Susanow:環境に対して自動最適化する高性能通信基盤

Hiroki SHIROKURA @slankdev slank.dev@gmail.com

powered by IPA-MITOU-program

Introduction

城倉 弘樹 (SHIROKURA Hiroki) aka slankdev

- セキュリティキャンプ 2015~
- アルバイト
  - Cybozu-Lab 「拡張可能なパケット解析ライブラリ」
  - Cybozu-Lab 「高性能 TCP/IP ネットワークスタック」
  - IIJ 研究所 「高性能パケット処理」(お休み中)

未踏事業 「環境に対して自動で最適化する高性能通信基盤」

プロジェクト概要

「汎用サーバを用いた高性能で超動的な NFV の実現」

draft.susanow.dpdk.ninja

- D2(Dynamic Thread Optimizaion) という技術を開発
- D2 を用いた動的な NFV 基盤開発
- 開発中の NFV 基盤上で動く VNF (開発中)
- NFV のいくつか空想を実現

Background

- SDN/NFV の未来予想図
- DPDK について

Network Function Virtialization

- ネットワーク機能を仮想化
- CAPEX/OPEX 低減
- 迅速なサービス変形

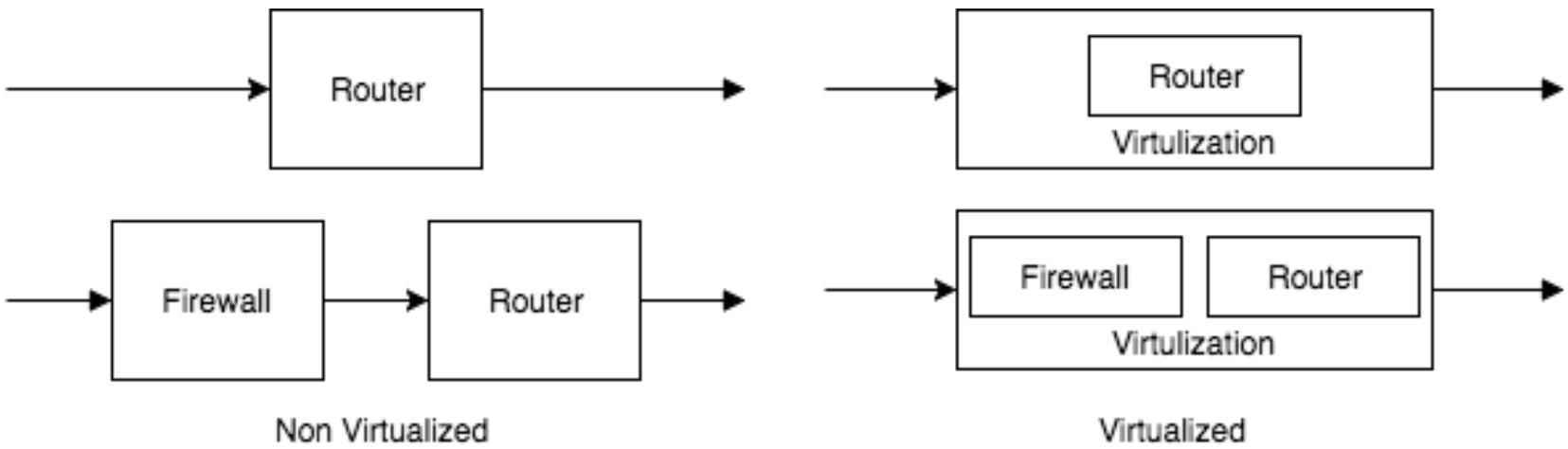


Figure 1: NFV

Service Function Chaining

- ネットワーク機能を細かく考える
- NFV の迅速性を利用し, 素早いサービス変形を柔軟に
- 現状は様々な方法で実現中 (Openflow など)
- プロトコルとしても標準化中

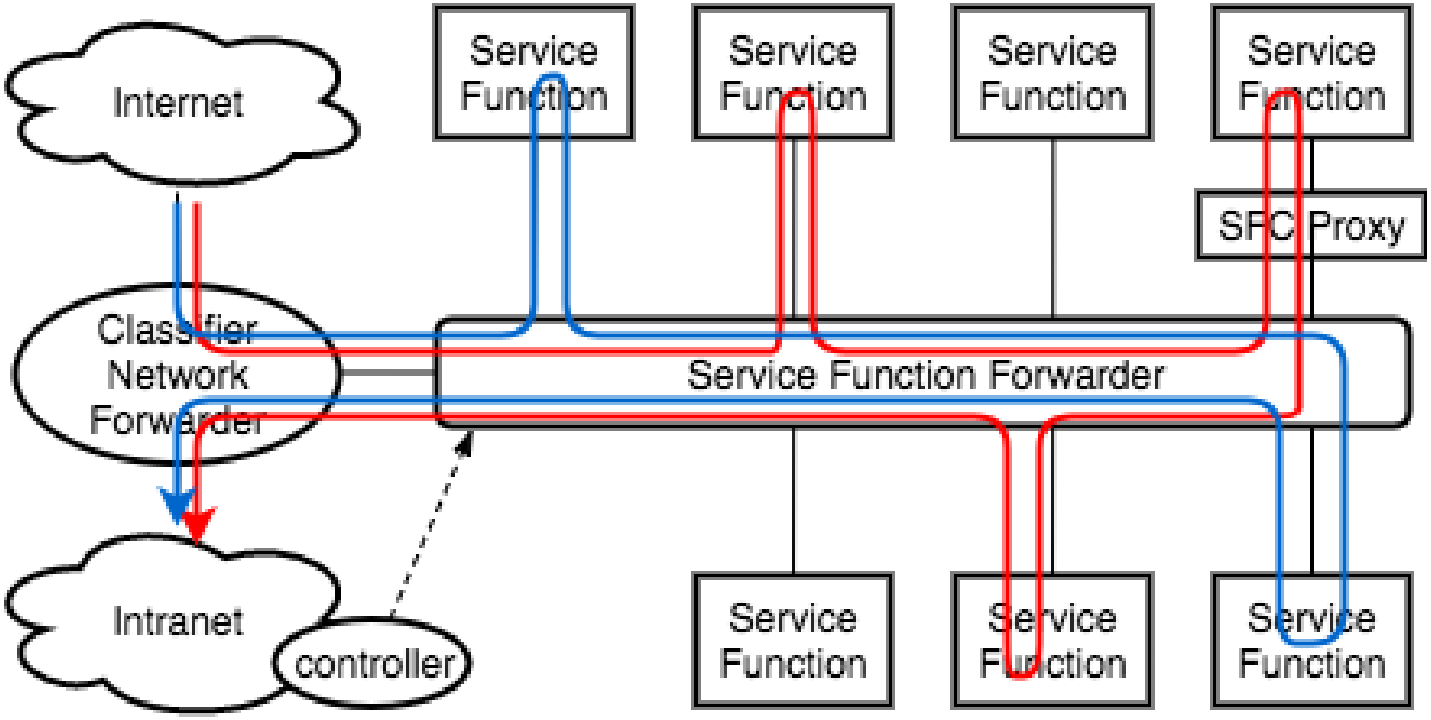


Figure 2: Service Function Chaining

動的なネットワーク変形の例

- DoS を検知したタイミングで新たに NF をデプロイ
- 必要に応じてその場その場でネットワークの機能をつなぎ合わせる
- FW 等はルールによっても必要な計算資源の量が違う.
- 最低限のリソースで最大限のパフォーマンス

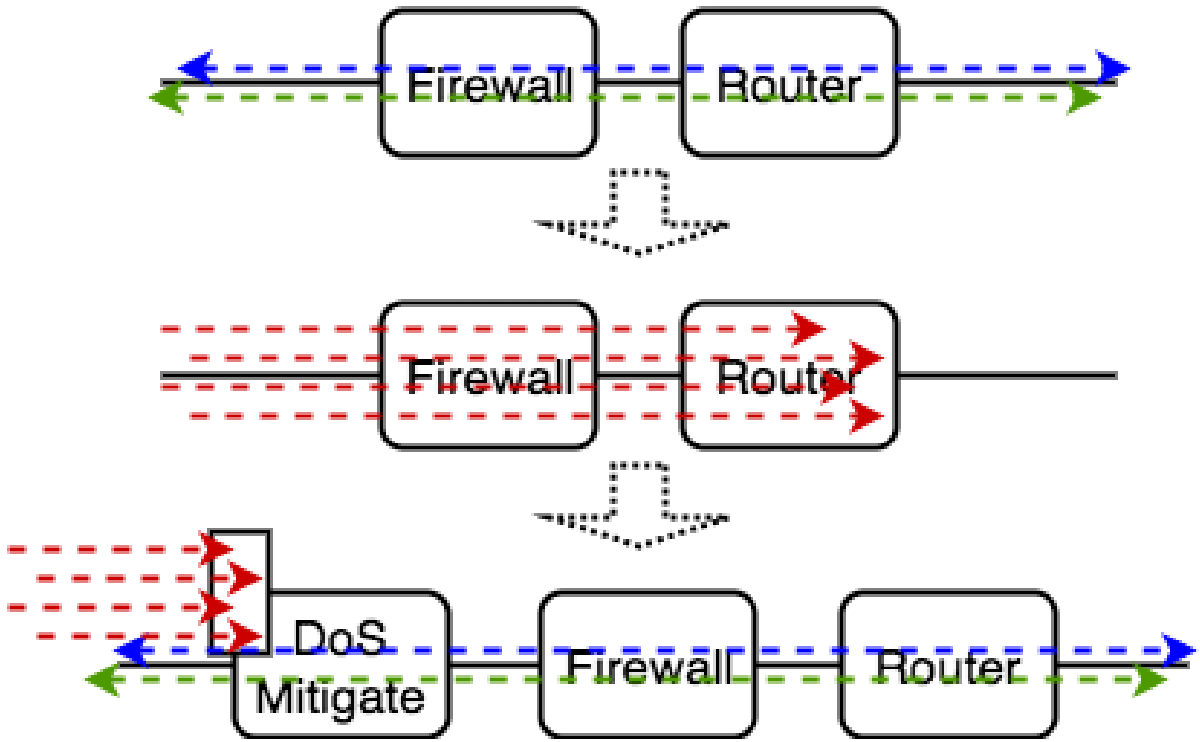


Figure 3: 状況に応じてネットワークを変形

DPDKによりIAサーバで高性能通信が可能

- 100Gクラスのトラフィックもパケットフォワード可能
- 現状のボトルネックは経路検索などのアルゴリズム
- 4つの特徴により実現

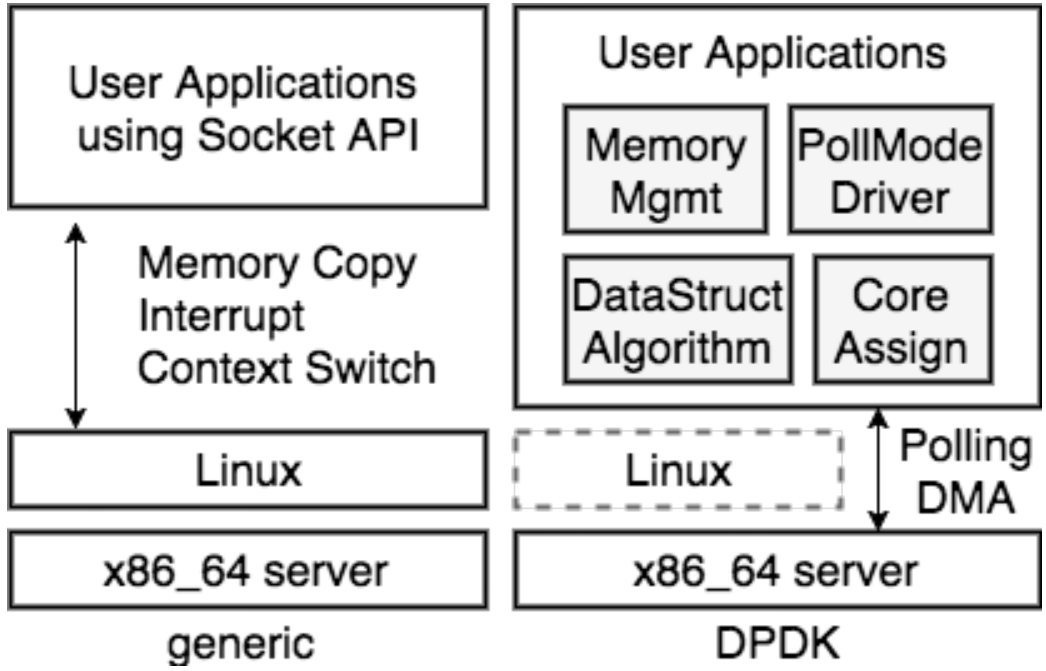


Figure 4: DPDK architecture

DPDK: 課題

- 高い開発コスト: コンピュータ理論に対する精通
- スレッド多重化率などのカリカリチューニング
- VM 環境でのオーバヘッド: 仮装 NIC のメモリコピー

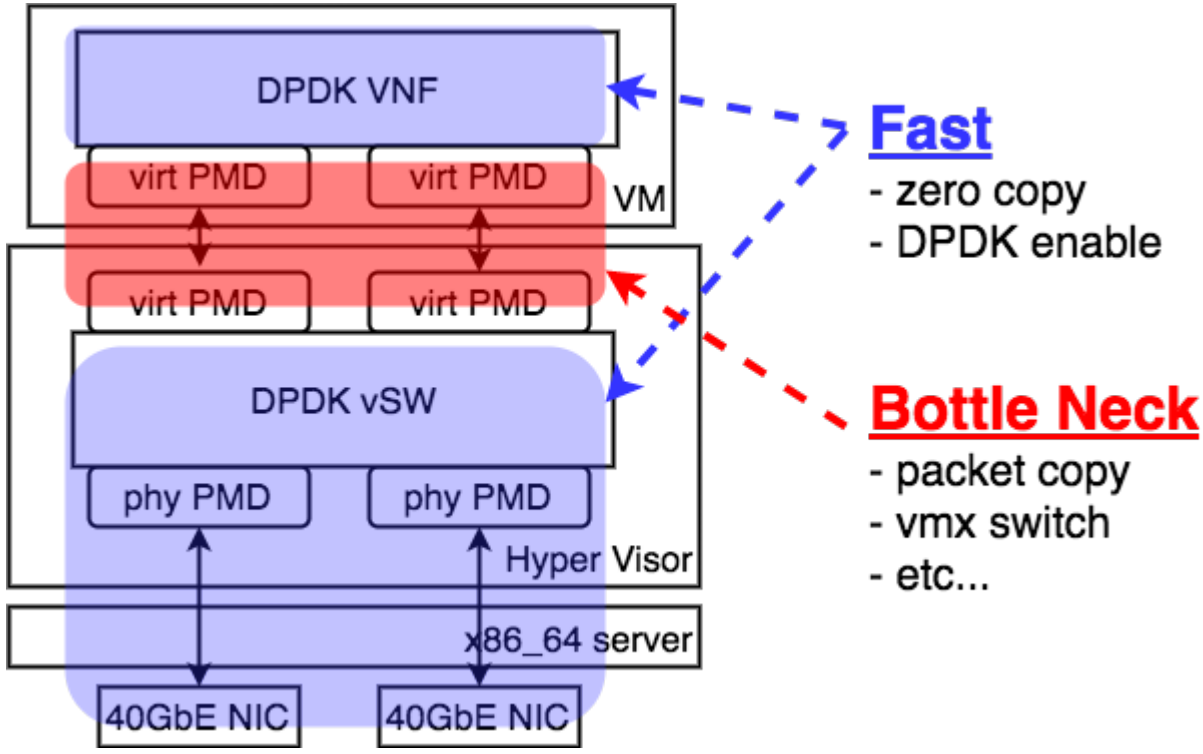


Figure 5: VM Overhead

Background: まとめ

現状とその課題

- NFV によりネットワーク制御はより動的に
- DPDK を用いることで高性能な NF を実装可能
- しかし開発コストがまだ高い
- 特定環境に最適化されている.

提案

- ssn-NFVi: nonVM NFV 基盤
- D2: DPDK VNF のスレッドチューニングの自動化
- ssn-NFVi 上で動作するし D2 で自動最適化を行う VNF (開発中)

Susanow Architecture

- ssn-NFVi: nonVM な NFVi
  - ポート管理やコアの管理をまとめて行う
  - VNF のデプロイインターフェース
- D2engine: 動的スレッド最適化技術

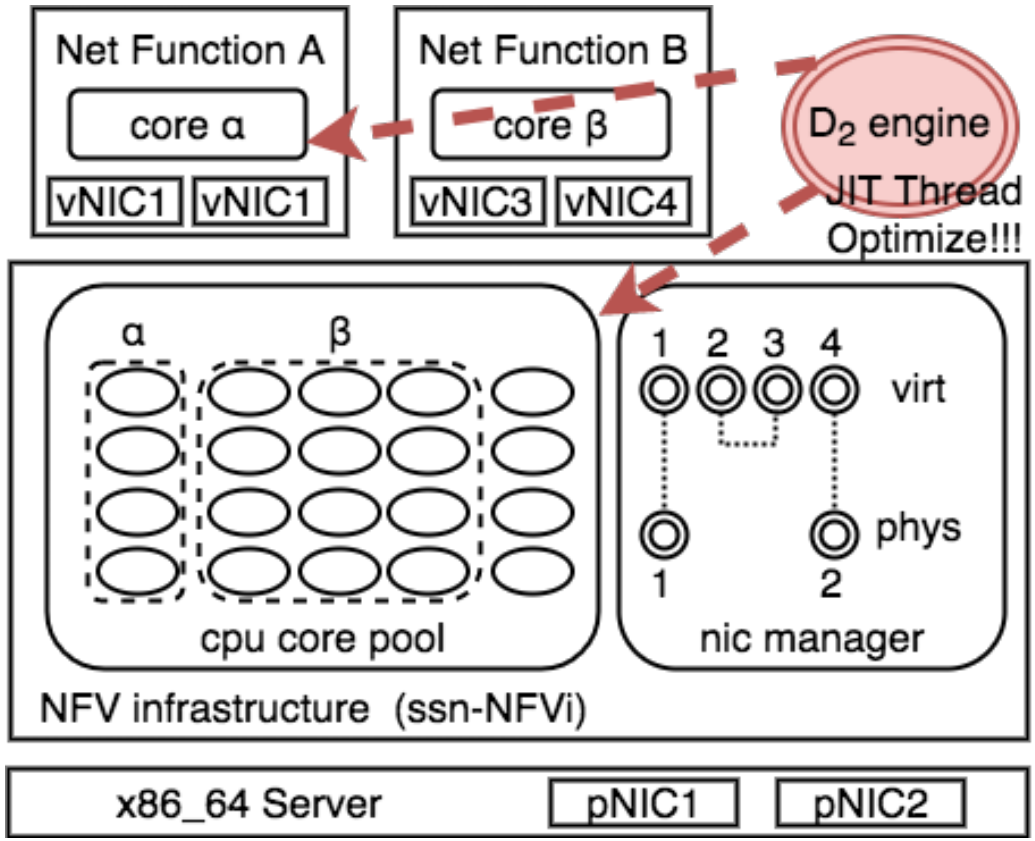


Figure 6: susanow nfvi

D2: Dynamic Thread Optimization

- Dynamic Thread Optimization -> DTO -> D2
- スレッド最適化を動的に行う技術のこと
- D2-API を用いて VNF を実装することで利用可能
- VNF 単体を動的に最適化する
- VNF 数を増やしてサービスを最適化ではない

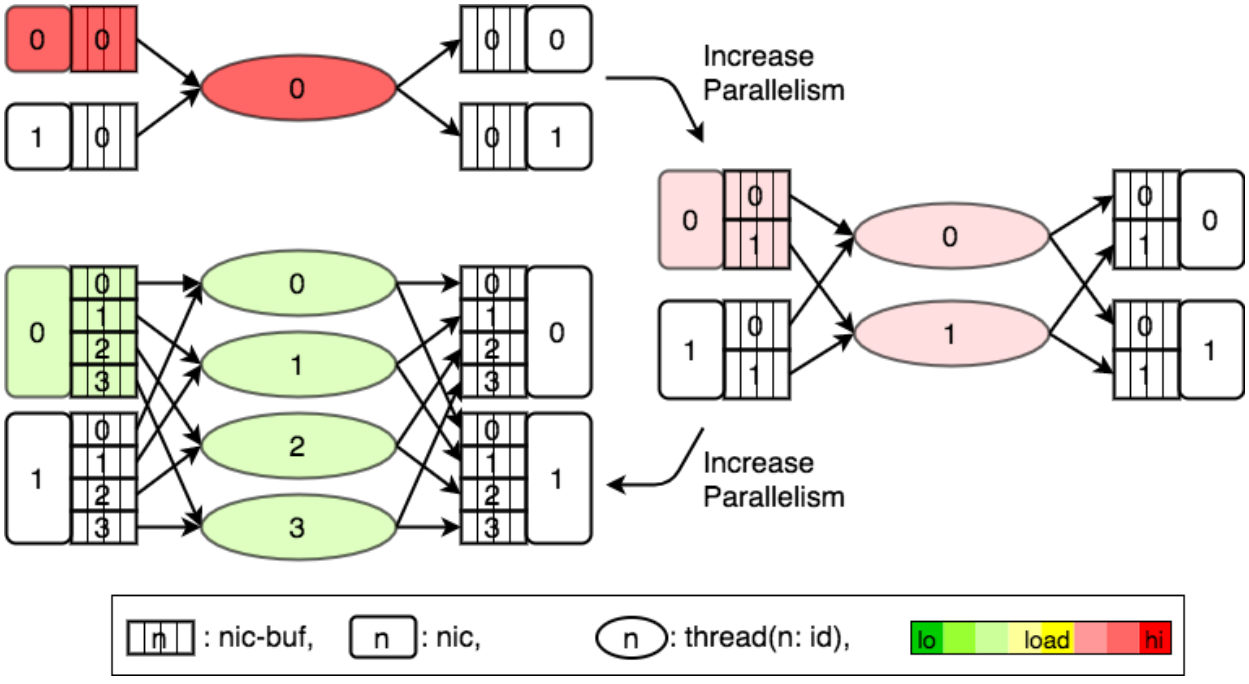
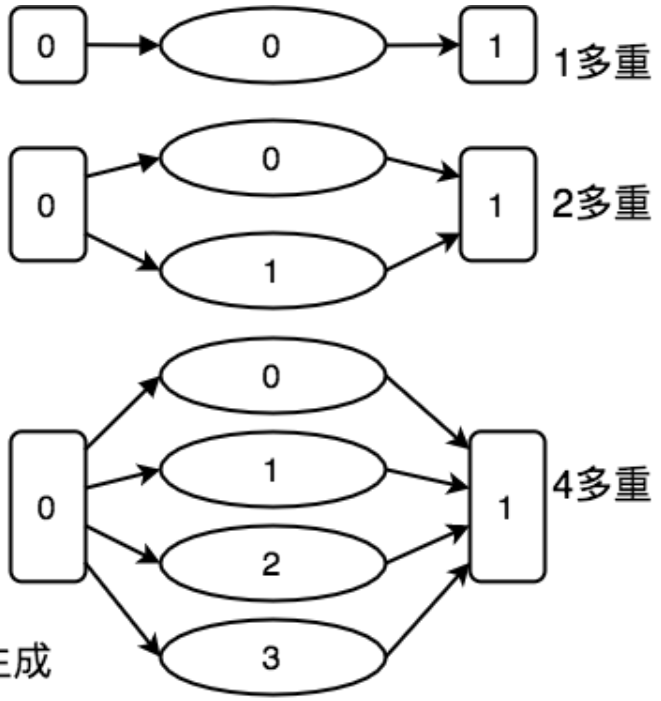


Figure 7: D2 Optimize Flow

D2: Dynamic Thread Optimization

- vSwitch やルータなどの L2/L3-NF から アプリケーションデータを扱う DPI Firewall まで幅広く対応可能
- パケットを受け取ったあとの処理は VNF 開発者が記述
- D2 はその処理を効率的に多重化が可能

```
void thread() {
  while (true) {
    npkts = rx_burst(0, mbufs, 32);
    for (i=0; i<npkts; i++) {
      process_packet(mbufs[i]);
      tx_burst(1, &mbufs[i], 1);
    }
  }
}
```



(n): thread, [n]: port (n:id)

開発者の書いた
ロジックからVNFを生成

Figure 8: D2 利用の流れ

D2: 最適化の流れ

- Thread: 先ほどのプログラムが動く (コアに固定される)
- Accessor: スレッドとポートのアクセスを仲介
- Nic-buf: NIC の Multiqueue

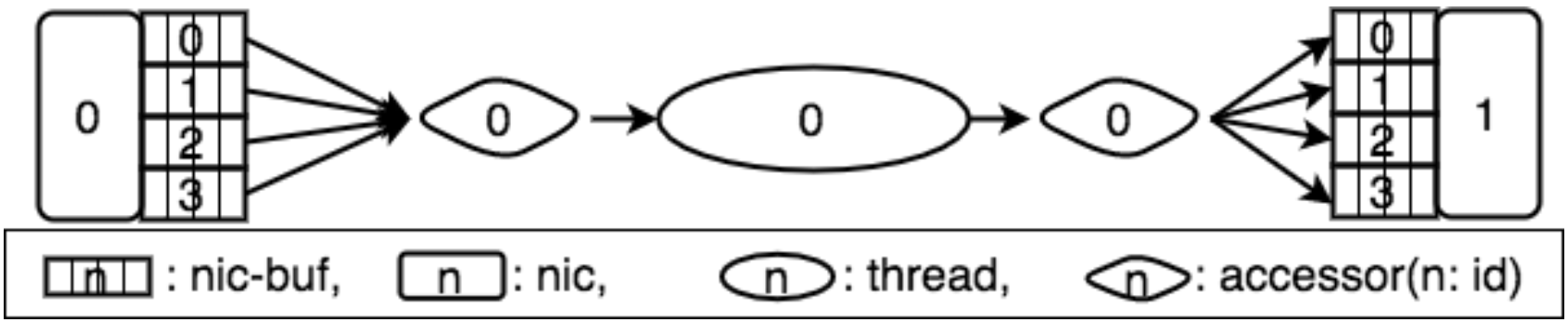


Figure 9: 1 多重

D2: 最適化の流れ

- 新たにスレッドを生成
- 枠内数字は id である

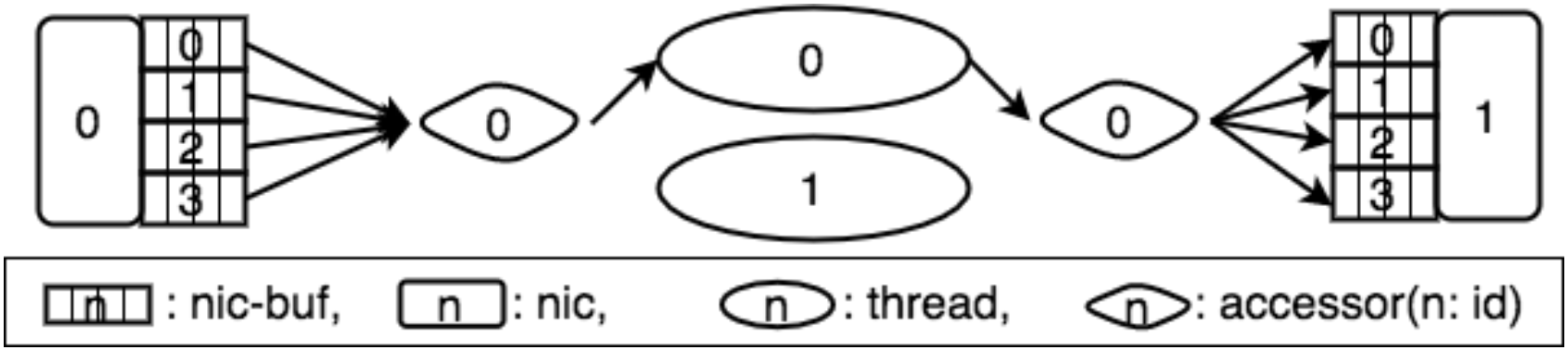


Figure 10: 1 多重

D2: 最適化の流れ

- ポートとスレッドでネゴシエーション
- ポートに対していくつのスレッドからのアクセスがあるかを確認する

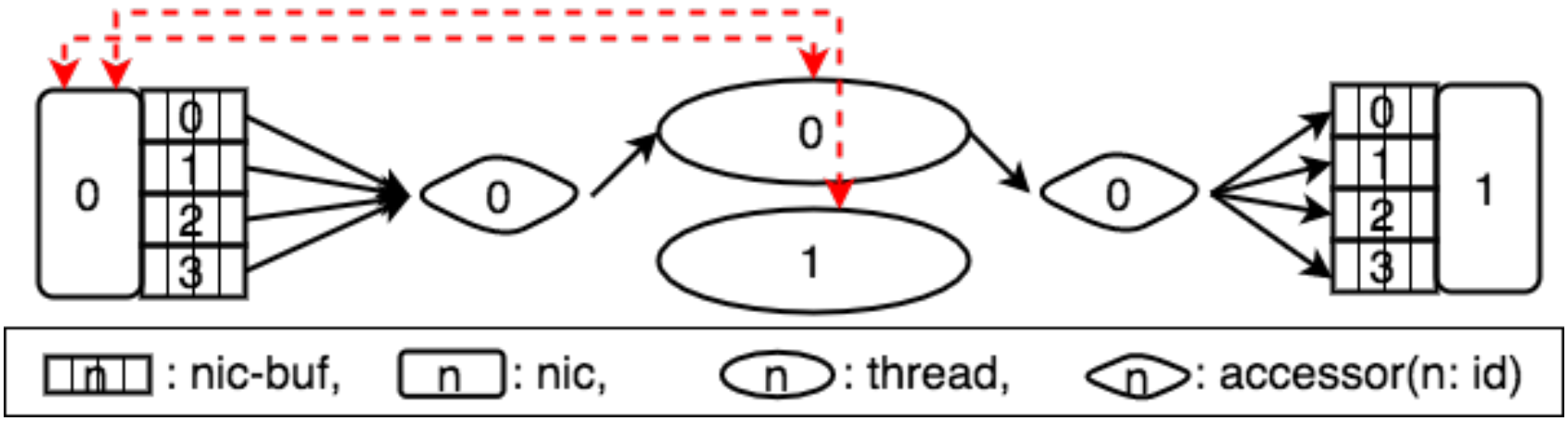


Figure 11: 1 多重

D2: 最適化の流れ

- ポートにアクセスするスレッドの数に合わせて Accessor を再構成

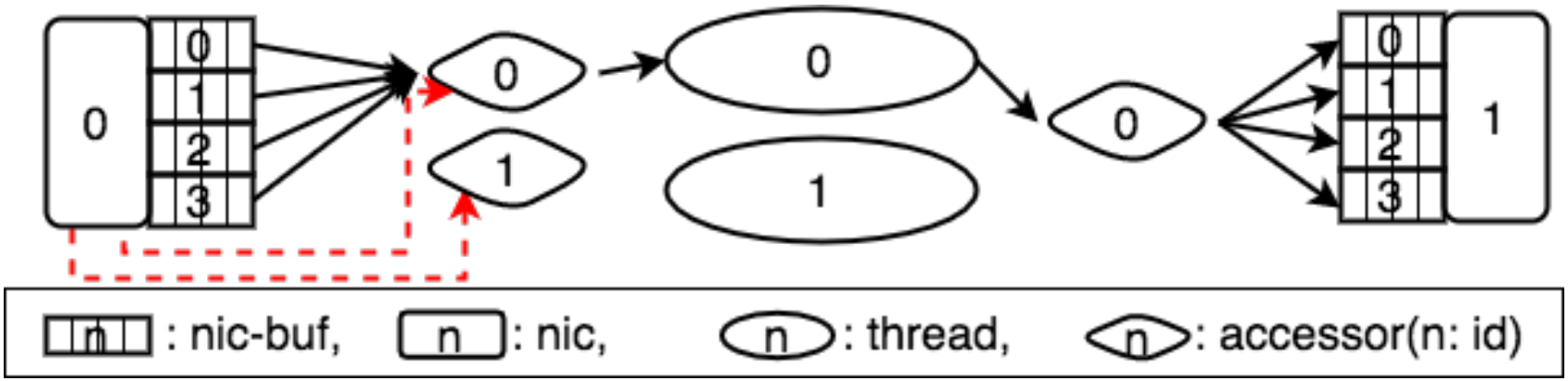


Figure 12: 1 多重

D2: 最適化の流れ

- Accessor とスレッドを再起動する
- これらの手順でスレッドの多重化を行う

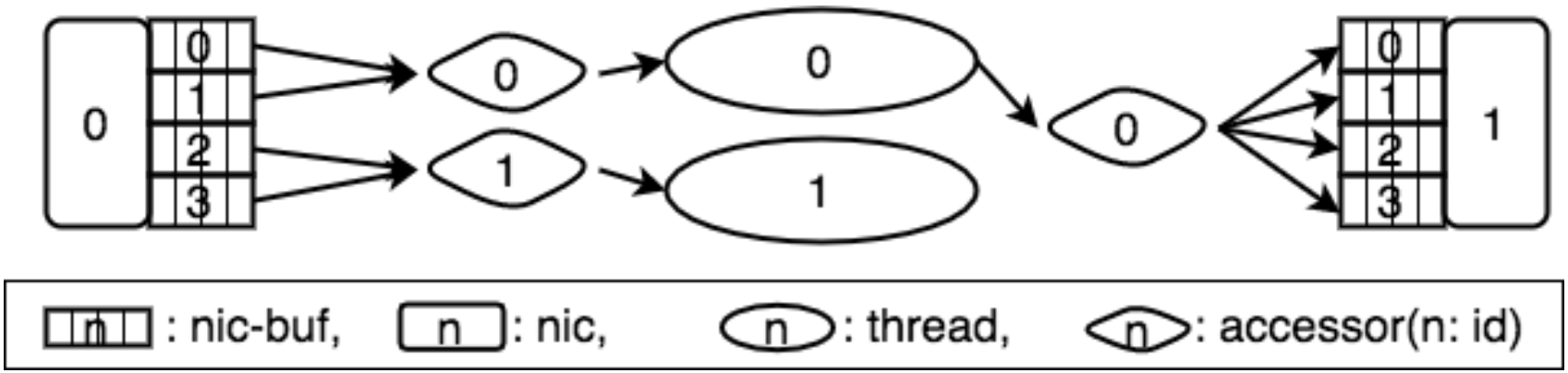


Figure 13: 1 多重

D2: 最適化の流れ

- 同様の手順で多重化を進めることで性能向上が可能

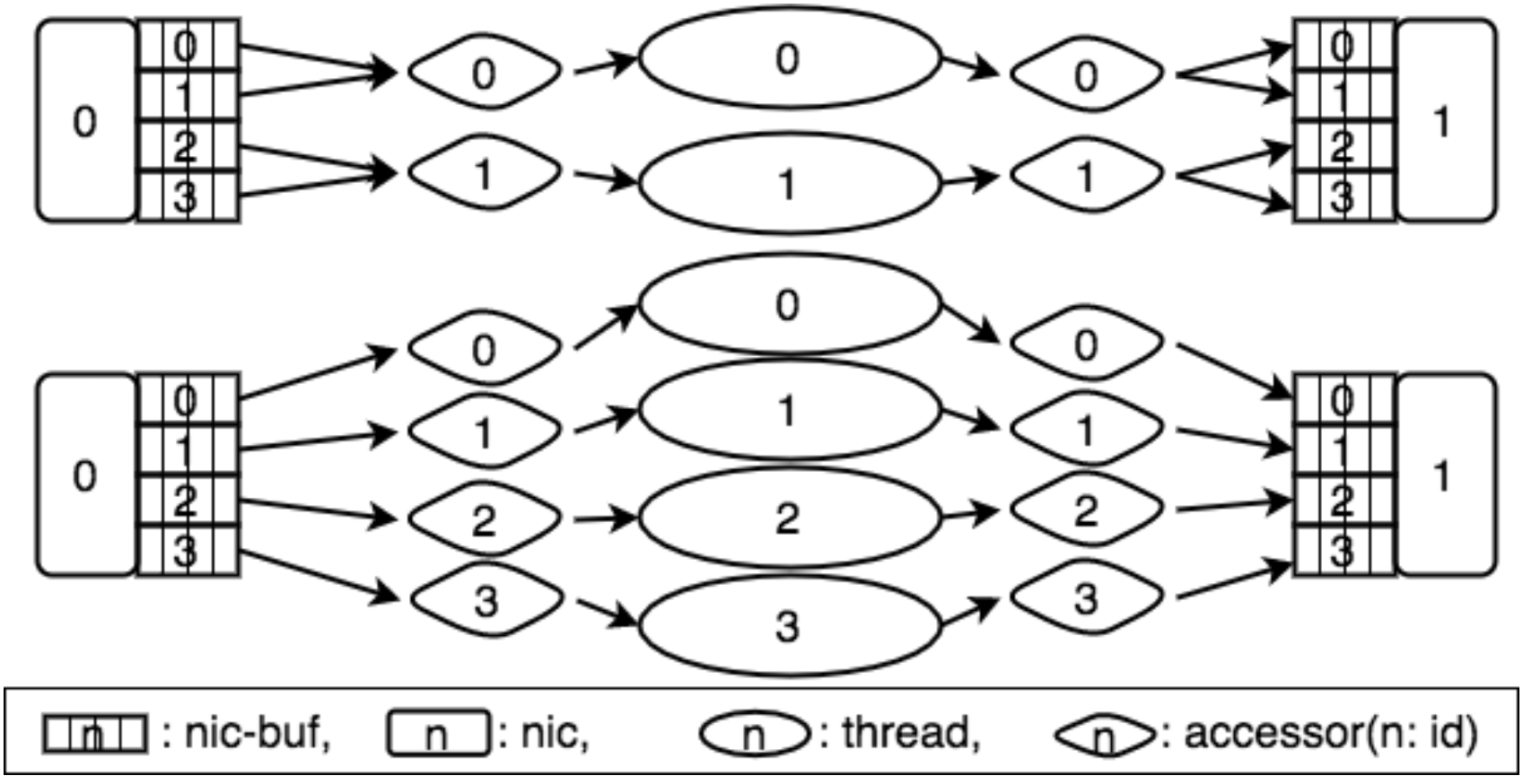


Figure 14: 2-4 多重

D2: まとめ

- 事前にスレッドとポートでネゴシエーションを行い, 各ポートの Accessor を構成し直す
- HW Queue の再構成はリンクダウンをしないといけないので滅多に起こさないようにする. (もちろん可能)
- Accessor が複数の nic-buf を監視している時はラウンドロビンに nic-buf を監視

D2: 最適化の流れ

1. 発火フェーズ
   - VNF を追加したり減らしたりするタイミング
   - トラフィックが増えたり, 減ったりするタイミング
   - タイマーで一定期間ごとに性能チェック.
2. 発見フェーズ (環境情報より発見)
   - NIC のスループット
   - パケット格納用の Queue の統計情報
3. 修正フェーズ
   - スレッドの多重度 (基本的にはこれ)
   - NIC の HW 設定をチューニング

Performance Evaluation

- 懸念点
  - D2 オーバヘッド: 何 ns の処理オーバヘッドか?
  - VM オーバヘッドとどのように: スムーズに進むか?
  - スレッドの起動の速度は?
  - D2 最適化中のトラフィックはどれだけどまるか
- 計測内容: 帯域, 遅延
- VNF: L2FWD, L3FWD, ACL, DPI

全て現在調べ中です. 8 合目合宿までに!!

全体のまとめ

- ssn-NFVi: nonVM な NFV 基盤の開発
- D2: 動的スレッド最適化技術の開発
- システムと分離した場所から D2 の最適化処理を制御するエージェント
- ssn-NFVi 上で動作する VNF 複数種類 (VNF リポジトリ)
  - DPI, Router, FW, etc..

以降補足スライド

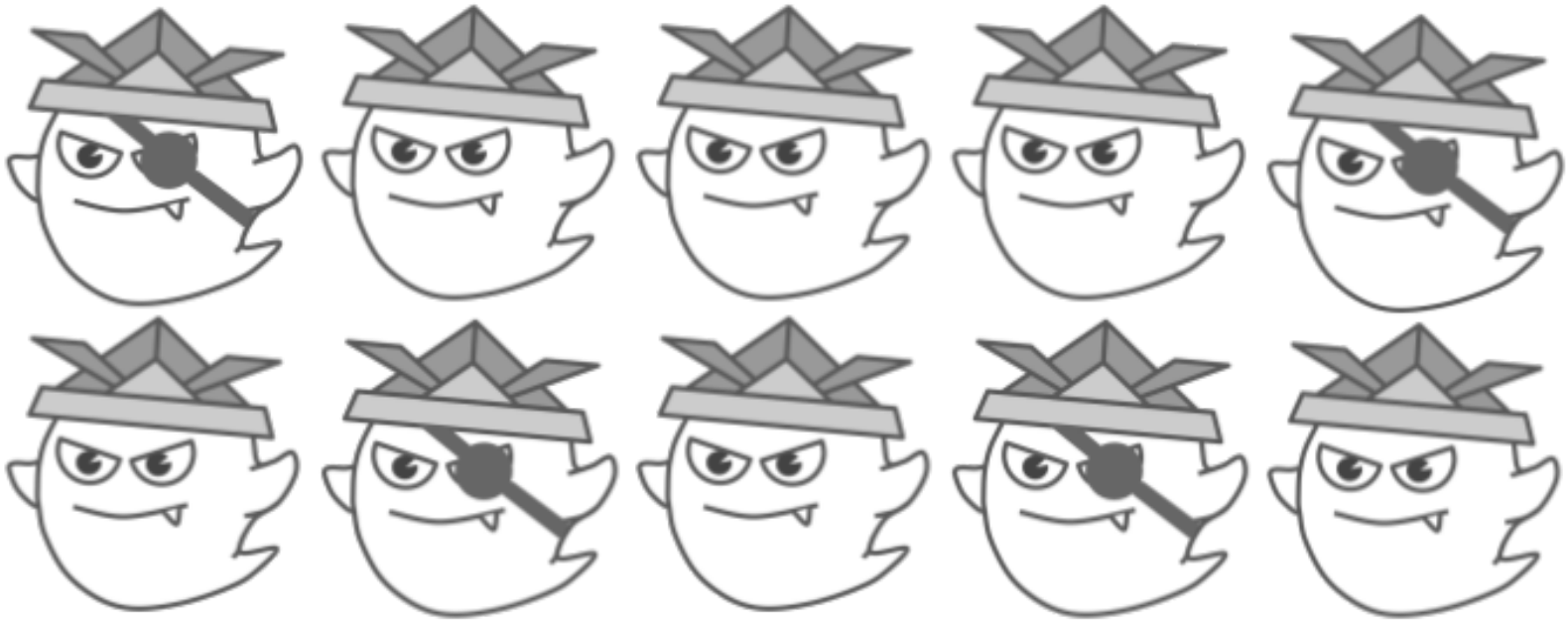


Figure 15: 以降は補足スライド

今後やる+プラスアルファ内容

- 複数ノードでのクラスタリングの動的な性能変更
  - VNF のマイグレーション機能
- 互換性向上
  - VM を用いた VNF のデプロイの対応
  - 物理ネットワークアプライアンスの対応
- VNF の実装
  - 現在開発中: L2fwd, L3fwd, 5tupleACL, DPI
  - 他にも良い VNF 案があれば御指摘ください