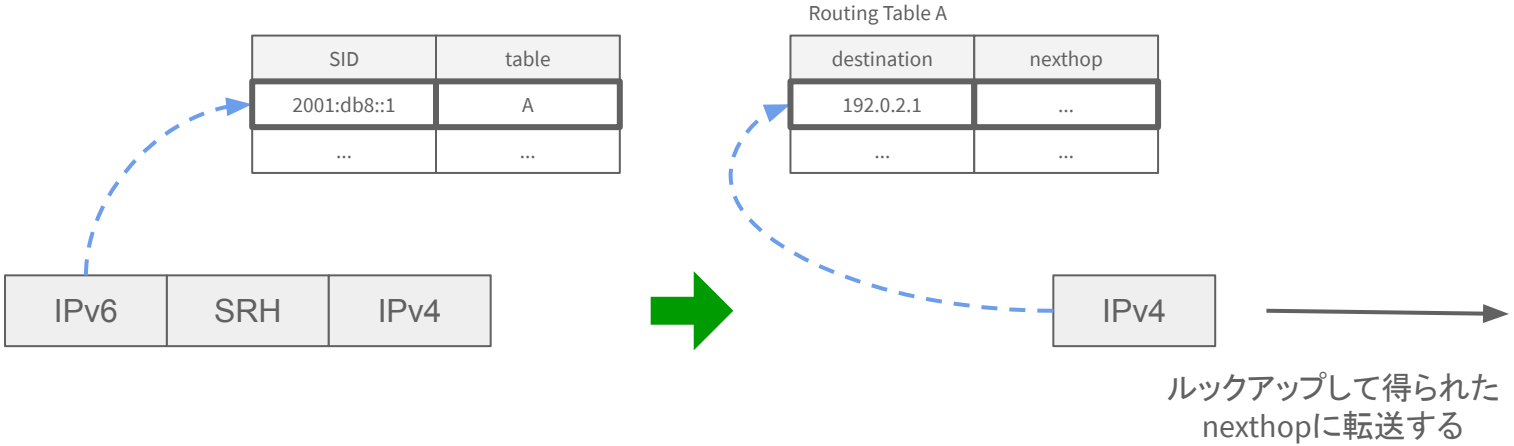
SRv6の概要

- Well-known Functions
  - End.DX4 (Decapsulation and IPv4 Cross-Connect):
    IPv6ヘッダ, SRHを外し、指定されたnexthopに転送する

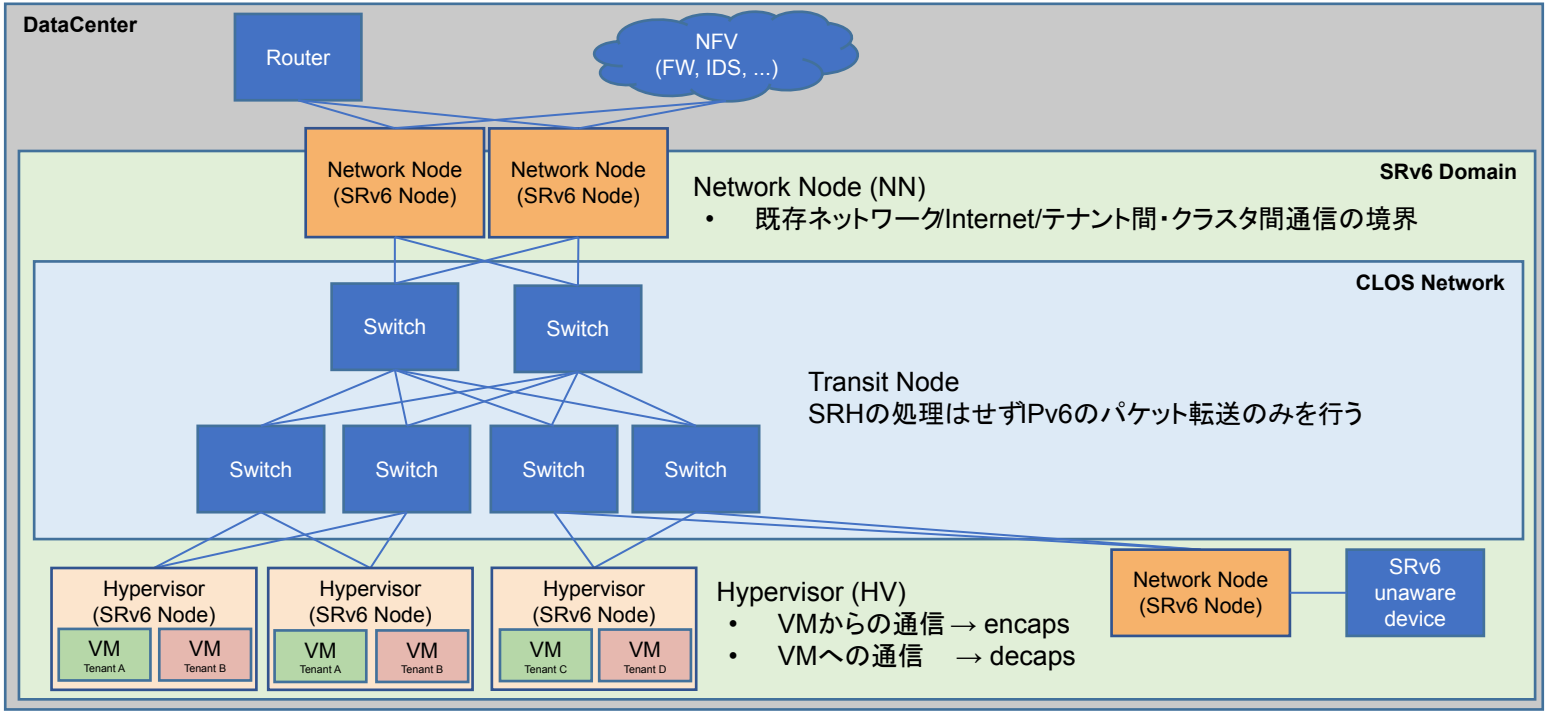


SRv6の概要

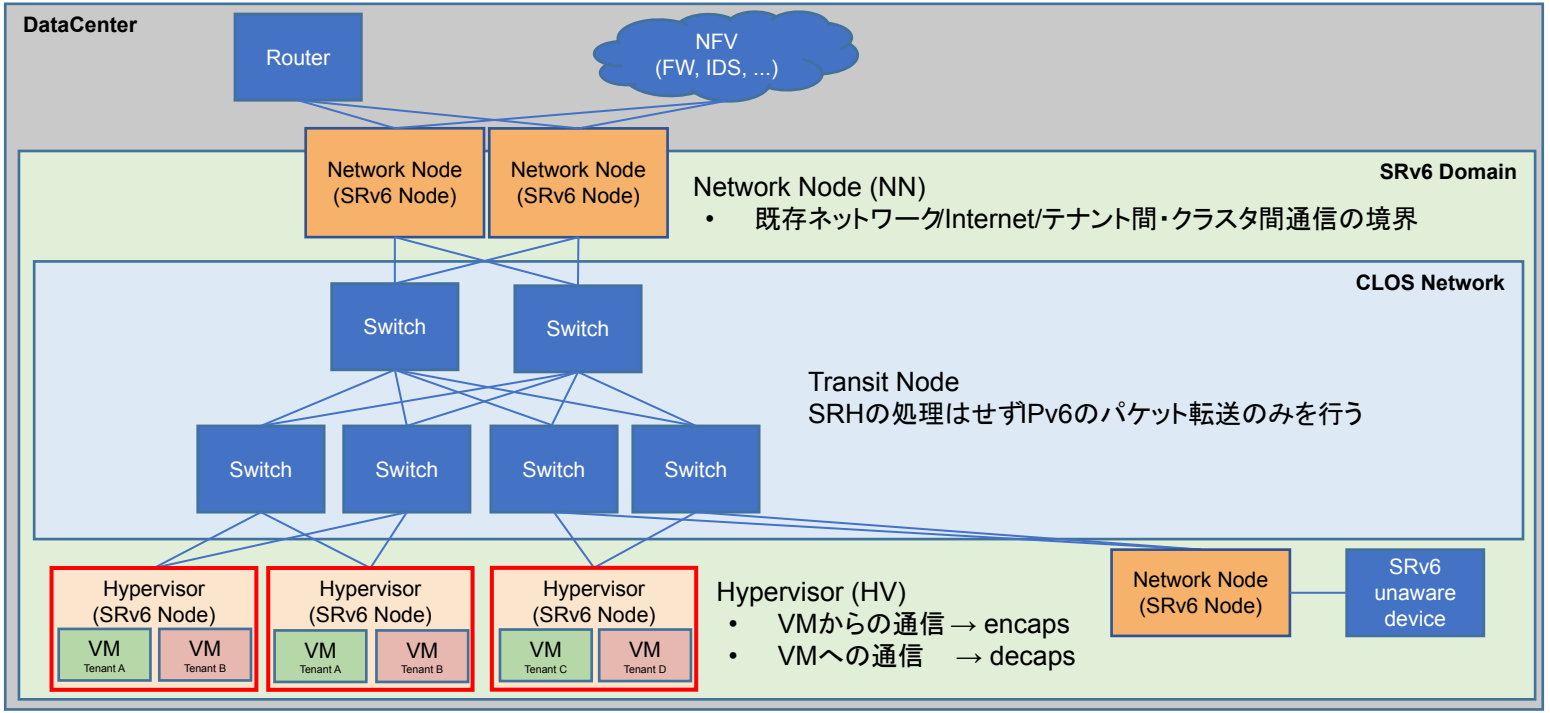
- Well-known Functions
  - End.DT4 (Decapsulation and Specific IPv4 Table Lookup):
    IPv6ヘッダ, SRHを外し、指定されたルーティングテーブルを用いて転送する



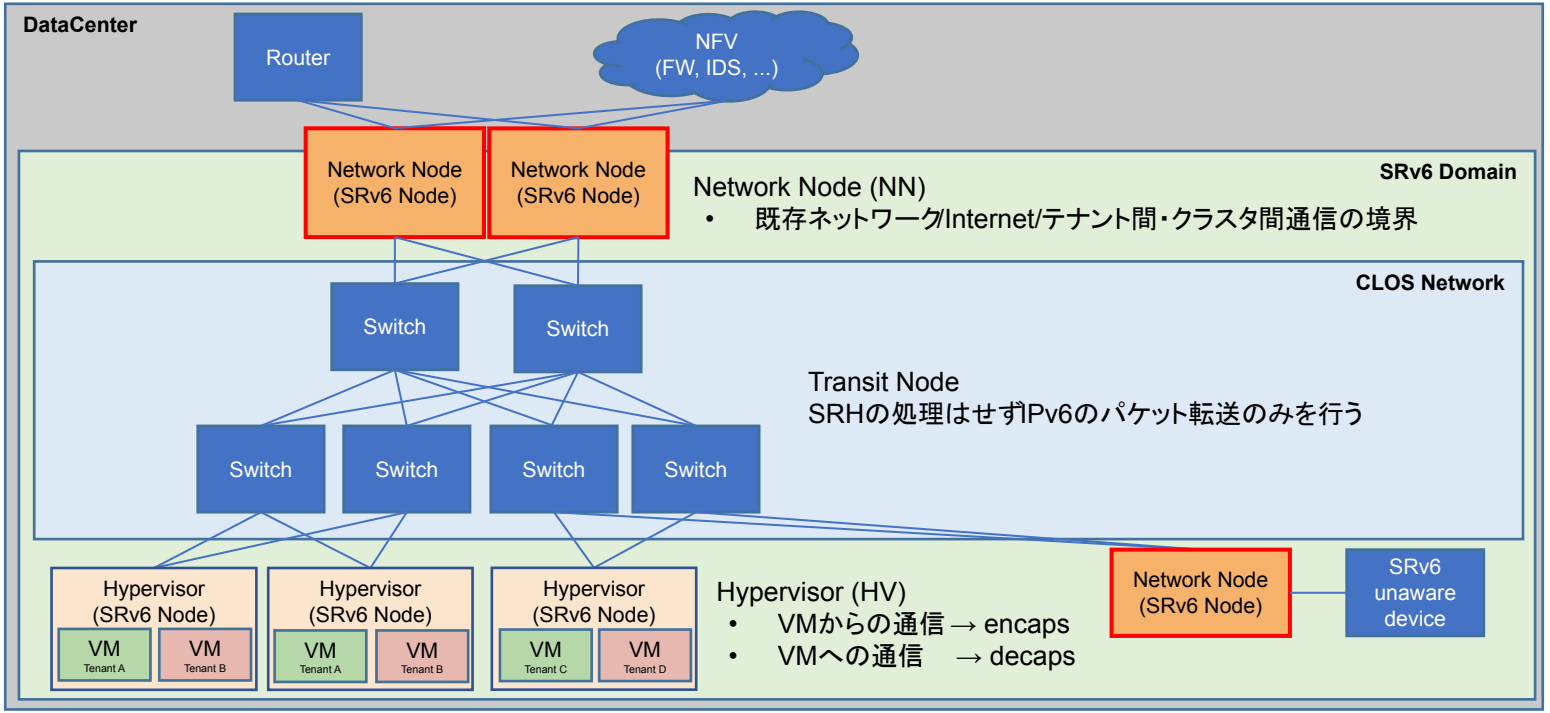
LINEにおける現在の構成



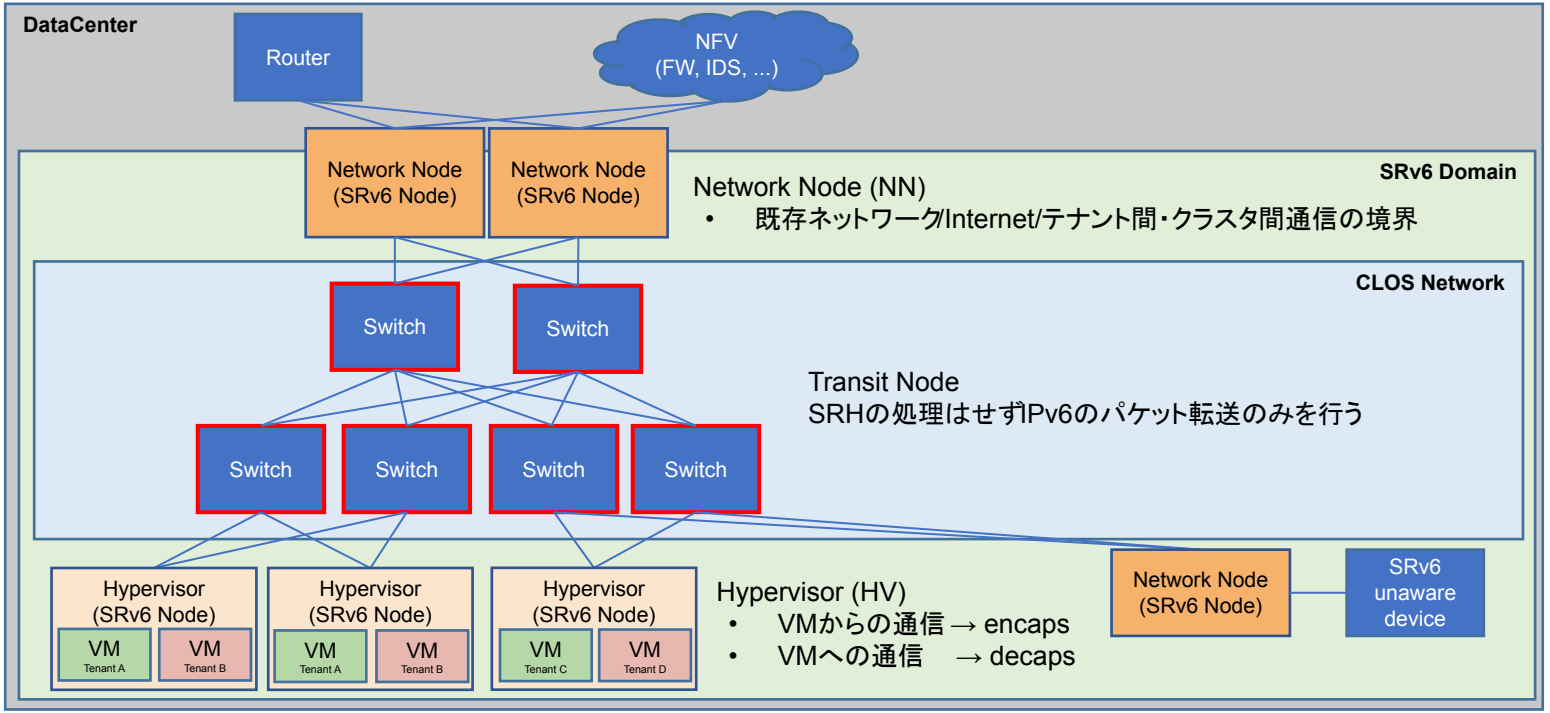
LINEにおける現在の構成



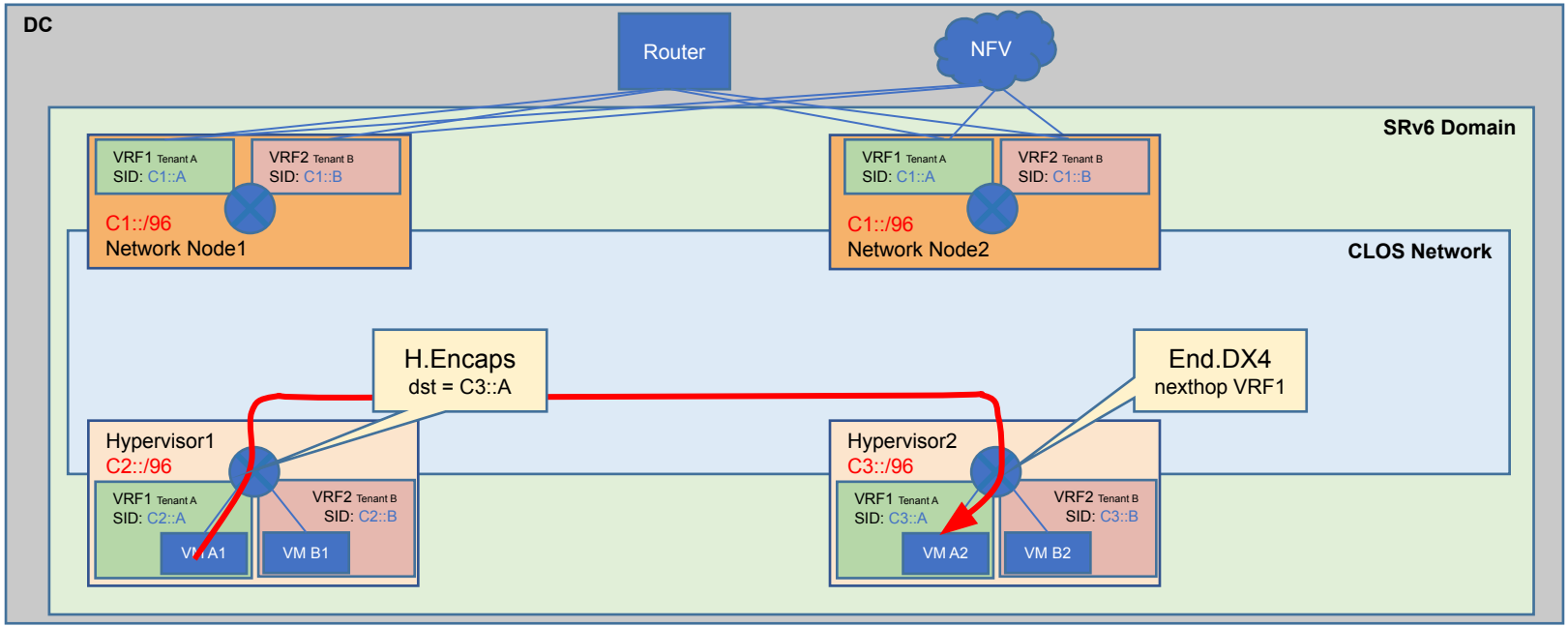
LINEにおける現在の構成



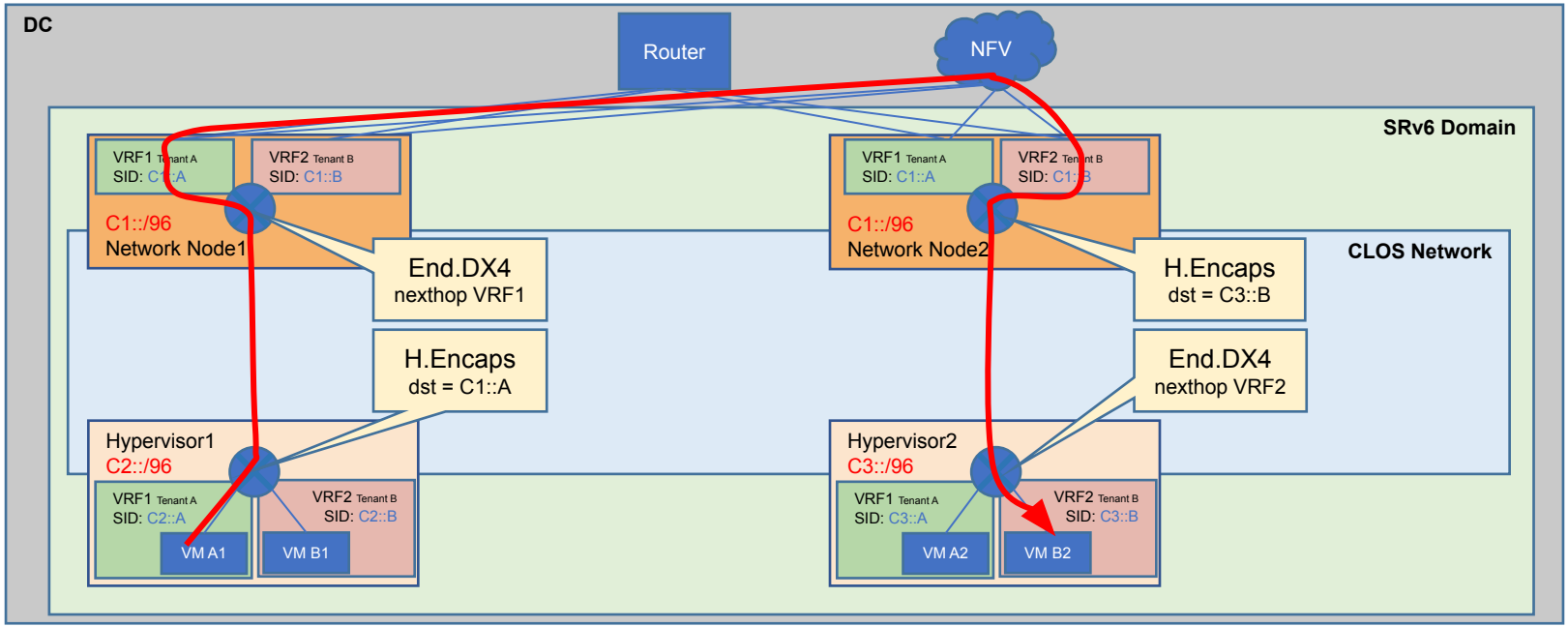
LINEにおける現在の構成



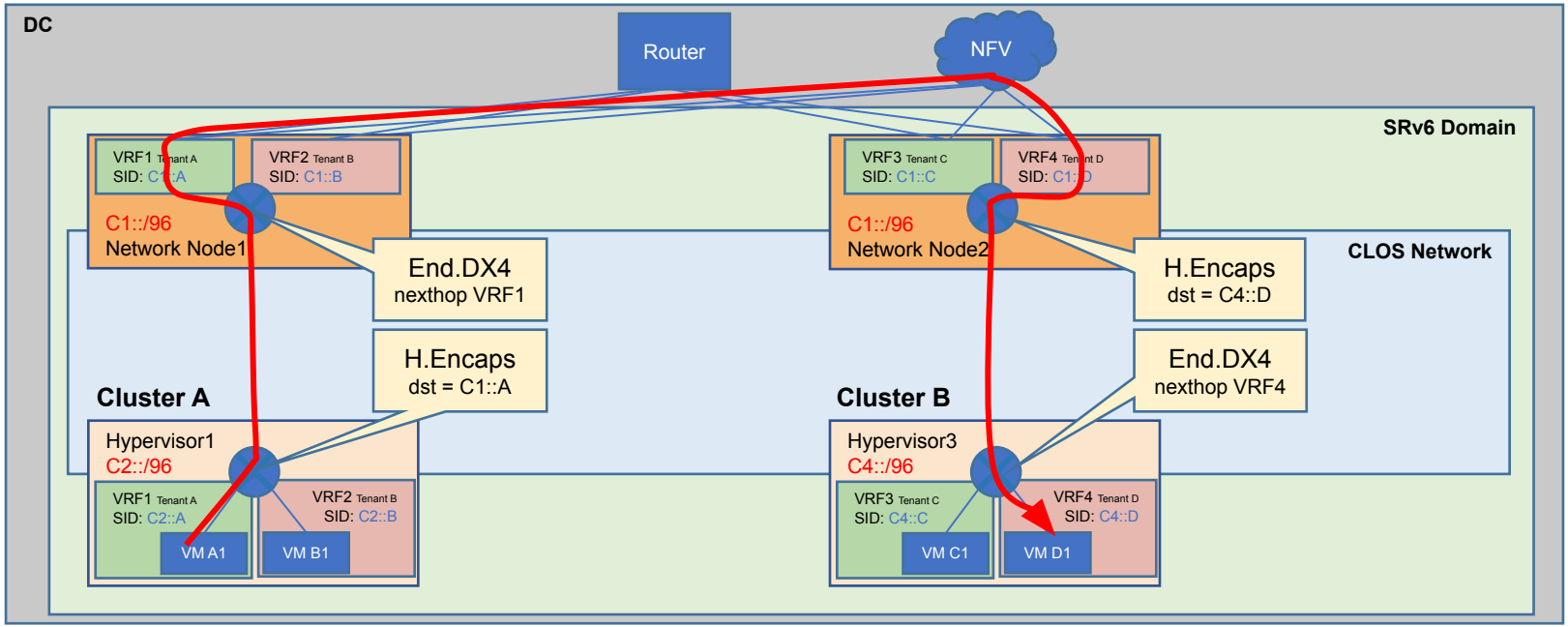
テナント内通信


DC
Router
NFV
SRv6 Domain
VRF1 Tenant A
SID: C1::A
VRF2 Tenant B
SID: C1::B
C1::/96
Network Node1
VRF1 Tenant A
SID: C1::A
VRF2 Tenant B
SID: C1::B
C1::/96
Network Node2
CLOS Network
H.Encaps
dst = C3::A
End.DX4
nexthop VRF1
Hypervisor1
C2::/96
VRF1 Tenant A
SID: C2::A
VRF2 Tenant B
SID: C2::B
VM A1
VM B1
Hypervisor2
C3::/96
VRF1 Tenant A
SID: C3::A
VRF2 Tenant B
SID: C3::B
VM A2
VM B2

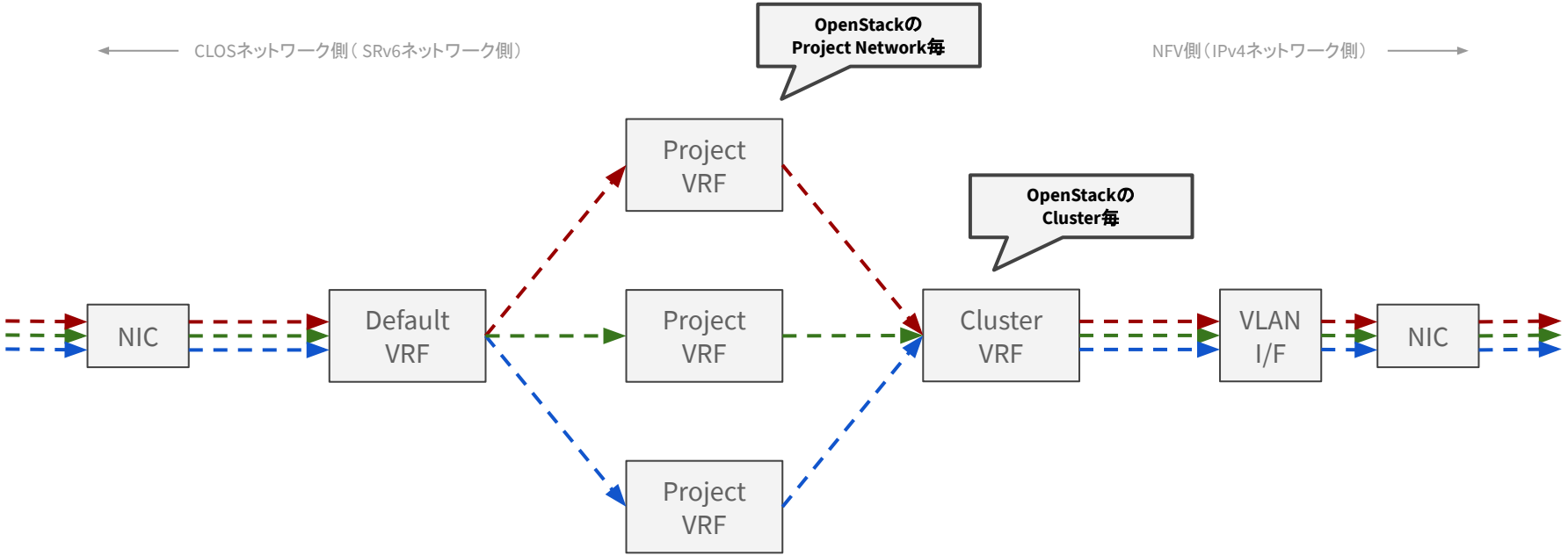
テナント間通信


DC
Router
NFV
SRv6 Domain
VRF1 Tenant A
SID: C1::A
VRF2 Tenant B
SID: C1::B
C1::/96
Network Node1
End.DX4
nexthop VRF1
H.Encaps
dst = C1::A
CLOS Network
VRF1 Tenant A
SID: C1::A
VRF2 Tenant B
SID: C1::B
C1::/96
Network Node2
H.Encaps
dst = C3::B
End.DX4
nexthop VRF2
Hypervisor1
C2::/96
VRF1 Tenant A
SID: C2::A
VRF2 Tenant B
SID: C2::B
VM A1
VM B1
Hypervisor2
C3::/96
VRF1 Tenant A
SID: C3::A
VRF2 Tenant B
SID: C3::B
VM A2
VM B2

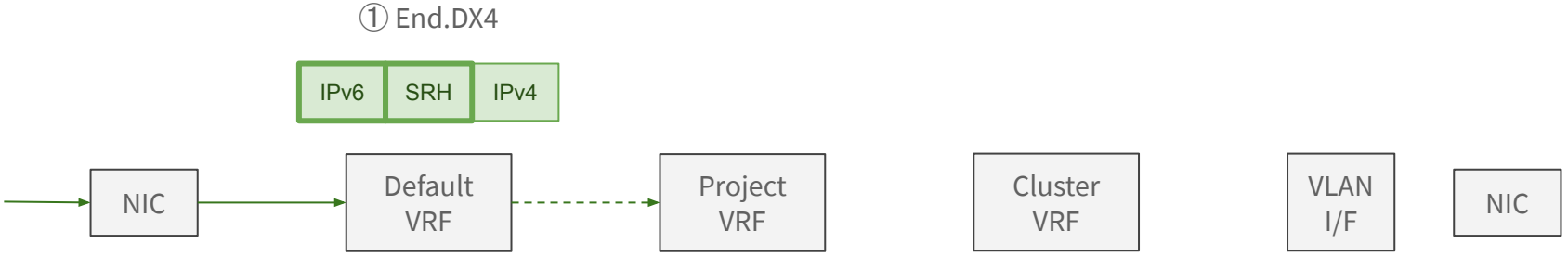
クラスタ間通信


DC
Router
NFV
SRv6 Domain
VRF1 Tenant A
SID: C1::A
VRF2 Tenant B
SID: C1::B
C1::/96
Network Node1
End.DX4
nexthop VRF1
H.Encaps
dst = C1::A
Cluster A
Hypervisor1
C2::/96
VRF1 Tenant A
SID: C2::A
VRF2 Tenant B
SID: C2::B
VM A1
VM B1
VRF3 Tenant C
SID: C1::C
VRF4 Tenant D
SID: C1::D
C1::/96
Network Node2
H.Encaps
dst = C4::D
End.DX4
nexthop VRF4
CLOS Network
Cluster B
Hypervisor3
C4::/96
VRF3 Tenant C
SID: C4::C
VRF4 Tenant D
SID: C4::D
VM C1
VM D1

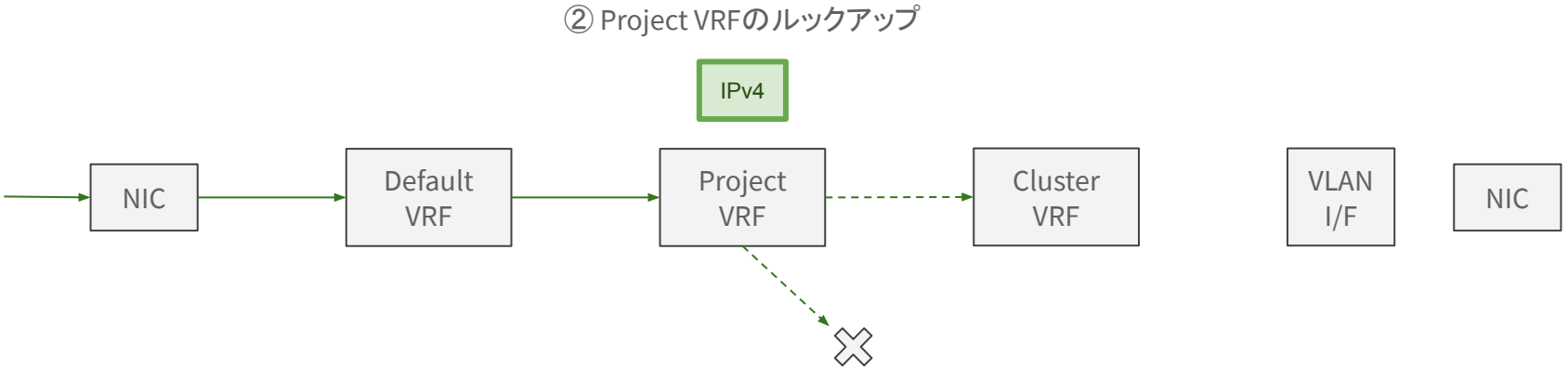
ネットワークノードの処理



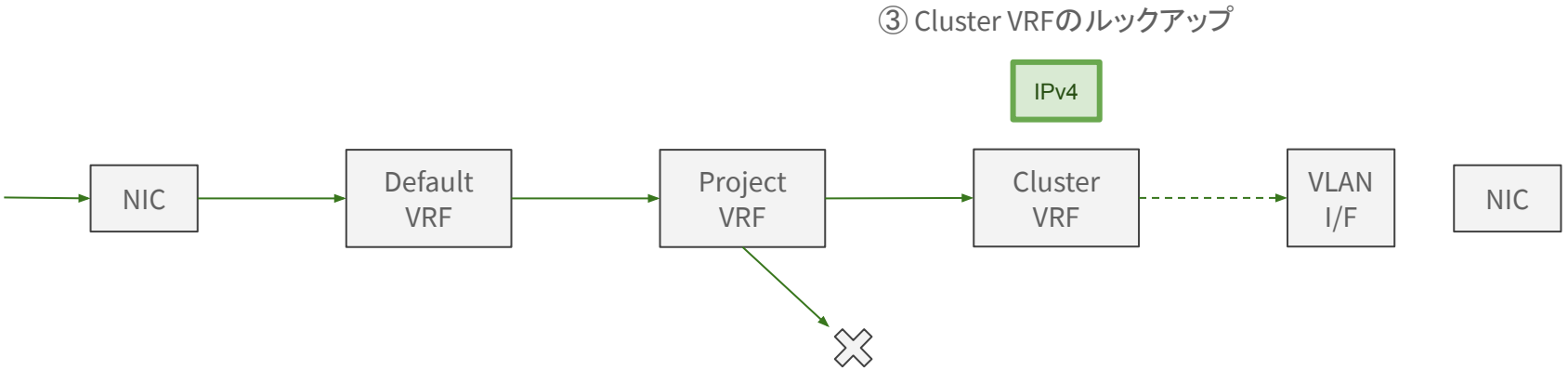
ネットワークノードの処理



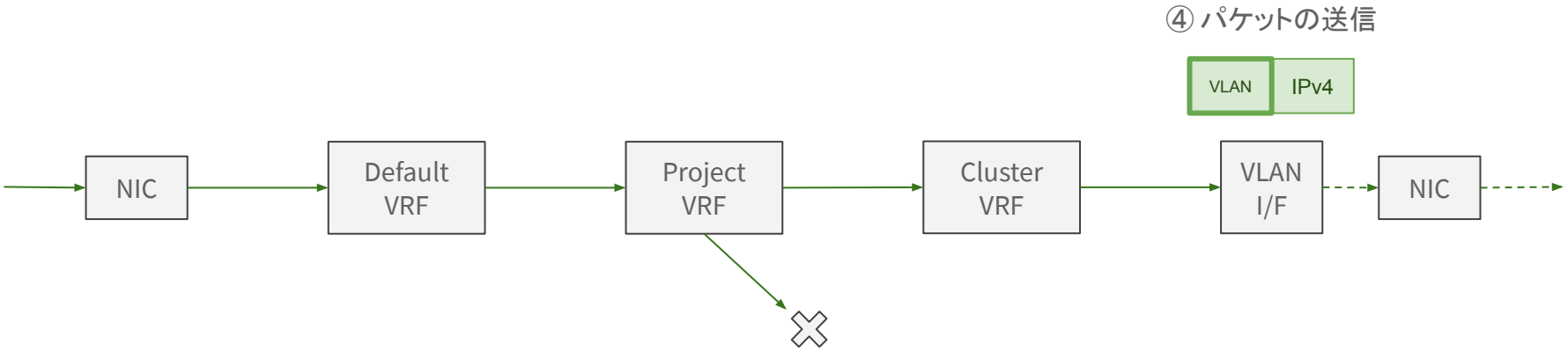
ネットワークノードの処理



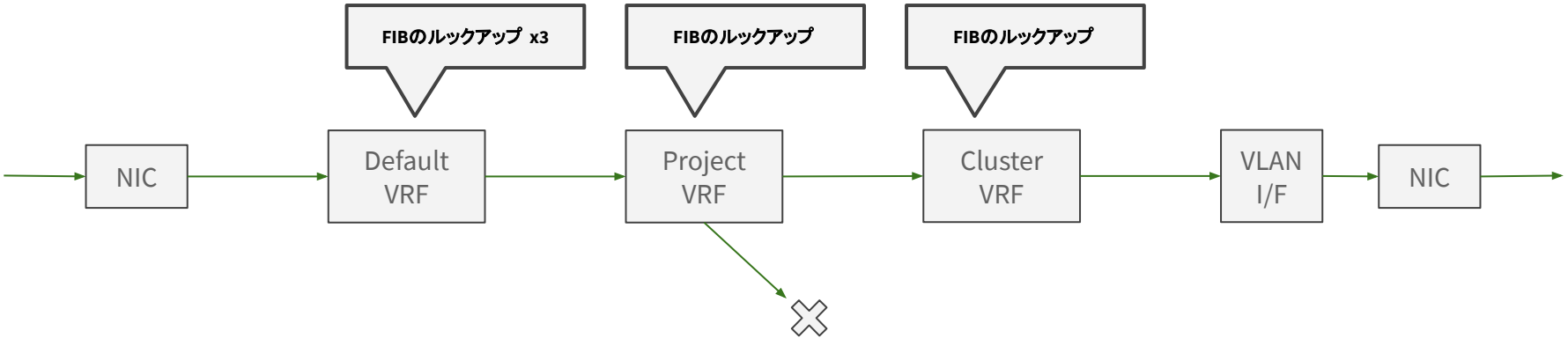
ネットワークノードの処理



ネットワークノードの処理



ネットワークノードの処理



問題点

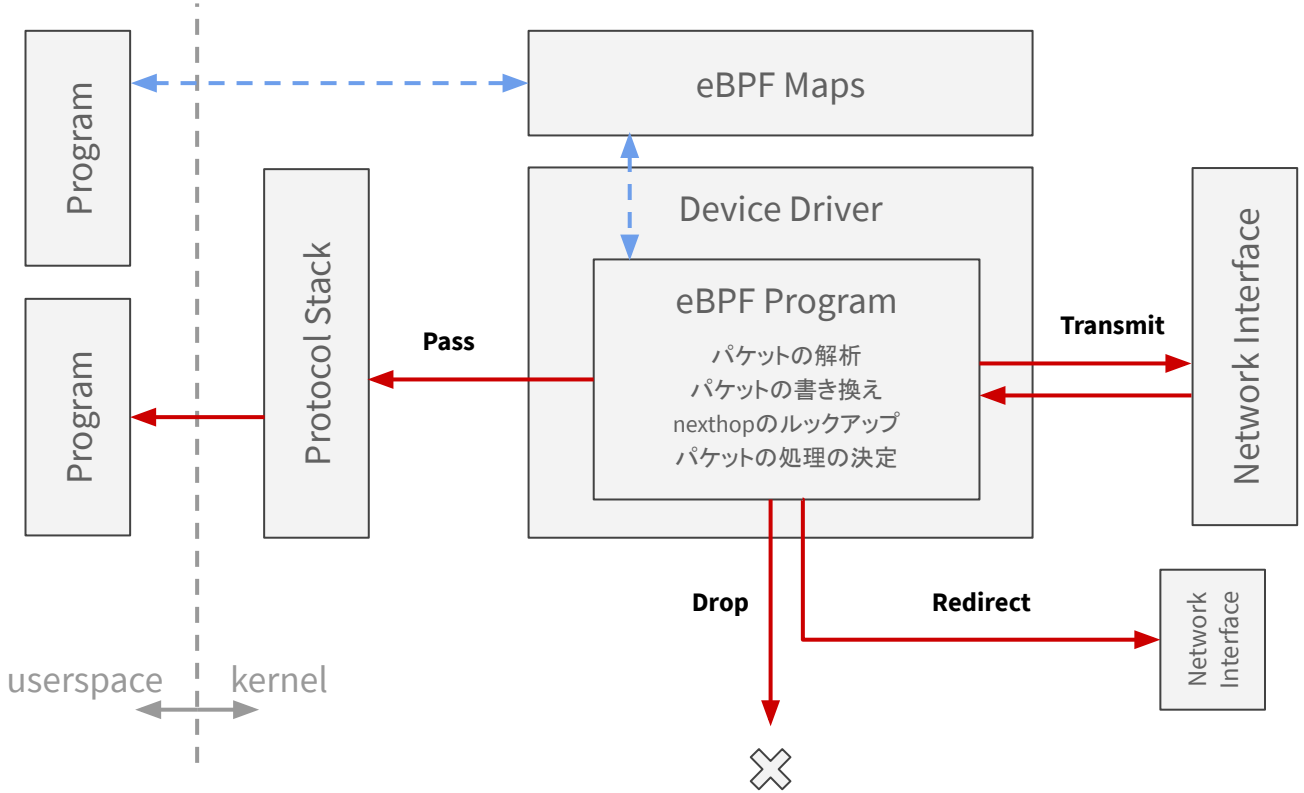
- ネットワーク構成
  - LinuxのVRFの仕様やルーティングを駆使して頑張っているが、構成が複雑になる
- 性能問題
  - ソフトウェアでの FIBのルックアップは比較的重いコストになる
  - 何度もFIBのルックアップを行うため、普通の通信よりも性能が出ない
  - ネットワークノードを模した性能測定では普通の IPv6転送の半分程度しか性能が出ない
    普通のIPv6転送: 1.09Mpps (Single Flow, 64bytes)
    既存の構成: 0.51Mpps (Single Flow, 64bytes)

データプレーンの実装方法

- **XDP**
- DPDK（VPP）
- Hardware Switch
- …

- L4LBでの使用実績がある
- ARPやICMPの処理をカーネルに依頼することができる
- LinuxカーネルのFIBを利用することができ、既存の構成からの変更点を少なくすることができる

XDP（eXpress Data Path）の概要

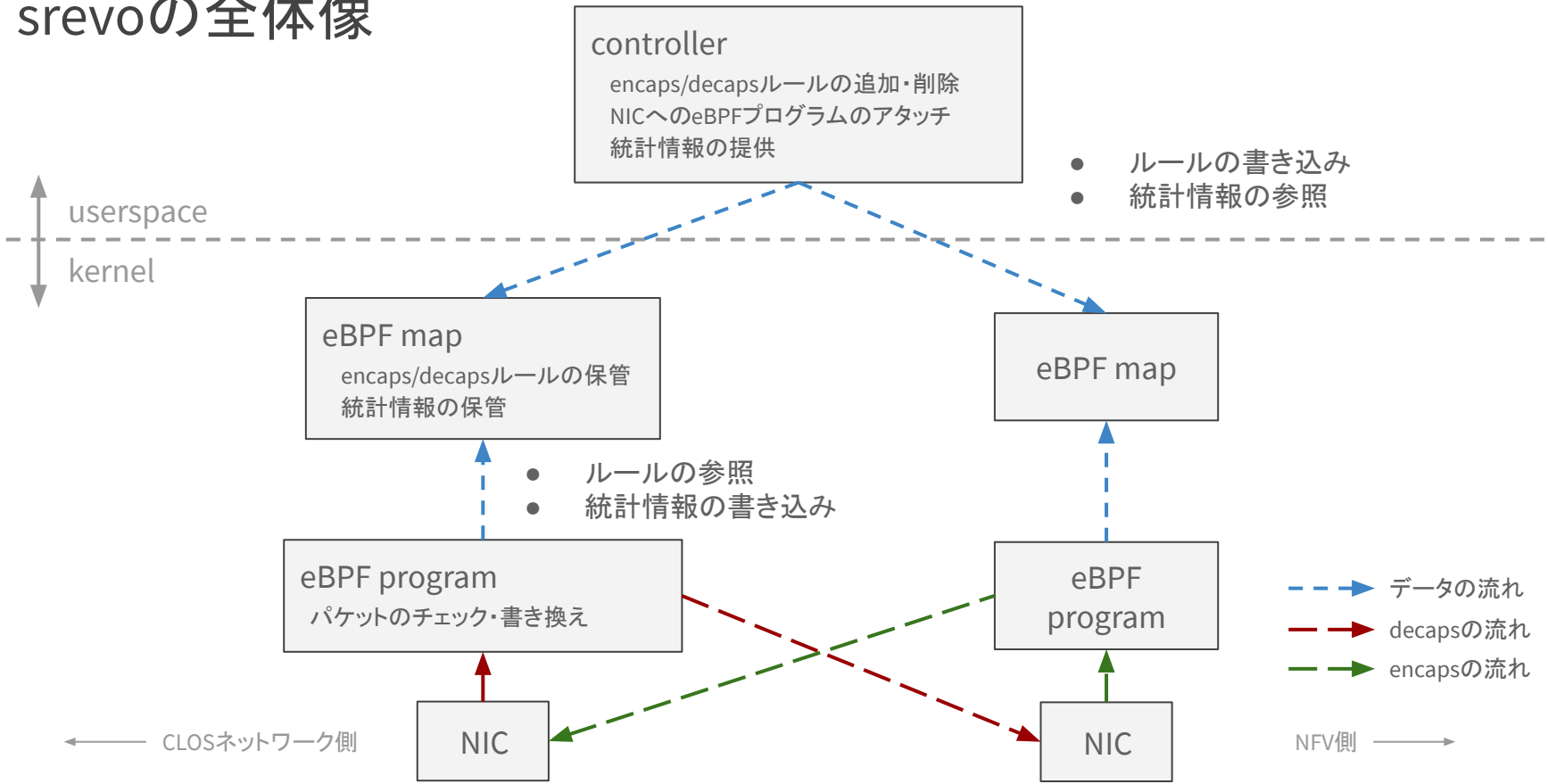


srevo : Faster SRv6 D-plane with XDP

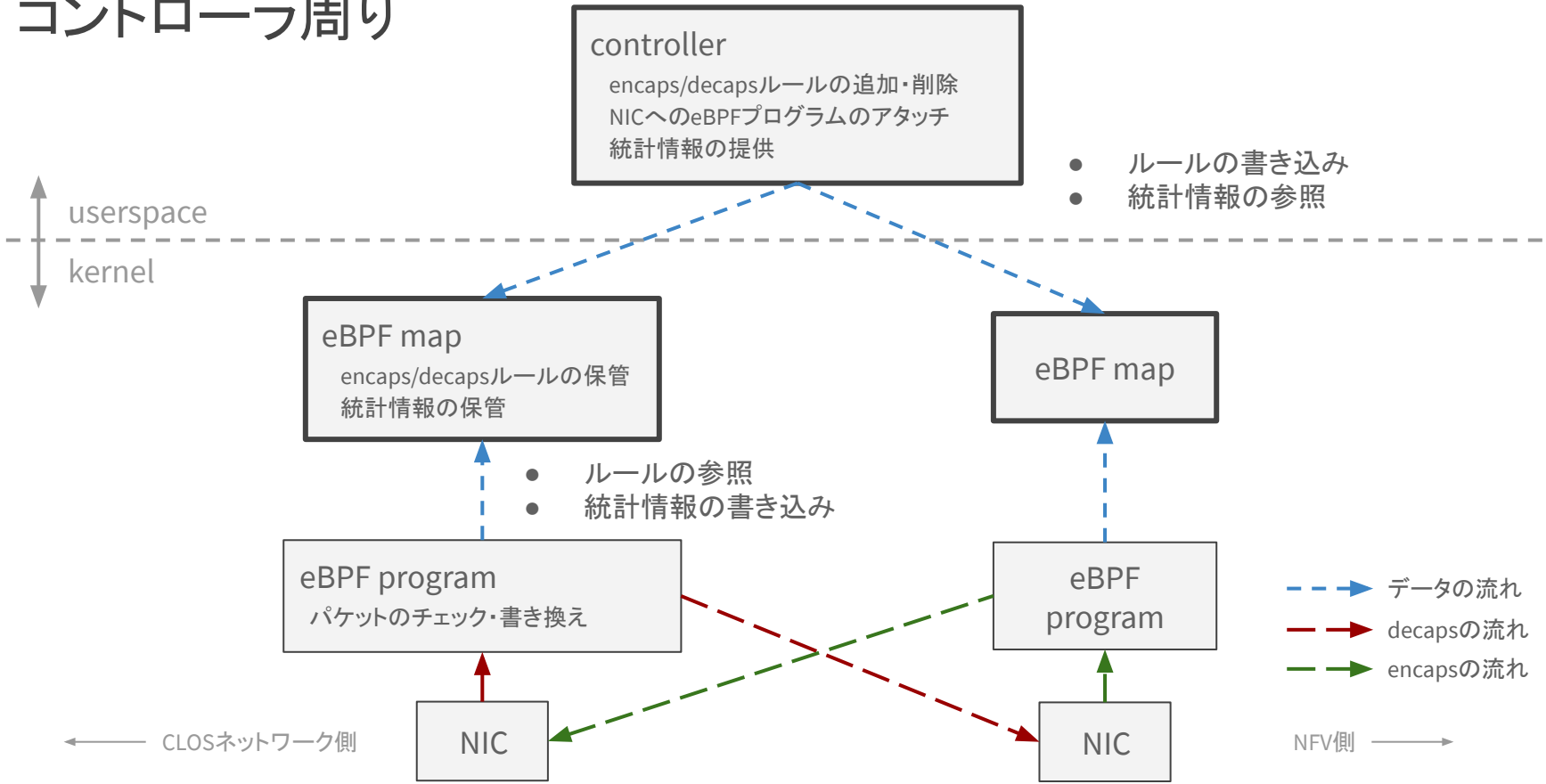
srevoの概要

- XDPを用いたSRv6のパケット処理基盤
  - さまざまな最適化を行い、既存の方法よりも高速に SRv6のパケットを処理できる
- LINEの用途に特化させることでシンプルな実装
  - 必要のない機能は実装を省いている
  - データプレーン自体は約 500行

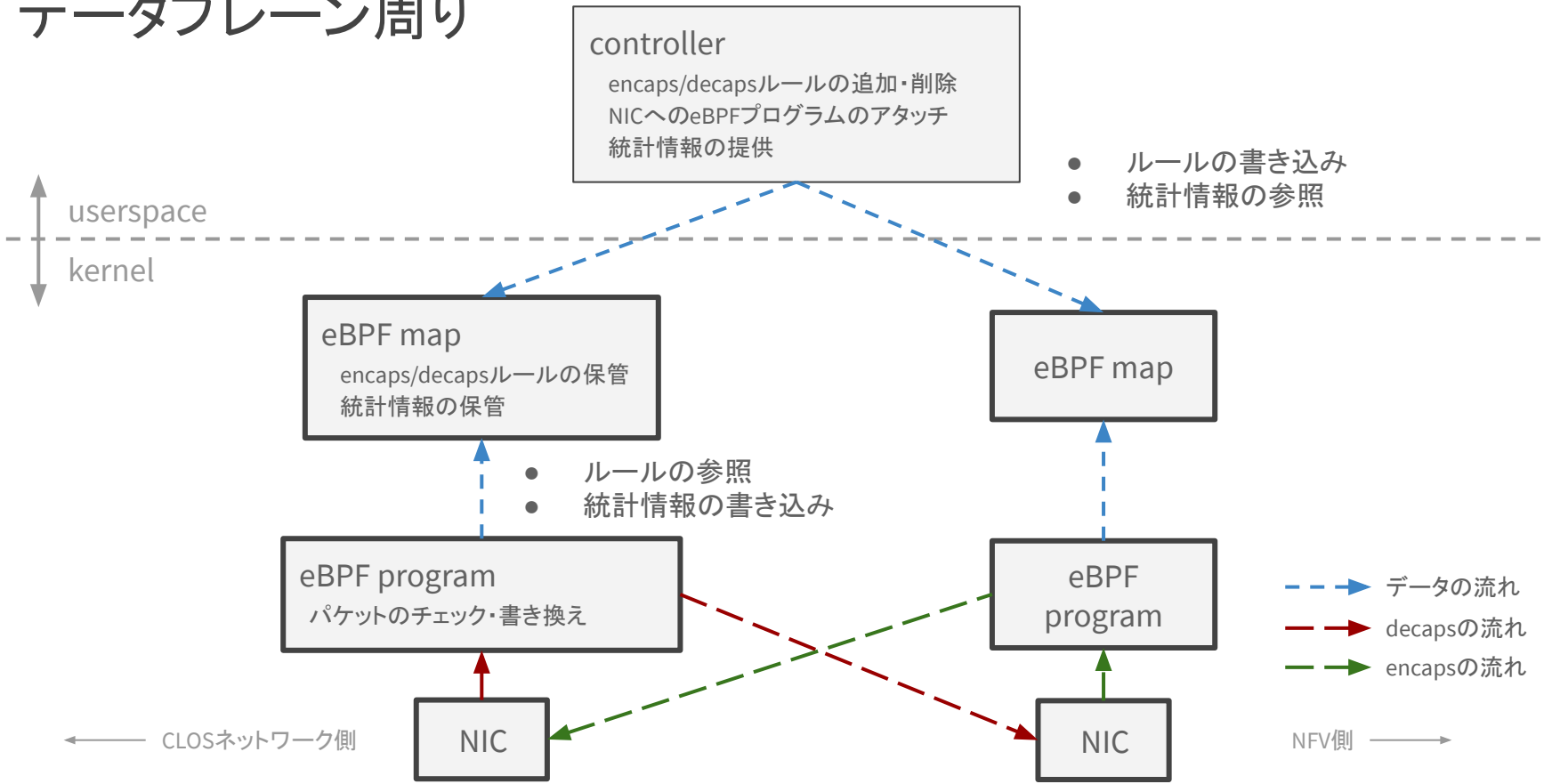
srevoの全体像


controller
encaps/decapsルールの追加・削除
NICへのeBPFプログラムのアタッチ
統計情報の提供
ルールの書き込み
統計情報の参照
userspace
kernel
eBPF map
encaps/decapsルールの保管
統計情報の保管
eBPF map
ルールの参照
統計情報の書き込み
eBPF program
パケットのチェック・書き換え
eBPF program
データの流れ
decapsの流れ
encapsの流れ
NIC
NIC
CLOSネットワーク側
NFV側

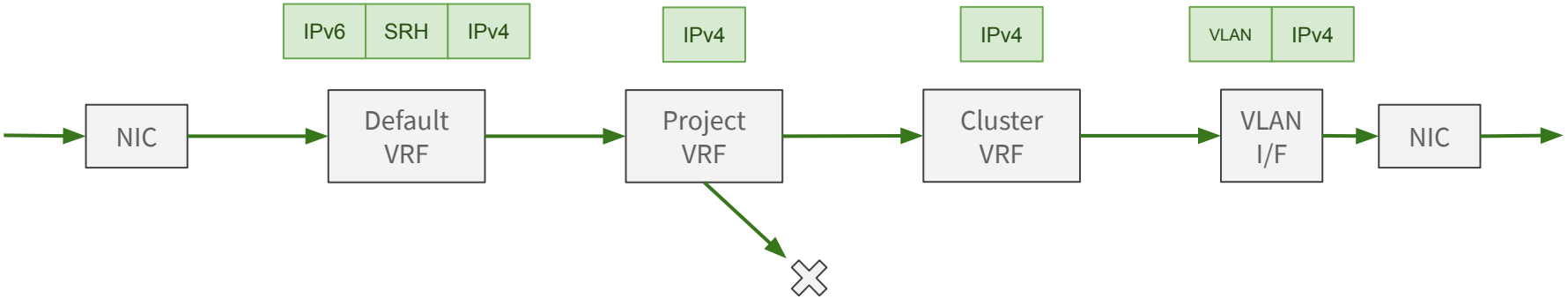
コントローラ周り


controller
encaps/decapsルールの追加・削除
NICへのeBPFプログラムのアタッチ
統計情報の提供
ルールの書き込み
統計情報の参照
userspace
kernel
eBPF map
encaps/decapsルールの保管
統計情報の保管
eBPF map
ルールの参照
統計情報の書き込み
eBPF program
パケットのチェック・書き換え
eBPF program
データの流れ
decapsの流れ
encapsの流れ
NIC
NIC
CLOSネットワーク側
NFV側

データプレーン周り


controller
encaps/decapsルールの追加・削除
NICへのeBPFプログラムのアタッチ
統計情報の提供
ルールの書き込み
統計情報の参照
userspace
kernel
eBPF map
encaps/decapsルールの保管
統計情報の保管
eBPF map
ルールの参照
統計情報の書き込み
eBPF program
パケットのチェック・書き換え
eBPF program
データの流れ
decapsの流れ
encapsの流れ
NIC
NIC
CLOSネットワーク側
NFV側

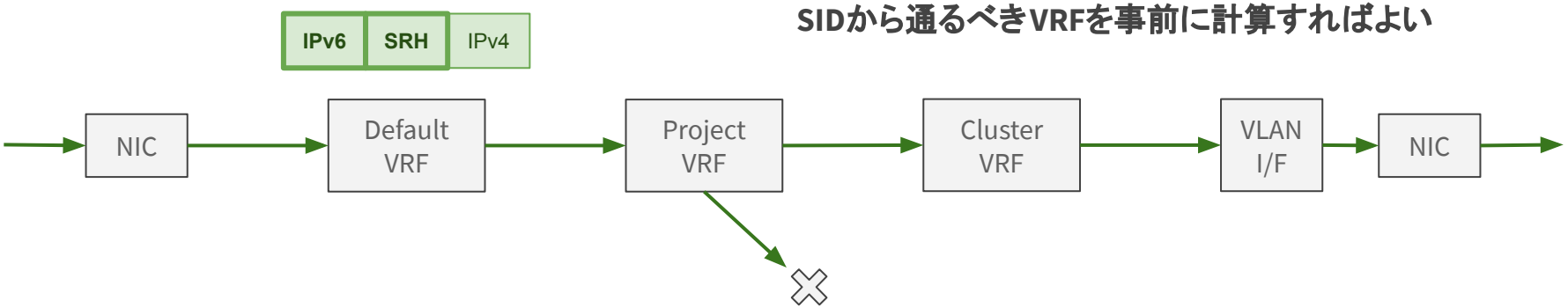
最適化の方針


IPv6
SRH
IPv4
IPv4
IPv4
VLAN
IPv4
NIC
Default VRF
Project VRF
Cluster VRF
VLAN I/F
NIC

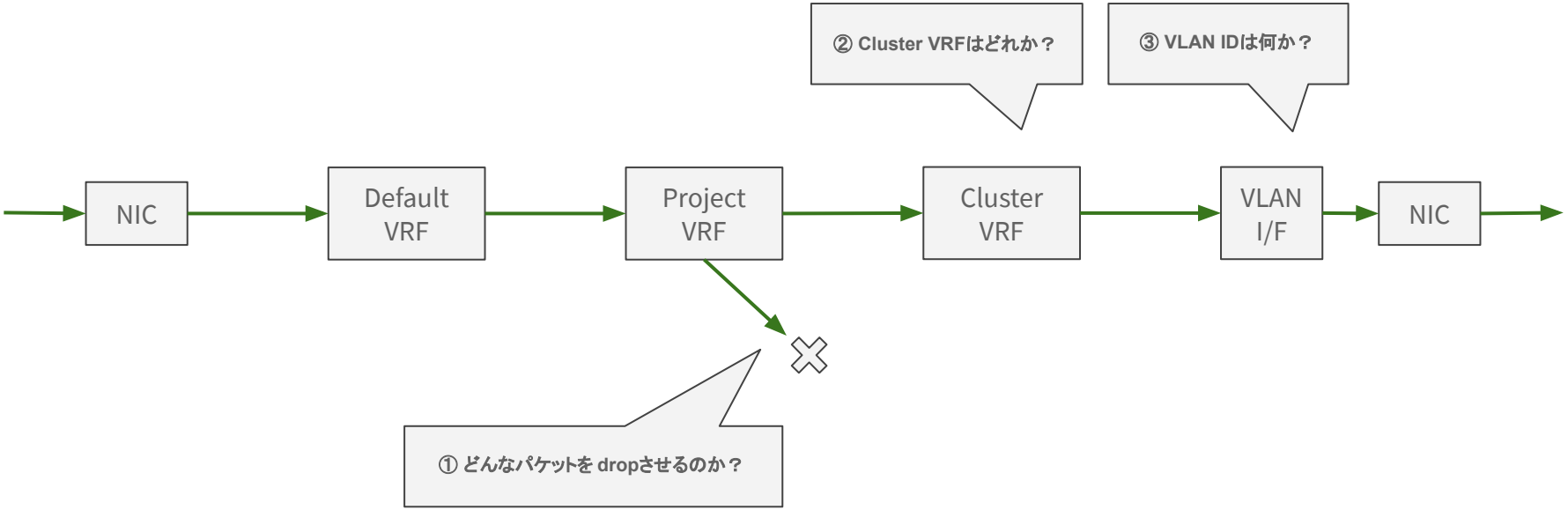
最適化の方針

SIDが決まれば通るべきVRFが決定する
SIDとVRFの関係は一度決まれば変更されない

SIDから通るべきVRFを事前に計算すればよい



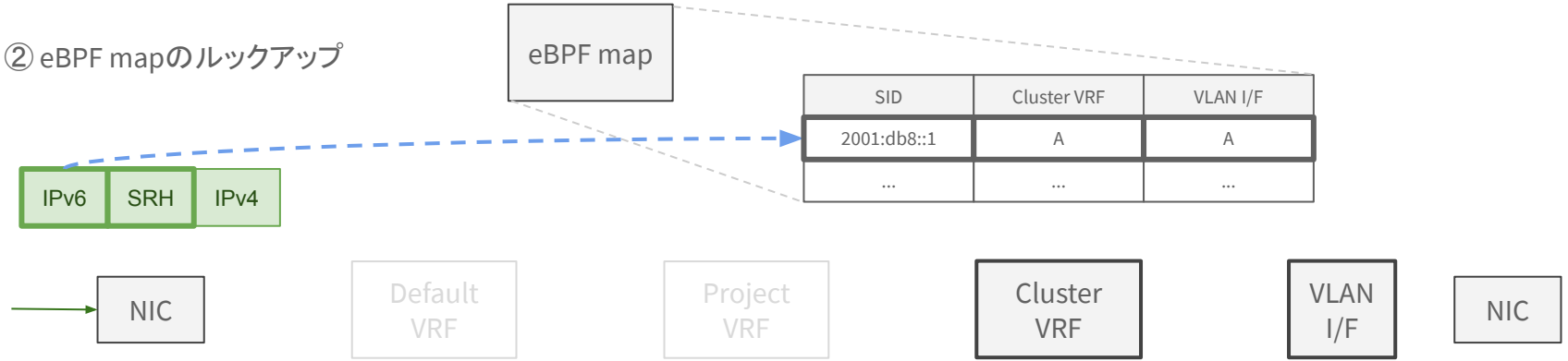
最適化の方針


② Cluster VRFはどれか？
③ VLAN IDは何か？
NIC
Default VRF
Project VRF
Cluster VRF
VLAN I/F
NIC
① どんなパケットを dropさせるのか？

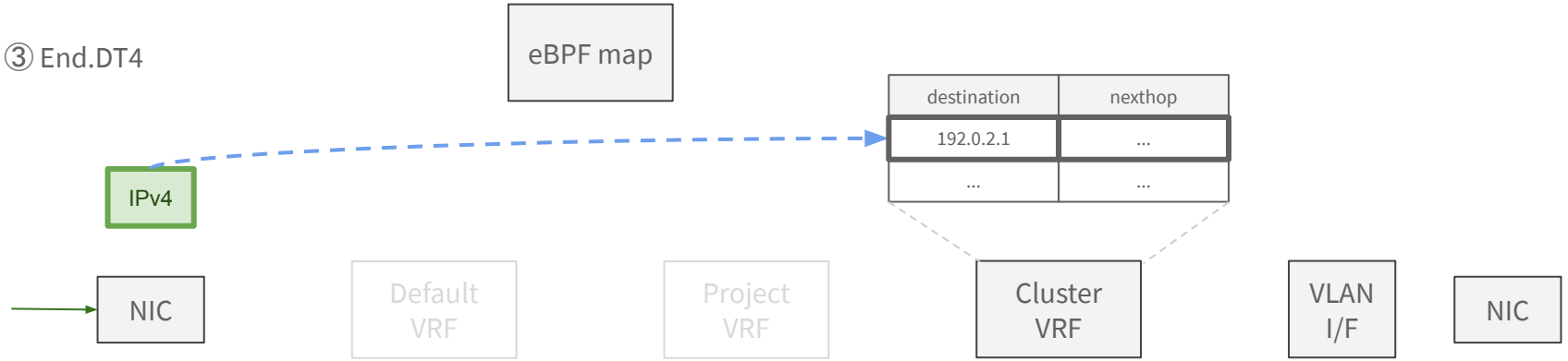
データプレーンの処理

① 処理すべきかチェック

データプレーンの処理

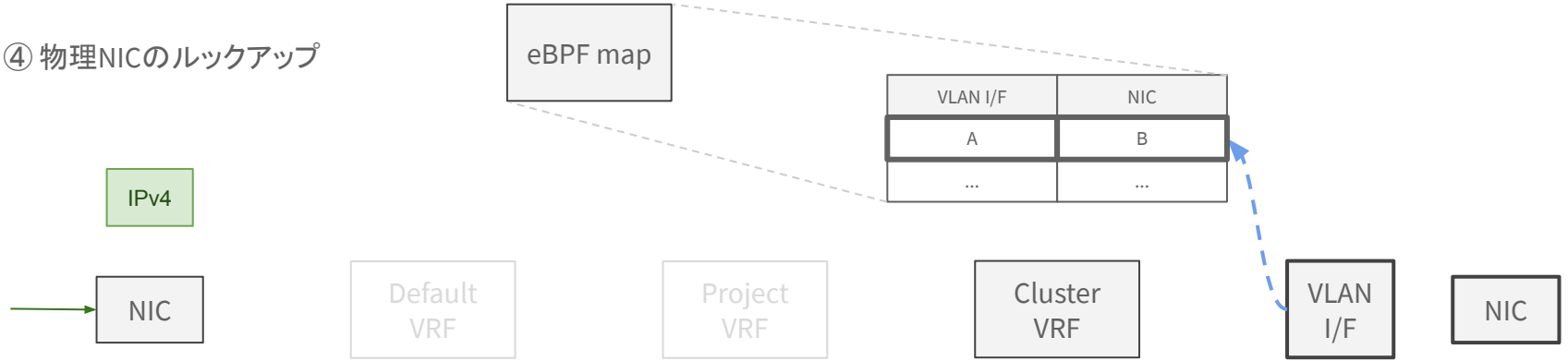

② eBPF mapのルックアップ
eBPF map
SID
Cluster VRF
VLAN I/F
2001:db8::1
A
A
...
...
...
IPv6
SRH
IPv4
NIC
Default VRF
Project VRF
Cluster VRF
VLAN I/F
NIC

データプレーンの処理


③ End.DT4
eBPF map
destination
nexthop
192.0.2.1
...
...
...
IPv4
NIC
Default VRF
Project VRF
Cluster VRF
VLAN I/F
NIC

データプレーンの処理

④ 物理NICのルックアップ

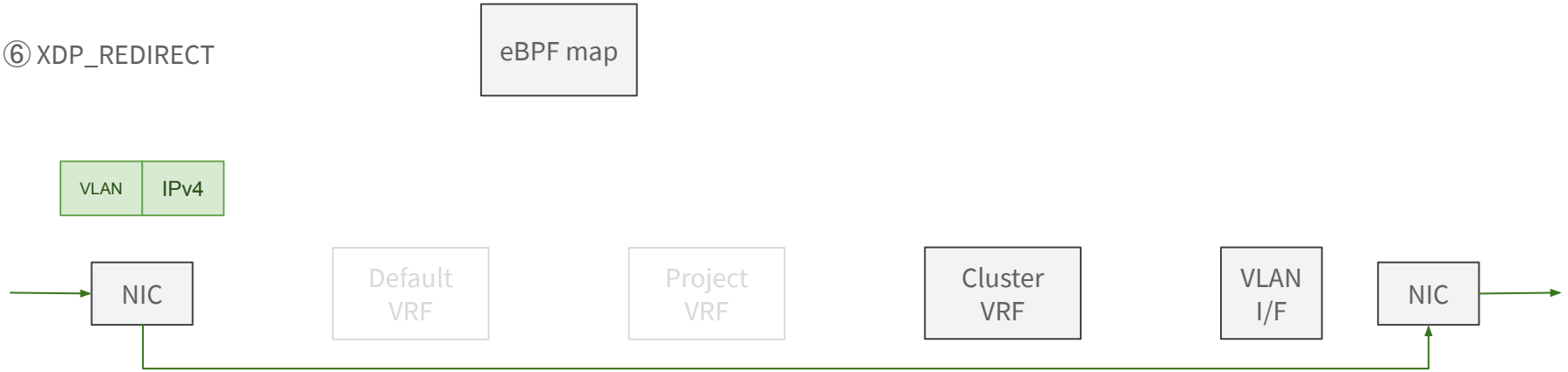


データプレーンの処理

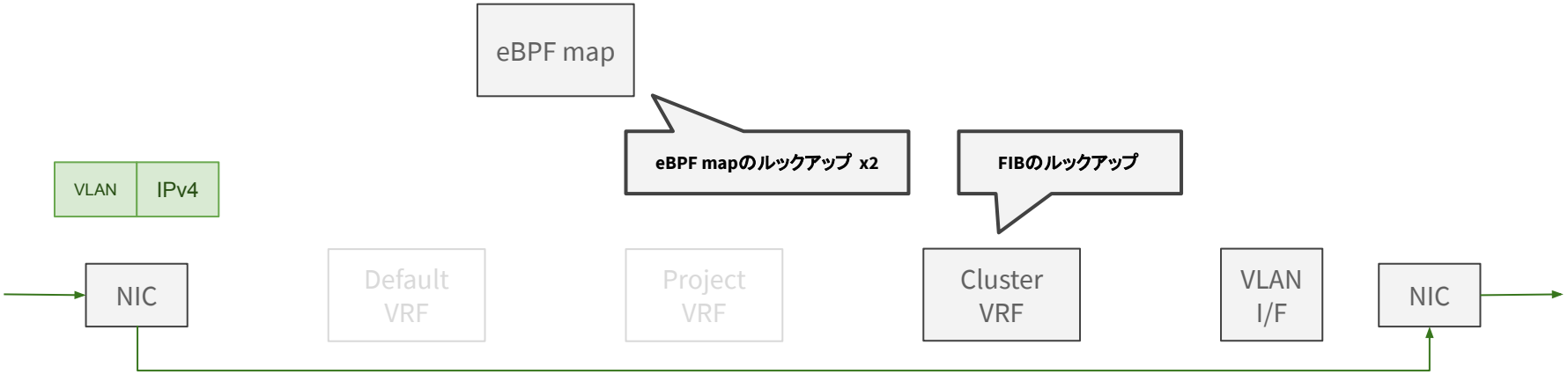
⑤ VLANヘッダの挿入

データプレーンの処理

⑥ XDP_REDIRECT



データプレーンの処理


eBPF map
eBPF mapのルックアップ x2
FIBのルックアップ
VLAN
IPv4
NIC
Default VRF
Project VRF
Cluster VRF
VLAN I/F
NIC

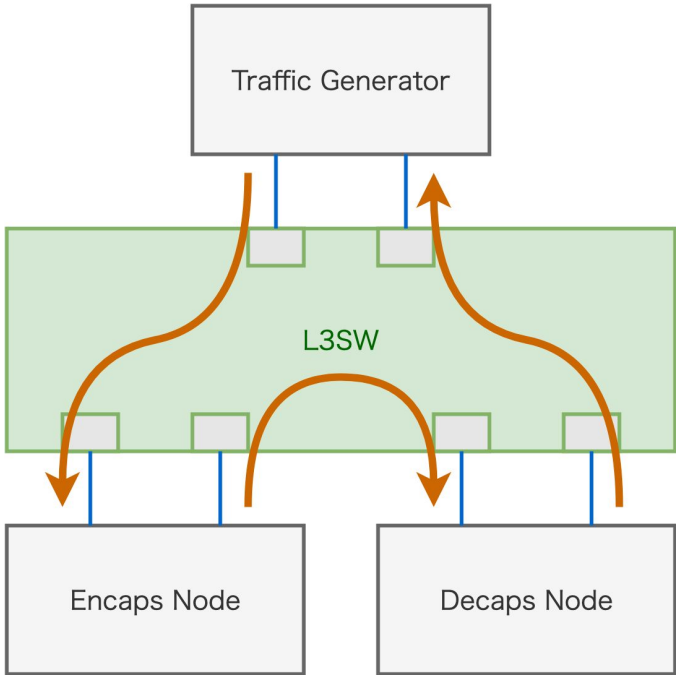
最適化の内容

- FIBのルックアップを減らすことが出来た
  - 5回 => 1回に減らすことができた
    - 代わりにeBPF mapを2回ルックアップする必要がある
- Project VRFを減らすことが出来た
  - VRFで頑張ってEnd.DT4していた部分が無くなったので余計な VRFが要らなくなった

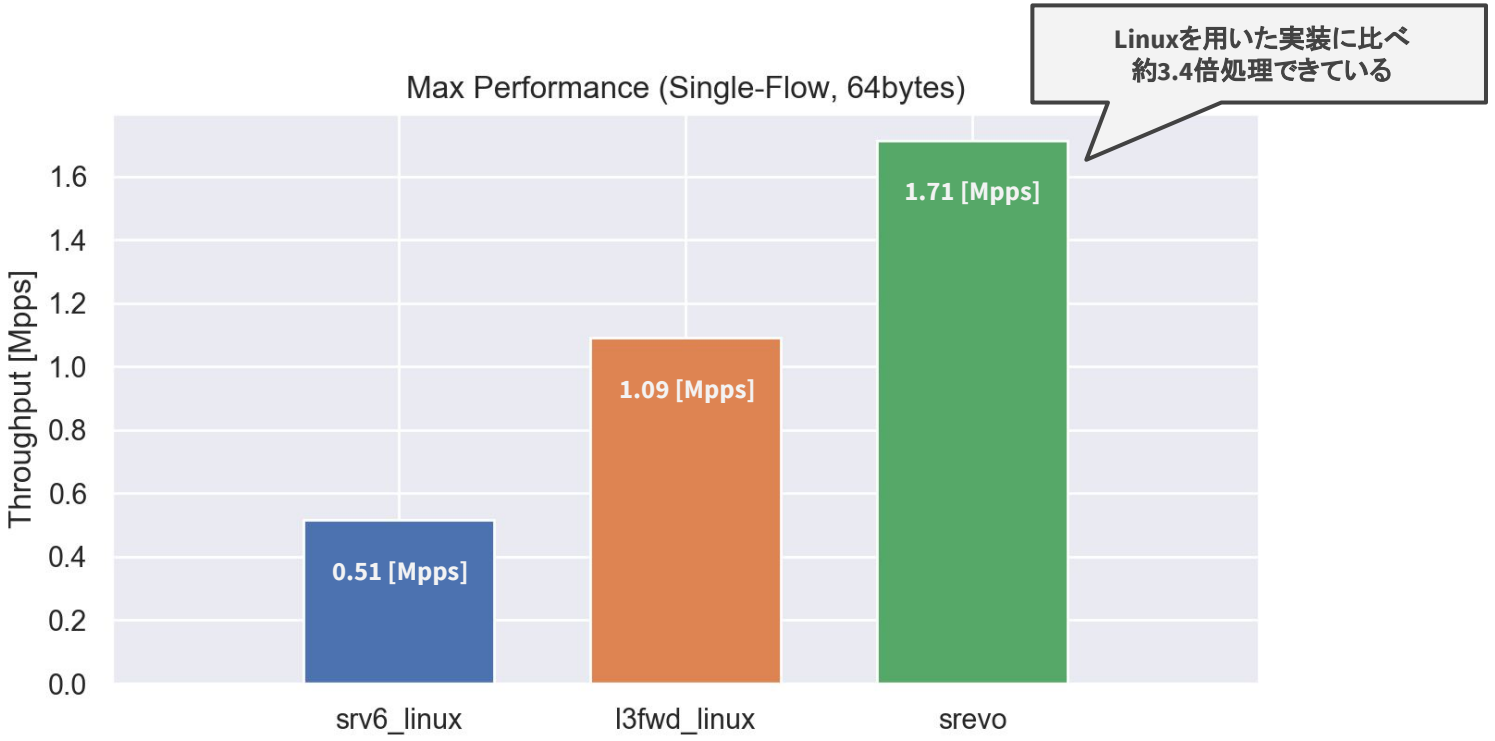
Performance

測定環境

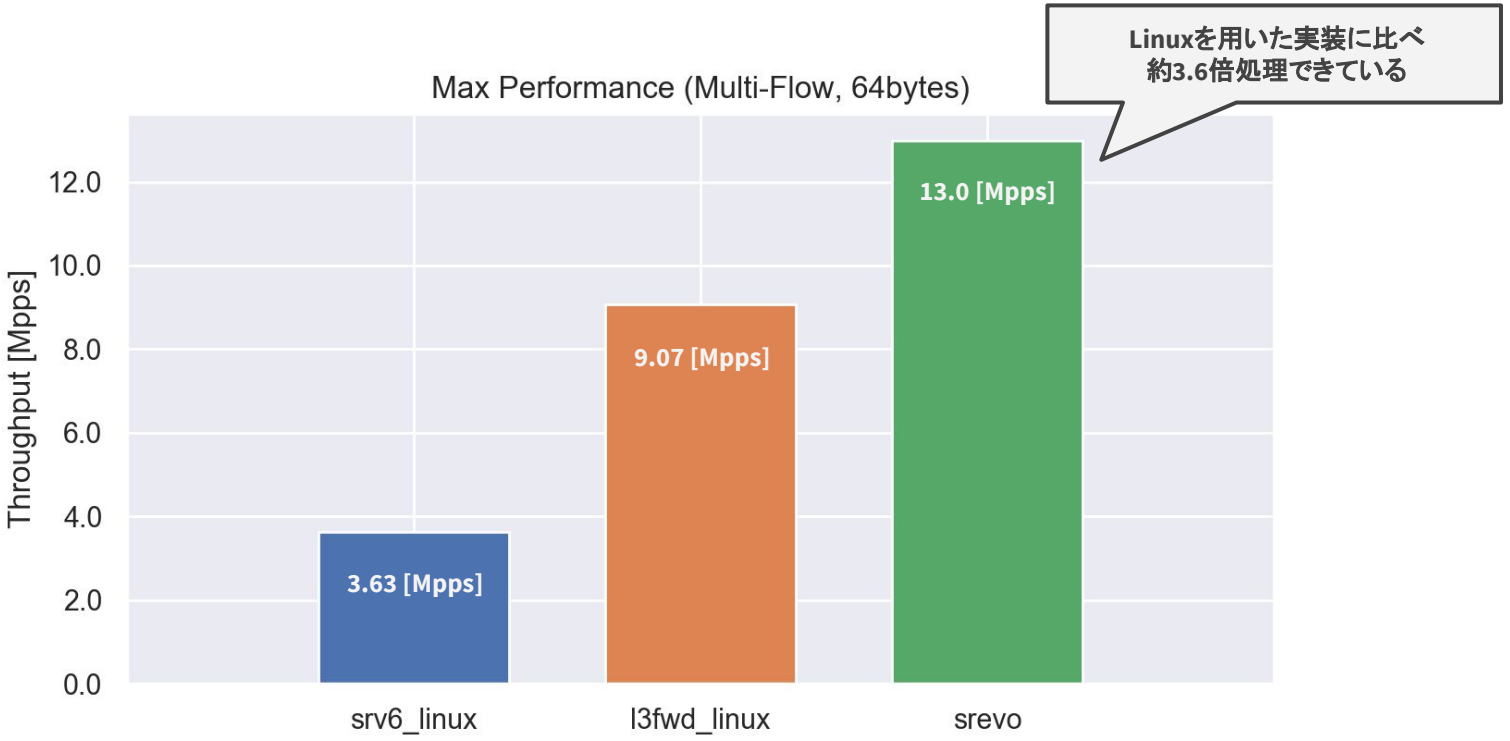
- Traffic Generator
  - Intel Xeon E5-2630 x2, 256GB
  - Intel XL710 40GbE 2port
  - Linux 3.10
  - TRex v2.50
- Encaps Node / Decaps Node
  - Intel Xeon Silver 4114 x2, 128GB
  - Mellanox MCX516A-CCAT 100GbE 2port
  - Linux 5.3 rc5
- L3SW
  - Mellanox SN2700
  - Cumulus Linux 3.5.3



測定結果


Linuxを用いた実装に比べ
約3.4倍処理できている
Max Performance (Single-Flow, 64bytes)
Throughput [Mpps]
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.51 [Mpps]
1.09 [Mpps]
1.71 [Mpps]
srv6_linux
l3fwd_linux
srevo

測定結果


Linuxを用いた実装に比べ
約3.6倍処理できている
Max Performance (Multi-Flow, 64bytes)
Throughput [Mpps]
12.0
10.0
8.0
6.0
4.0
2.0
0.0
3.63 [Mpps]
9.07 [Mpps]
13.0 [Mpps]
srv6_linux
l3fwd_linux
srevo

XDPを選択してよかったこと

- Linuxカーネルの恩恵を受けることができる
  - ARPの処理やFIBの管理はカーネルに依頼することができる
- 必要な機能だけ実装するのでシンプルに保つ
  - データプレーンのプログラム自体は 500行程度
  - 新しい機能が欲しくなれば実装すればよくなる
- 既存のコントロールプレーンを利用できる
  - 大規模な変更を加えることなく、今動いている SRv6の仕組みにそのまま乗っかれる
  - srevoが問題を起こした際のフォールバックとして既存の実装がそのまま利用できる

苦労した点

- デバッグがしづらい
  - tcpdump等のデバッグツールが使えないのでパケットが消えたときにすごく困る
- カーネルのすべての情報に触れるわけではない
  - FIBのルックアップをしてもVLAN I/Fの情報しか取れなく、物理 NICとの対応関係は自分で管理する必要がある

今後の課題

- デバッガブルなプログラムにする
  - 監視用のインターフェースの実装
  - デバッグモードの実装
- Service Function Chainingに対応する
  - 複数のSIDでencapsできるようにする
  - End.DT4以外のFunctionの実装（EndやEnd.Xなど）
- さらなる最適化
  - decaps / encapsの処理の高速化
  - ルールを格納するデータ構造の持ち方

議論したいポイント

- この取り組み（DCNにSRv6を用いることやXDPを用いた最適化）について
  どう思いましたか？

- SRv6を用いてどのような世界が実現できると嬉しいと思いますか？

- データプレーンのコントローラにどんな機能があると嬉しいと思いますか？（デバッグ機能など）