# 階層化コントローラアーキテクチャによる SDN における運用問題解決手法

中山 裕貴†、林 經正†、阿多 信吾‡
†株式会社ボスコ・テクノロジーズ
‡大阪市立大学

BOSCO Technologies

# SDN の導入意欲の向上

- 近年ネットワークオペレーションコストの削減を目指して SDN 導入が検討されている
  - 人手の介入が減ることによるヒューマンエラーの削減
  - ベンダ非依存な物理インフラを構築することによる運用管理の容易化
  - ネットワーク業務の簡素化

- SDN には多種多様の実装形態がある

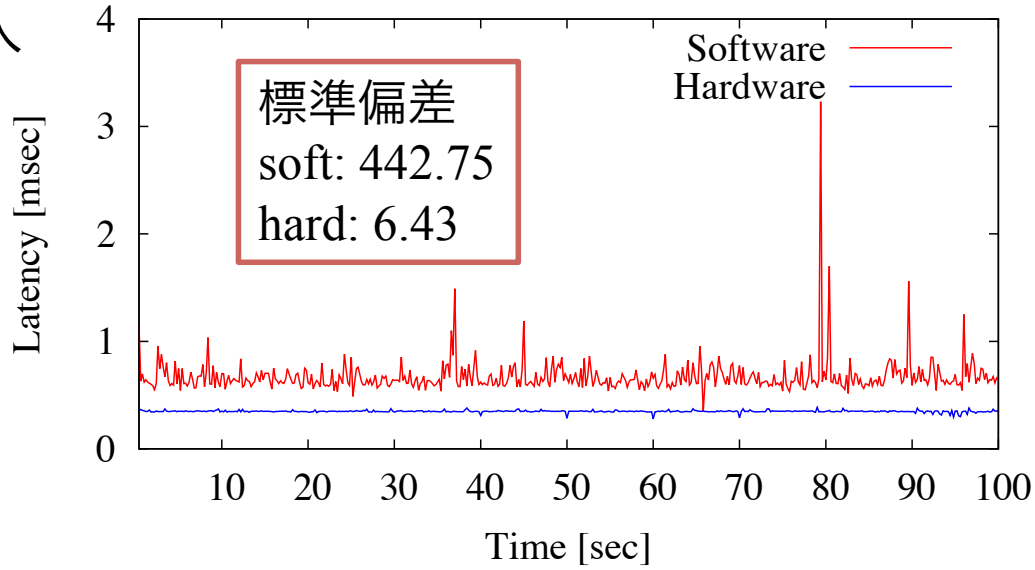
- OpenFlow はプロトコルが明確に存在し、多数のスイッチベンダが対応
  - 様々な場面において OpenFlow の導入検討が進む

# 導入した際の落とし穴

- OpenFlow を導入するも様々な問題が発生
  - パフォーマンス
    - フロー数の不足
      - 現在のスイッチでは 2 ~ 4k
      - 実際に需要は 200k (現在も毎月 1k ずつ増加)
    - packet-in 処理が重い
      - 秒間 300 packet-in しか処理できず、他はブロードキャスト
  - ベンダロックインの加速
    - コントローラ + OpS + スイッチのトータルソリューションとしてロックインさせる動向
    - OpenFlow は転送方法を指定することに特化したプロトコル
      - 実際は OVSDB や OFConfig, CLI も使用するなどスイッチの実装に強く依存するものを使用
      - optional の実装の有無

# これまでの解決策と問題点

- フロー数問題
  - OF スイッチに FPGA の導入
    - フロー数、CAPEX 増加
    - ベンダロックイン加速
  - ソフトウェアスイッチの導入
    - CAPEX 低下
    - 転送性能、精度が低い

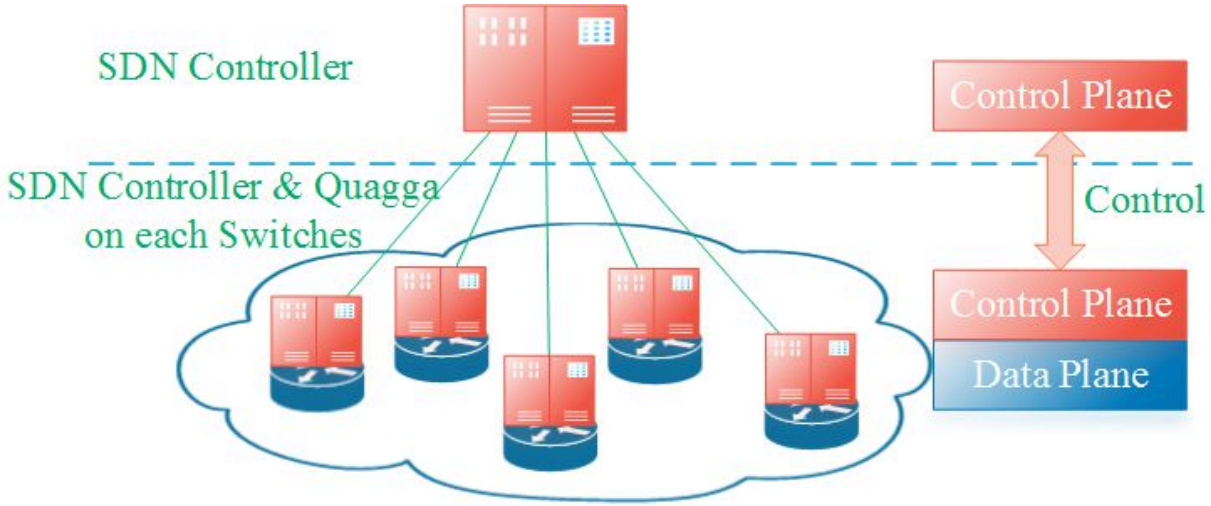



- packet-in の問題
  - 分散コントローラの導入
    - これまで多数の研究が多数存在
    - 実際の導入方針・方法は未定
- ベンダロックイン回避
  - 現在のところ解はなく、ベンダロックインが促進

# 目的

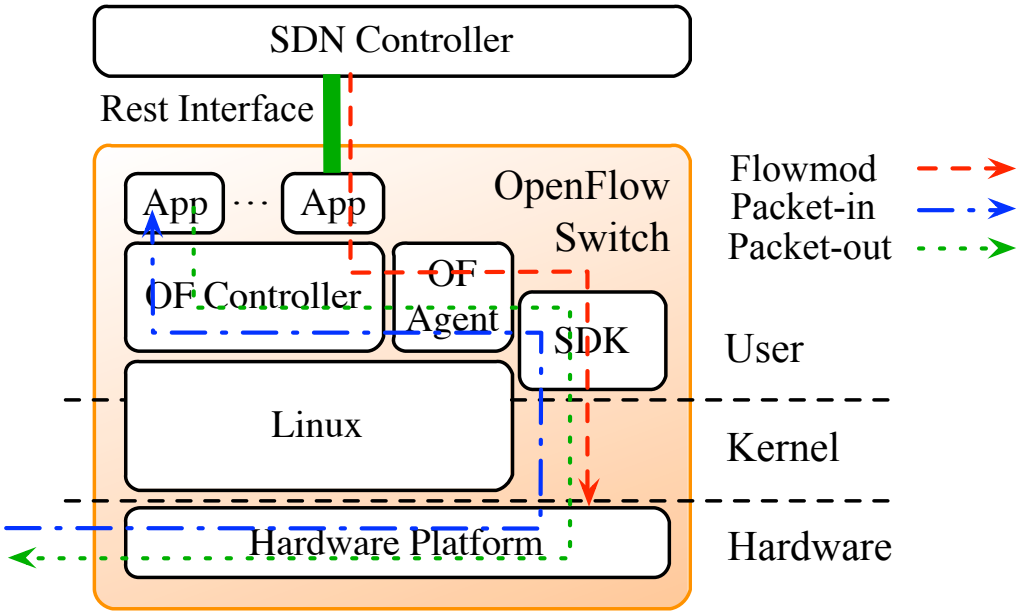
- OpenFlow 導入時の各課題の解決
  - フロー数問題を低価格で実現
  - 中央コントローラでの packet-in 処理を削減
  - ベンダロックインを回避したい

- スイッチ上でコントローラを動作させるモデルの提案
- 実際にどの程度の効果が得られるか検証



# ローカルコントローラの導入効果 (1)

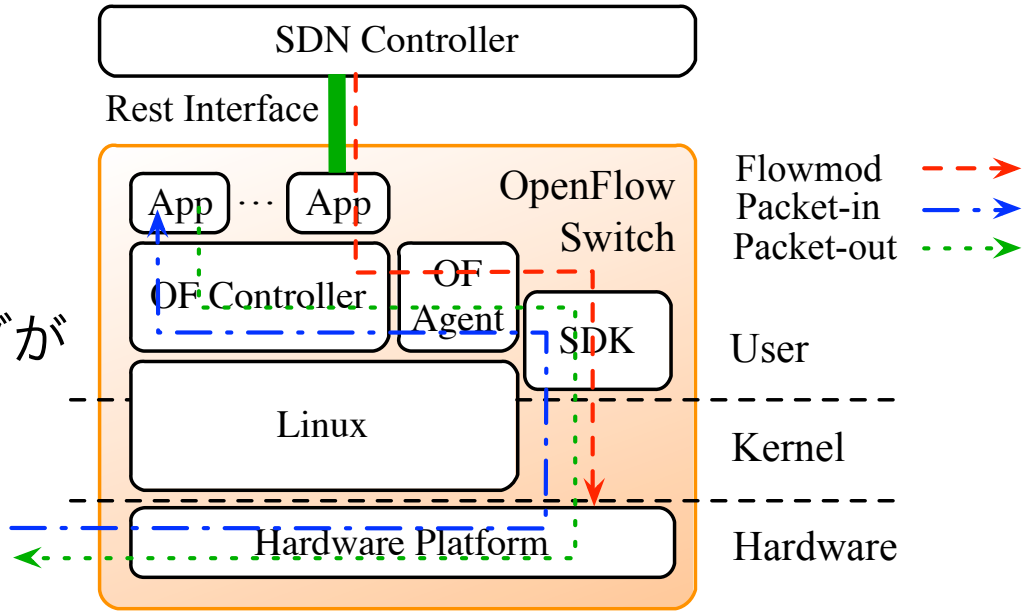
- コントローラのアプリを用いることによる任意の機能とインタフェースを実現可
  - ベンダロックインを回避
    - REST API を用いて ユーザ（業務）に インタフェースを任せる
  - ハードウェアでなく ソフトウェアで機能追加
    - ローカルコントローラで フローを保持し、ハードウェアには アクティブなフローのみを入れておく




ベンダロックイン回避、フロー数問題は解決可能

# ローカルコントローラの導入効果 (2)

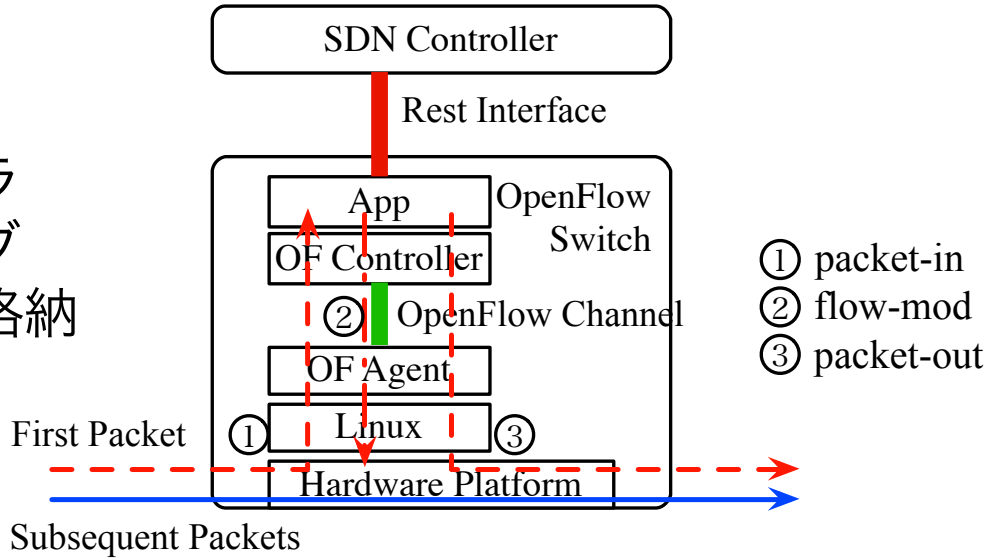
- スイッチとコントローラが1対1で packet-in 処理可能
  - packet-in 処理の 9 割以上 ARP, BGP などのプロトコル もしくは ping, Ethernet OAM などの OAM 機能
    - 本来スイッチ単体で完結すべき処理をわざわざ中央制御
    - OAM 機能の中には中央制御するべきものもある
  - packet-in 分の遅延を削減
    - 従来のスイッチの処理時間と比較し、packet-in の伝送分の遅延のオーダが大きい




packet-in 問題も解決可能

# ローカルコントローラアプリの実装

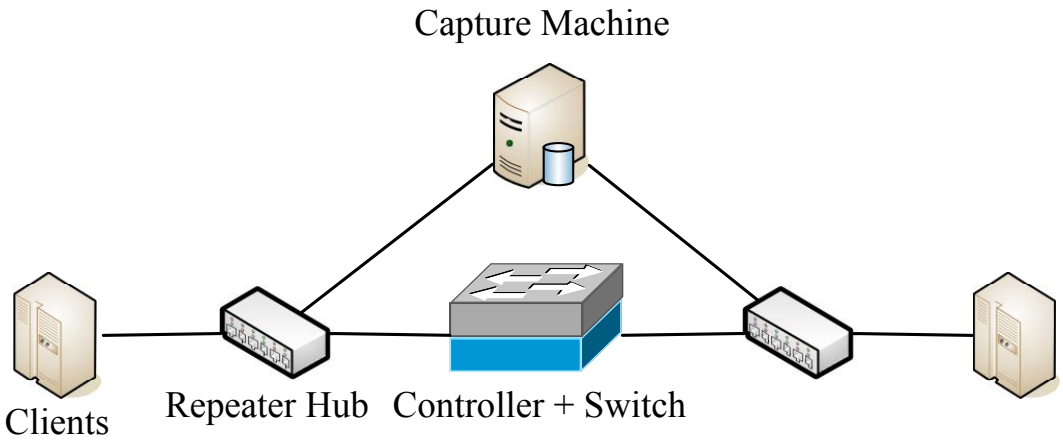
- フローテーブルの拡張
  - フローテーブルをコントローラアプリ内で保持し、アクティブなフローのみハードウェアに格納
  - 実際にアクティブなフローは 5% 程度




- フローをハードウェアに追加する際は idle_timeout を設定
- ハードウェアに存在するフローの統計情報は flow-removed メッセージに含まれる統計情報を利用
- ローカルコントローラのテーブルの一部を抽象化することで ARP 等の処理にも柔軟に対応

フローテーブルの拡張およびスイッチ単体で
プロトコル処理が完結

# 評価環境



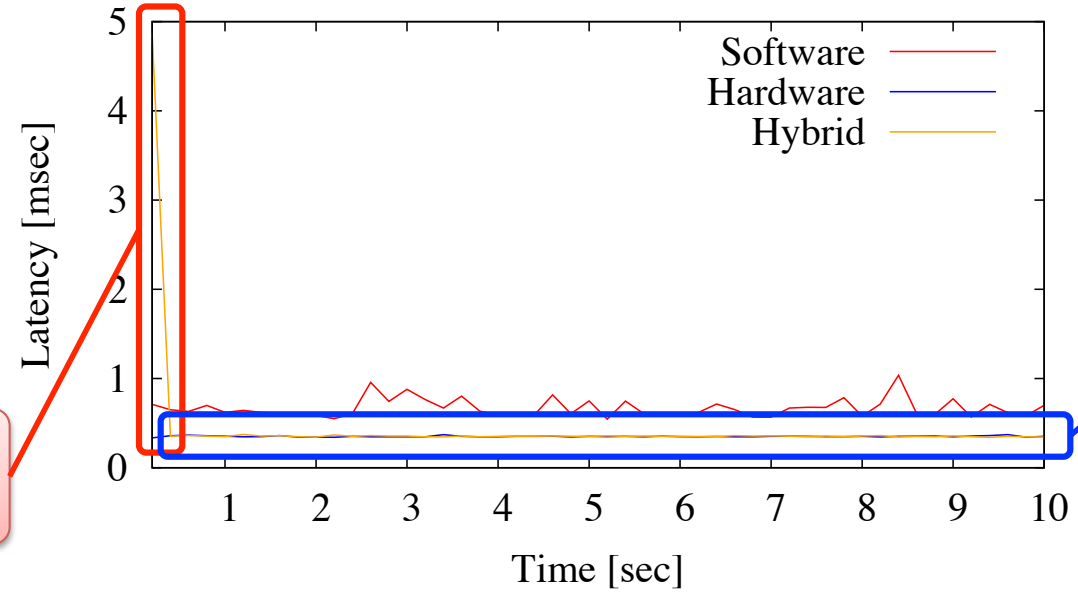

| | A | B | C |
|---|---|---|---|
| CPU (GHz) | 0.5 | 1.4 | 2.4 |
| アーキテクチャ | MIPS | X86 | X86 |
| RAM (GB) | 0.25 | 1G | 4G |

| | |
|---|---|
| コントローラ | RYU 3.18 |
| 使用言語 | Python |
| その他 | マルチスレッド無し<br>CPython, PyPy 等は使用せず |

- 性能評価
  - 既存の OpenFlow スイッチは CPU のスペックが低い
  - どの程度までならば実用的か評価
- 評価内容
  - First packet の遅延
  - 経路の切り替え時間

# First packet の遅延分析

- 100 フロー格納した場合の処理時間の評価




First packet のみ遅延

Second packet 以降はハードウェア処理

| | |
|---|---|
| パケット受信 => packet-in 送信 | 0.35 ms |
| packet-in 送信 => packet-in 受信 | 0.05 ms |
| packet-in 受信 => packet-out (flow-mod) 送信 | 3.4 ms |
| packet-out 送信 => packet-out 受信 | 0.05 ms |
| packet-out 受信 => パケット送出 | 0.35 ms |

ネットワークを介す場合、コントローラでの処理時間を超える

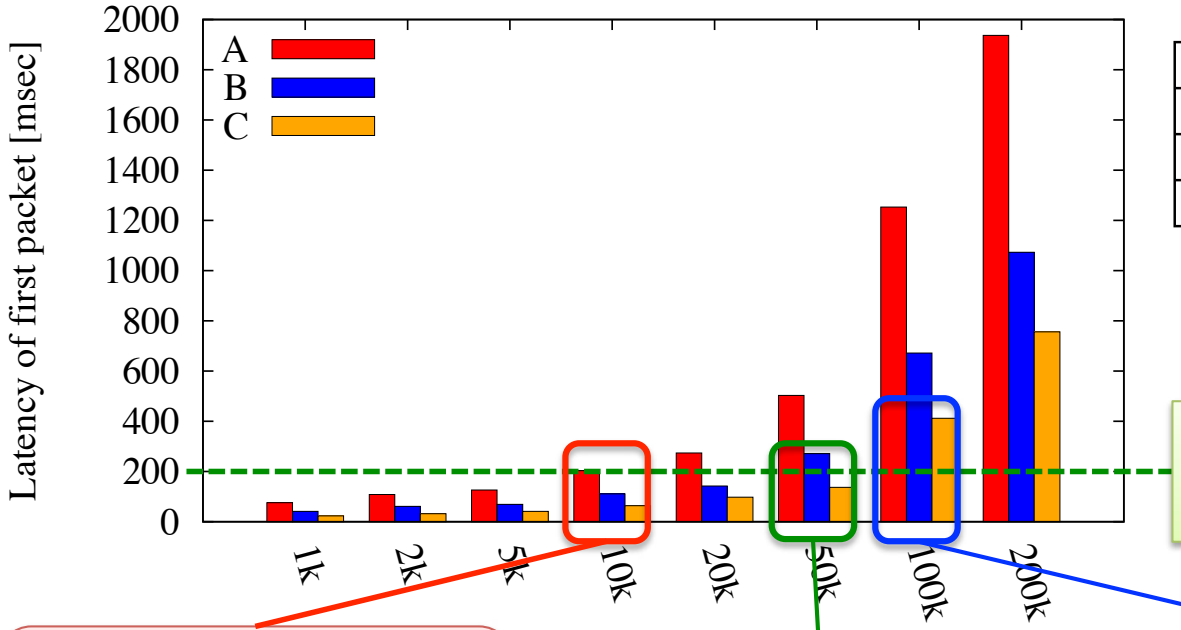
ローカルコントローラによるメリット



BOSCO Technologies

# フロー数の増加に伴う遅延の変化

- First packet の遅延をローカルコントローラ内のフロー数ごとに評価




| | A | B | C |
|---|---|---|---|
| CPU (GHz) | 0.5 | 1.4 | 2.4 |
| アーキテクチャ | MIPS | X86 | X86 |
| RAM (GB) | 0.25 | 1G | 4G |

リアルタイム切り替えの基準値 = 200 ms

A の場合、10k までならば 200 ms 以内で交換可能

C の場合、50k までならば 200 ms 以内で交換可能

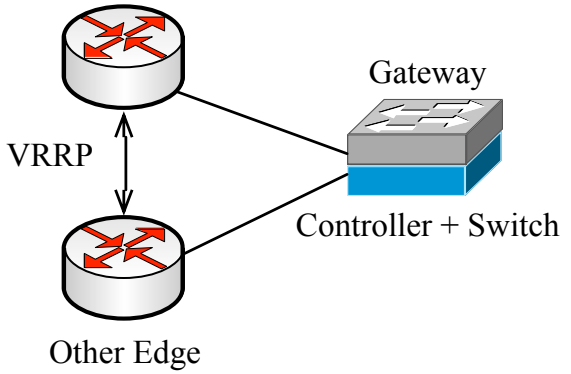
C の場合、100k までならば 400 ms 以内で交換可能

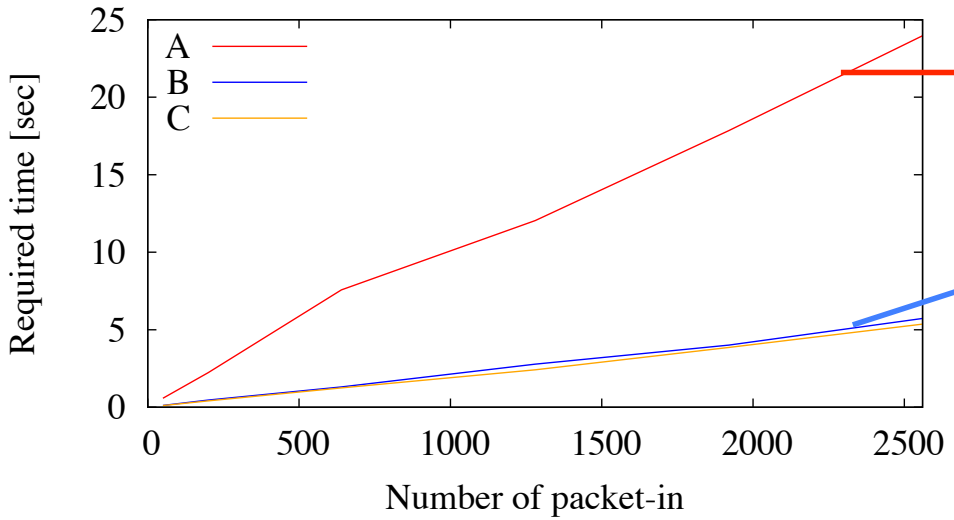


BOSCO Technologies

# GARP による経路切り替えの影響

- 冗長構成において VRRP で監視しあい、切り替えが生じると GARP を送出
  - 回線オーダの変更、計画工事、故障時
  - GARP を受信後 packet-in し、フローが切り替わるまでの時間を評価




コントローラでの処理時間がネック

スイッチでのフロー書き込みがネック

(参考) 中央のスイッチで制御しているシステムでは 300 packet-in / sec 程度しかコントローラで捌けない

ローカルコントローラではスイッチごとに 500 packet-in / sec で処理可能



BOSCO Technologies

# 近年のスイッチ事情

- OpenFlow スイッチは Linux をベースにし、その上で OVS （厳密にはその派生）を動作させているものが多い
  - OVS がハードの API を叩いている
- 任意のプログラムを実行できる環境も増え始めている
  - 任意の python スクリプトを実行できる環境
  - deb パッケージをインストール可能
  - 複雑なハードウェア API が OpenFlow に置き換えられたスイッチと考えこともできる
- 任意のプログラムを動作させるためにはそれなりのスペックが必要
  - CPU は 1 GHz 超え、RAM も 2GB 以上となっている

これまでの OpenFlow の課題をソフトウェアで十分解決可能

# まとめと今後の課題

- まとめ
  - OpenFlow 導入時の課題を解決しつつ、任意の機能を実行できる特徴を活かす仕組みを検討
    - フロー数問題を低価格で実現
    - 中央コントローラでの packet-in 処理を削減
    - ベンダロックインを回避したい
  - 実際にどの程度の効果が得られるか検証
    - 200ms 以内の切り替えであれば 50k 程度のフローは容易に実現
    - 中央コントローラへの packet-in の 9 割程度はローカルで処理可能
  - スイッチの動向から本実装は容易に実現可能
- 今後の課題
  - テーブル検索時の処理を最適化し、パフォーマンスを向上