SKBパケット選抜総選挙
～ 僕たちは誰について行けばいい？ ～

オープンソースカンファレンス2021 Hokkaido
2021/06/26

本日のスライド：
https://speakerdeck.com/chikuwait/osc21do

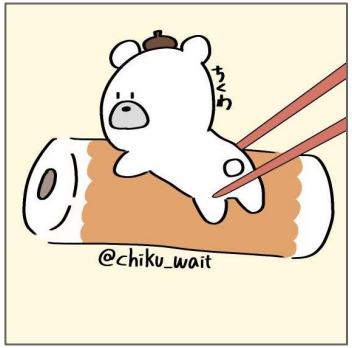
公立はこだて未来大学　システムソフトウェア研究室
中田 裕貴

自己紹介

中田裕貴 / chikuwait

Twitter: chiku_wait　GitHub: chikuwait

- 公立はこだて未来大学大学院
  システム情報科学研究科　高度ICT領域　修士2年
- システムソフトウェア研究室
  - 仮想化技術(コンテナ・ハイパーバイザ)に関する研究
- ハイパーバイザからOS，L2～L4ネットワーク，Kubernetesのあたり触ってます



タイトルのアレ

SKB？

あー，あれね．知ってる知ってる．うん，すごいよね

SKB = sk_buff

皆さんご存知

パケットを管理するLinuxカーネルの構造体

SKB　パケット選抜　総選挙？

はい

SKB(をバイパスする)パケット(処理技術)選抜総選挙

～ 僕たちは誰について行けばいい？ ～　＝　某アイドルの総選挙(2016)のサブタイトル

ちなみに私はアイドルについては専門外なので詳しいことは分かりません（タイトル案は指導教員が作成しました）

どちらかというとCPUアイドルのほうが興味があります

セミナーの内容・ゴール


```
ネットワークパケットを高速に処理できる仕組みがいくつも登場してきました.
このセミナーでは，eBPFやコンテナ技術などに興味がある「低レイヤこじらせ学生」たちが，
XDP・DPDKから最新コンテナネットワーキング技術までをゆるく解説していきます.
```

1. **汎用OSにおけるパケット処理の歴史と移り変わりを知る**
    - Linuxにおけるパケット処理の仕組み
    - 汎用OS上での高速なパケット処理が求められるようになった背景
2. **代表的な高速パケット処理技術について知る**
    - 高速パケット処理を実現するアーキテクチャ
    - 高速パケット処理技術がどのように応用されているのか

パケット処理技術の仕組みと応用を知り

ネットワークインテンシブなシステム・実行基盤を開発・構築する際の取っ掛かりや選択肢を増やす

Linuxカーネルのパケット処理の移り変わりと高速パケット処理技術の登場

カーネル 2.6以前のパケット受信処理の流れ（2003年以前）

1. NICがパケットを受信

2. ハードウェア割り込みが発生

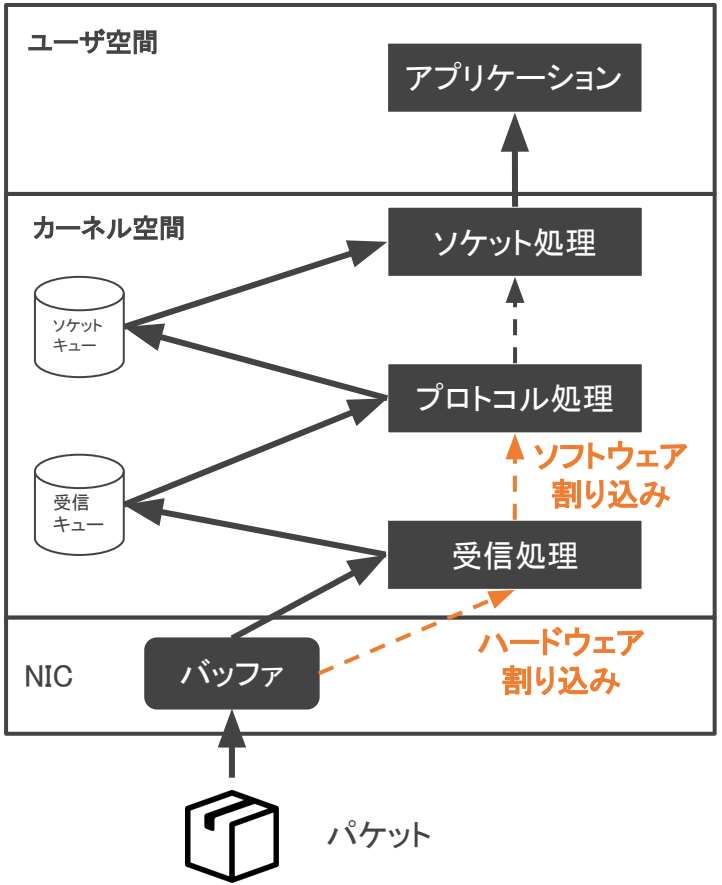
- NICがCPUにパケットを受信したことを通知
- NICに存在するパケットを受信キューにコピー

3. ソフトウェア割り込みが発生

- カーネルは受信キューからパケットを取り出す
- TCPなどのプロトコル処理をしてソケットキューにコピー

特徴

- **1パケット受信するたびに割り込みを受けて処理**
  - パケットを受信するとすぐにCPUに通知
  - パケットを受信してから処理するまでの遅延が小さい



カーネル 2.6以前のパケット受信処理における課題

割り込みの多さ

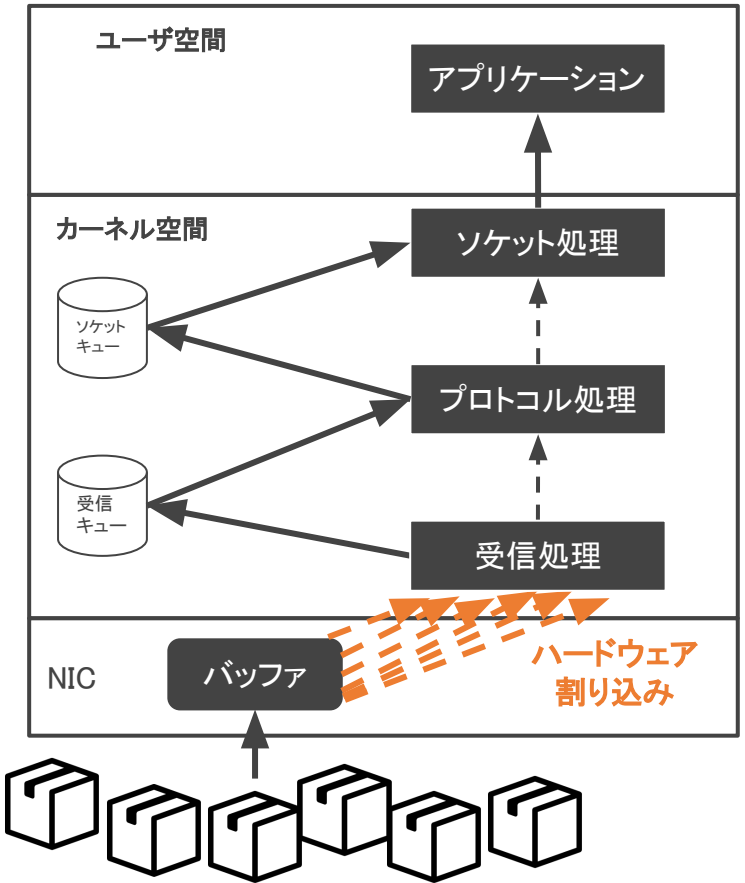
- 1パケット受信するたびに割り込みを受けて処理
  - **パケットを大量に受信すればするほど大量の割り込みが発生**

割り込み処理のオーバーヘッド

- CPUの状態を一時的に保存し，割り込み処理完了後に復元（コンテキストスイッチ）が発生
- 割り込み処理は他のタスクよりも優先度が高い

受信頻度が上がるにつれコンテキストスイッチが増加し，CPUのスループットが低下

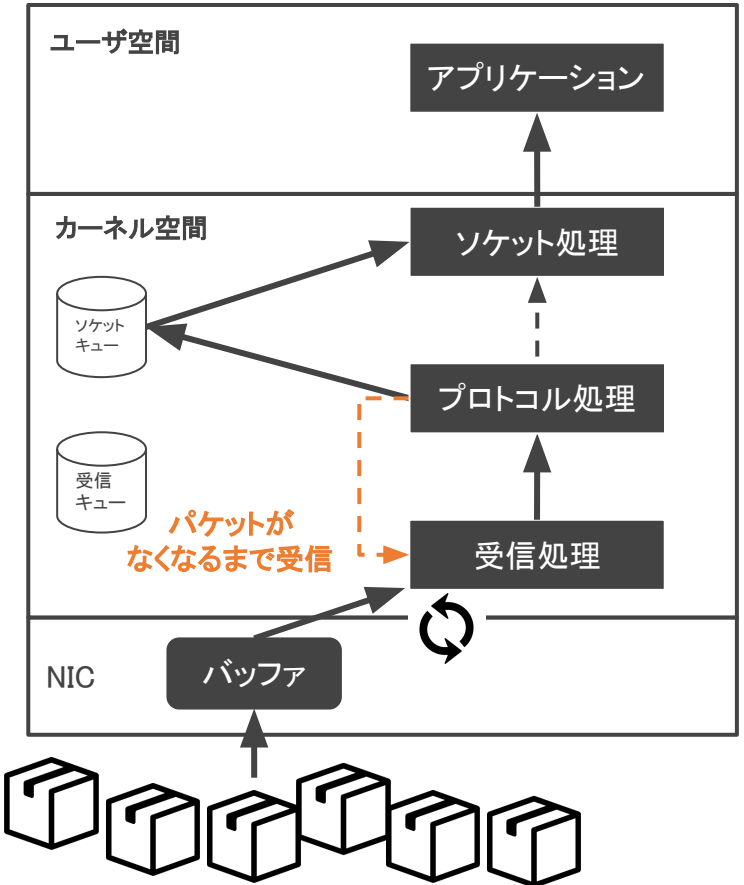
- **最終的に本来すべきタスクが実行できなくなる**
  - Receive Livelock現象
  - **カーネルは割り込みハンドラばかり処理**



カーネル 2.6以降の割り込み削減手法：NAPI（2003年～）

NAPI（New API）

- **パケット受信でハードウェア割り込みが発生すると処理が完了するまでNICからのハードウェア割り込みを一時的に無効化**
  - ドライバの挙動を割り込み駆動からポーリング駆動に切り替え
- ポーリング処理でパケットをNICから取得
  - パケットを受信キューに格納せず直接取り出してプロトコル処理を実施
- **パケットの受信頻度が高い時のみポーリングが有効**
  - 新規パケットがないことをポーリングで検出すると通常の割り込み駆動になる



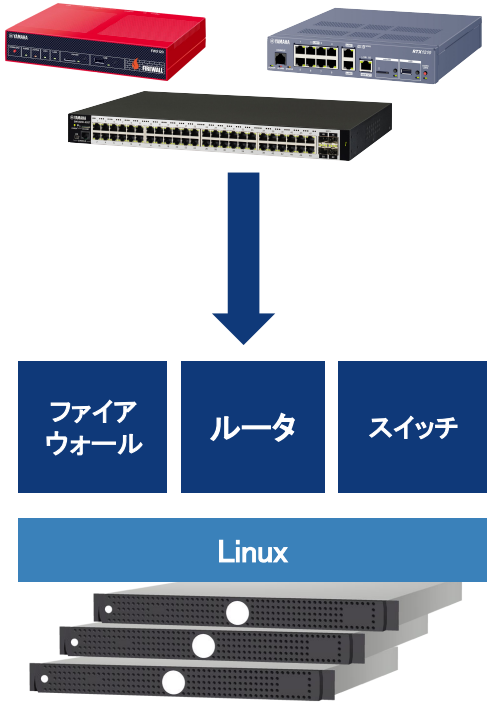
高性能NICの登場とLinux適用領域の拡大(2010年～)

高性能なNICの登場

- 10GbEや25GbE，40GbEのような高速な通信が可能なNICが登場
  - ここ最近は100GbE以上のものも…

Linux適用領域の拡大

- NFV(Network Functions Virtualization)の登場
- 専用のネットワーク機器ではなくPCサーバ上の汎用OS上で
  仮想マシンやコンテナを用いてルータなどのソフトウェアを運用
- ハードウェアコストを削減しながら柔軟な構成を実現



Linux上で高性能なNICを用いて
大量のパケットを処理するようなユースケースが登場

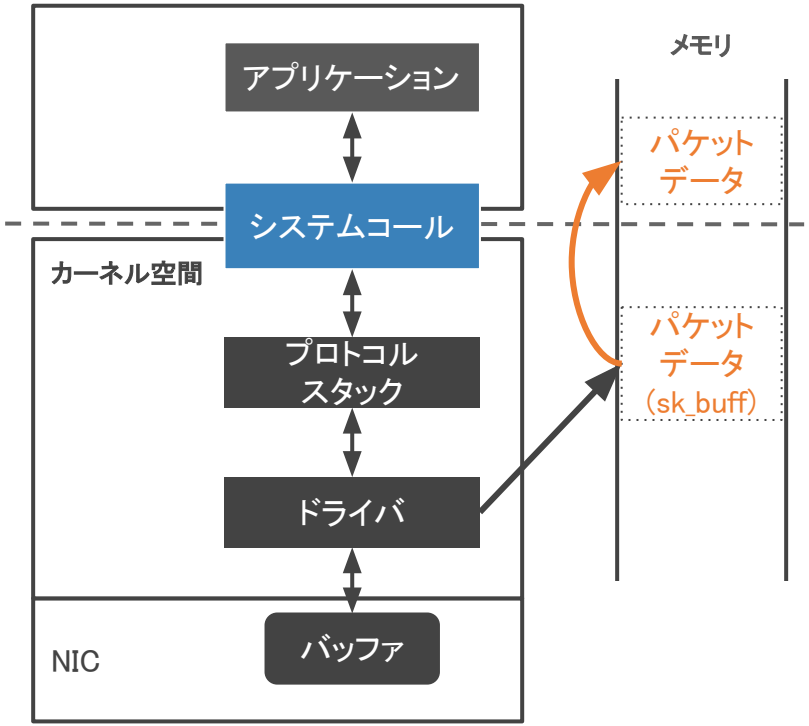
環境変化によって顕在化するネットワークのボトルネック

LinuxはパケットOS専用のOSではない

- 多様なアプリケーションに対応するような汎用的なアーキテクチャ
- **高性能なNICを使用して大量のパケットを処理することで，**
  汎用的な仕組みが逆にボトルネックとなる
  - 高性能を活かしきれない
- ボトルネックに成り得る処理
  - **プロトコルスタック**
  - **コンテキストスイッチ**
  - **メモリ管理におけるキャッシュミス**

プロトコルスタックにおける処理のボトルネック

- **データコピー**
  - 受信したパケットはカーネルがNICのバッファから取り出す
  - パケットはカーネル空間の領域に配置
  - アプリケーションはシステムコール経由で
    カーネル内のパケットを取得
    - **ユーザはカーネル空間にアクセスできない**
    - **パケットをユーザ空間のメモリ領域のコピーする必要**
- **管理構造の肥大化**
  - sk_buff：各パケットに割り当てる管理用構造体
  - 様々なプロトコルに対応することで肥大化
  - **パケットが処理され，ユーザ空間に移ると開放**
  - **パケット毎にsk_buffの確保と開放が**
    大量に行われCPUリソースを消費



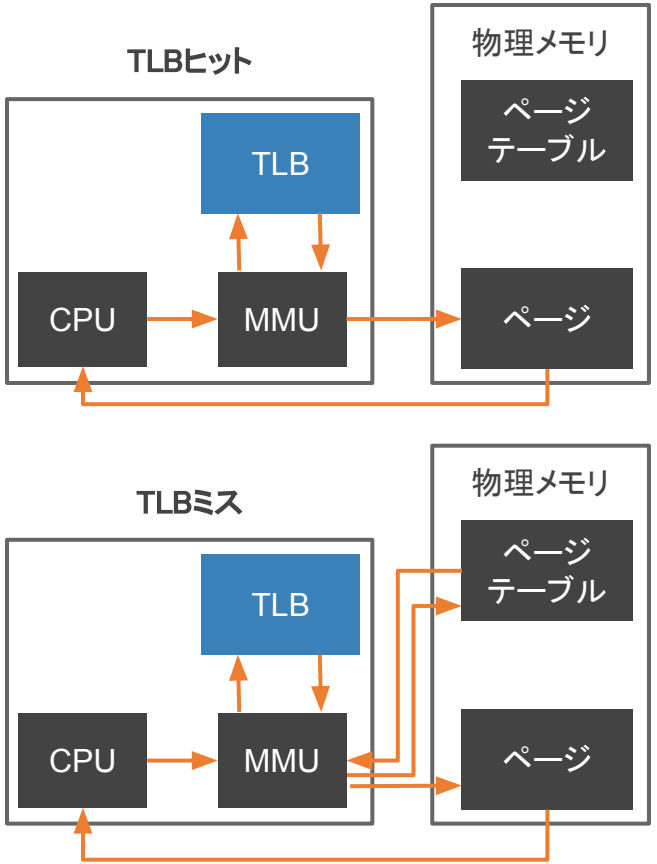
コンテキストスイッチの多さ

- **NICがパケットを受信する度にハードウェア割り込みが発生**
  - CPUの状態を一時的に保存し，割り込み処理完了後に復元
  - **NAPIによって高負荷時の割り込みは抑制**
    - **パケット受信時の割り込みを完全になくすことはできない**
- **システムコール呼び出し**
  - ユーザ空間のアプリケーションは
    パケットを受信するためにソケットに関するシステムコールを呼び出す
  - **システムコールを呼び出す度に，カーネル空間とユーザ空間の切り替え処理が発生**
- **プロセス・コンテキストスイッチ**
  - OSで実行中のプロセスは一定時間でCPUを明け渡し、他のプロセスも実行されるようスケジューリング
  - **常に特定のプロセスがCPUを利用して処理できるわけではなく，切り替え処理も必要**

メモリキャッシュミス

Linuxではページ（4KB単位の領域）でメモリを分割・管理

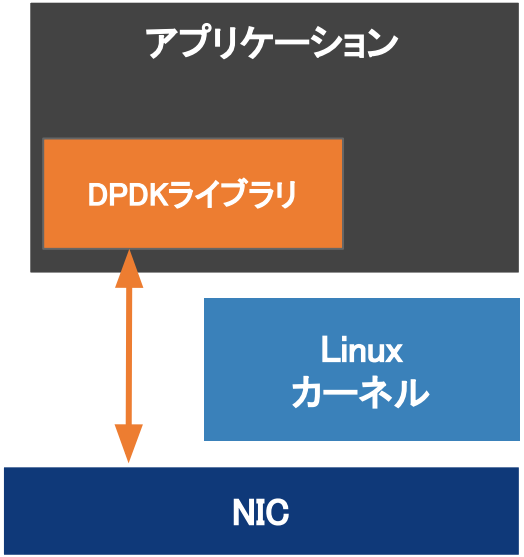
- TLB(Translation Lookaside Buffer)
  - **MMU(Memory Management Unit, アドレス変換実施)**
    に存在するアドレス変換のキャッシュ
  - 物理・仮想アドレスのマッピング管理
  - **物理メモリ上のページテーブル参照より高速**
- TLB上に対応するエントリがあれば
  物理アドレスが返ってくる（TLBヒット）
- **TLBにエントリがない（TLBミス）の時**
  ページテーブルを参照してアドレス変換・TLBエントリの入れ替え
  - TLBに比べて遅い
  - **大量のメモリを用いたパケット処理を行うソフトウェアでは**
    TLBミスの回数が増加してメモリアクセス性能が低下



DPDK（Data Plane Development Kit）

2012年にIntelが公開したOSSの高速パケット処理ライブラリ

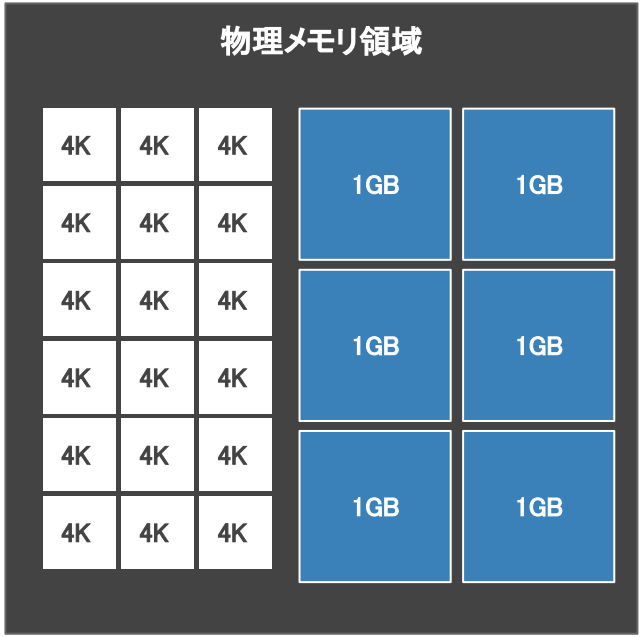
- カーネルをバイパスした高速パケット処理を実現
  - OSカーネルがボトルネックならスルーしてしまえという考え
  - カーネルを介さず，
    ハードウェアとユーザ空間のアプリケーションが直接パケット送受信
- TLBミスへの対策
  - サイズの大きいページを使用
- データコピーの対策
  - カーネルバイパスによるゼロコピー
- 割り込み・コンテキストスイッチの対策
  - ポーリング・CPUコア占有



サイズの大きいページ使用の実現

Hugepages

- 4KB以上にページサイズを大幅に拡張可能
  - 2MBと1GBのページサイズに対応
- ページサイズを大きくすることでアドレス変換テーブルを削減
  - 一つのページで多くの情報を持てる
  - TLBからのヒット率が向上し，メモリアクセス性能向上を実現
- ストレージのスワップ領域は使用しない
  - ページイン・ページアウト処理が発生しない
  - メモリに大きな領域を割当て，
    ページイン・ページアウトしないためメモリの使用効率は悪くなる



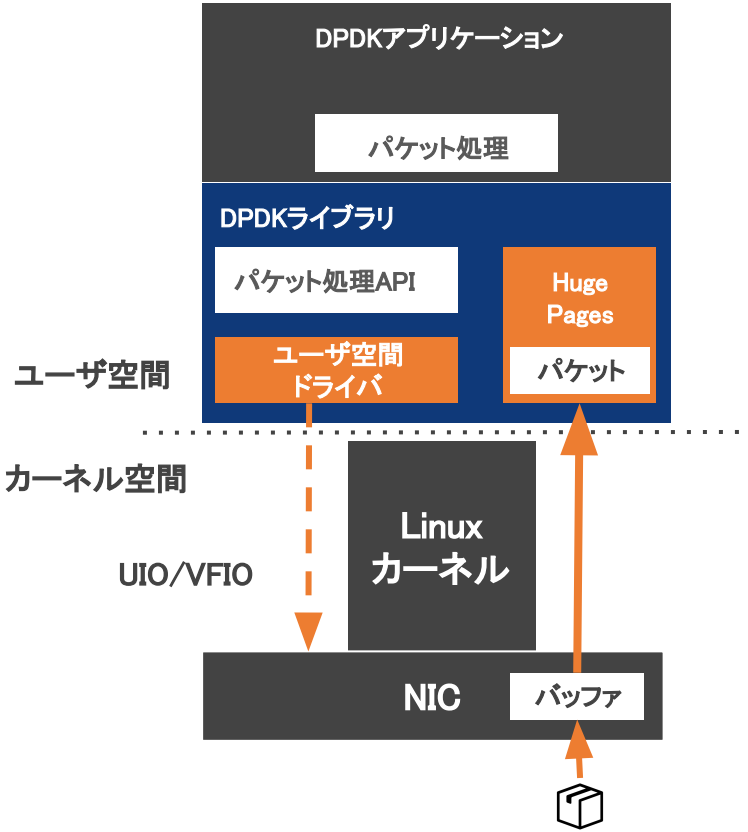
カーネルバイパスの実現

UIO（User Space I/O）/ VFIO(Virtual Function I/O)

- カーネル介さずユーザ空間から直接ハードウェアを制御できるLinux機能
- ハードウェアを制御するレジスタは
  ユーザ空間からはアクセスできない領域に存在
- ユーザ空間上のメモリに制御レジスタをマッピング
  - ユーザ空間で実装したデバイスドライバから制御可能
  - UIO/VFIOを使用すると，NICの制御はカーネルから外れる

ユーザ空間ドライバがNICのキューから
Hugepagesで確保した領域に書き込み

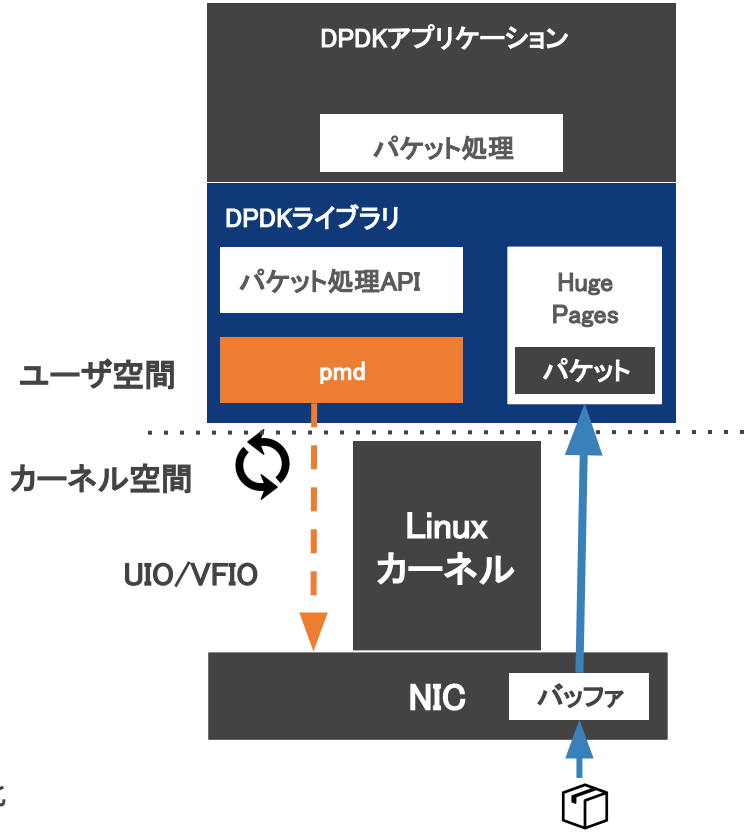
- ユーザ空間でそのままパケットを処理可能



ポーリング・CPU占有を用いた割り込み・コンテキストスイッチの対策

PMD(Poll Mode Driver)

- UIO/VFIOを利用した高速パケット処理を行うデバイスドライバ
- NICに対して常にポーリングでパケット受信を監視
  - NICがパケットを受信次第，
    すぐにHugepagesで確保した領域に書き込む
  - 割り込み無しでパケットを受信



CPUコアの占有による
プロセス・コンテキストスイッチを抑制

- Core affinity機能で特定のプロセスにCPUコアをバインド

割り当て例：受信・パケット処理・送信処理毎にコアを割り当て，ブン回す

- ハイパースレッディングも意識する必要
- L2/L3キャッシュを共有するため，スレッドごとに処理が違うと効率悪化

Linuxカーネルコミュニティの取り組み：BPF（1997年～）

BPF（Berkeley Packet Filter）：カーネル内パケットフィルタリング

- 1992年にUNIX(BSD)上でパケットキャプチャ・フィルタリングを効率的に行うために開発[1]
  - 独自の命令セットを持ったカーネル内仮想マシン
  - 1997年にLinuxカーネルに移植
  - **カーネル空間でパケットフィルタリングを実施**
  - **ユーザ/カーネル空間切り替え処理の削減**
- **BPFプログラムによるカーネルクラッシュを防ぐために検証機構を持つ**
  - **安全にカーネル内でパケット処理に介入可能**
- **あくまでもキャプチャ・フィルタリング目的であった**

  e.g., tcpdump

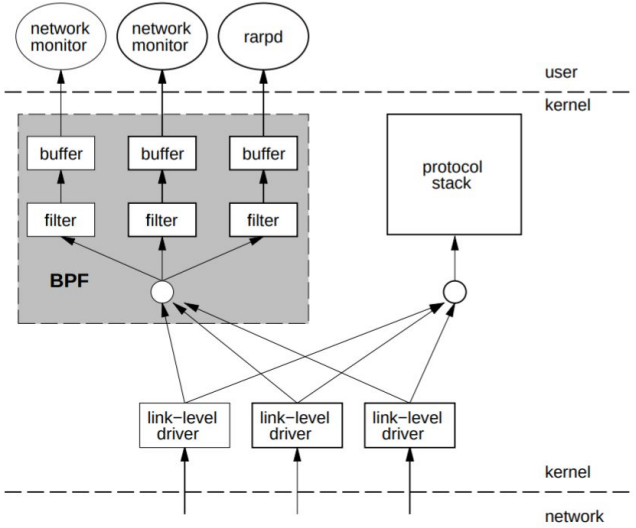


Figure 1: BPF Overview

[1]Steven McCanne and Van Jacobson. 1993. The BSD packet filter: a new architecture for user-level packet capture. In Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings (USENIX'93). USENIX Association, USA, 2.

Linuxカーネルコミュニティの取り組み：eBPF（2014年～）

eBPF(extended BPF)

- **BPFをより汎用的なカーネル内仮想マシンにするための拡張**
  - Linuxカーネル3.14で実現
  - 命令セットの一新，レジスタ数・幅の増加
  - パケット以外のカーネル内のあらゆる操作をフック可能
- **eBPF mapsの導入**
  - **ユーザ/カーネル空間やBPFプログラム間でのデータやりとり**
  - eBPF内でKVSが使用可能
- **eBPFプログラムから別のeBPFプログラムへのジャンプ**

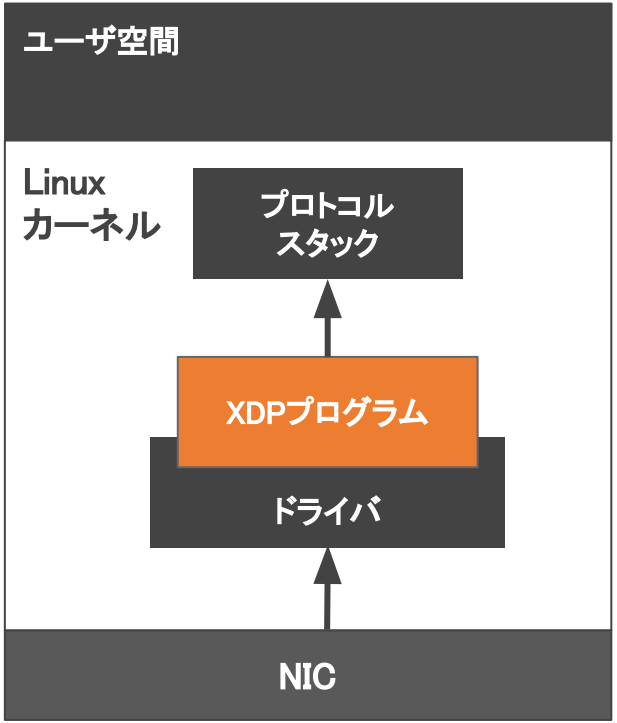
カーネルの様々なパケット処理にアタッチしたeBPFプログラム間で
連携しながら処理を行い，カーネル・ユーザ間で効率的にデータを共有

- **カーネル内での高速パケット処理の機運**

Linuxカーネルコミュニティの取り組み：XDP（2016年～）

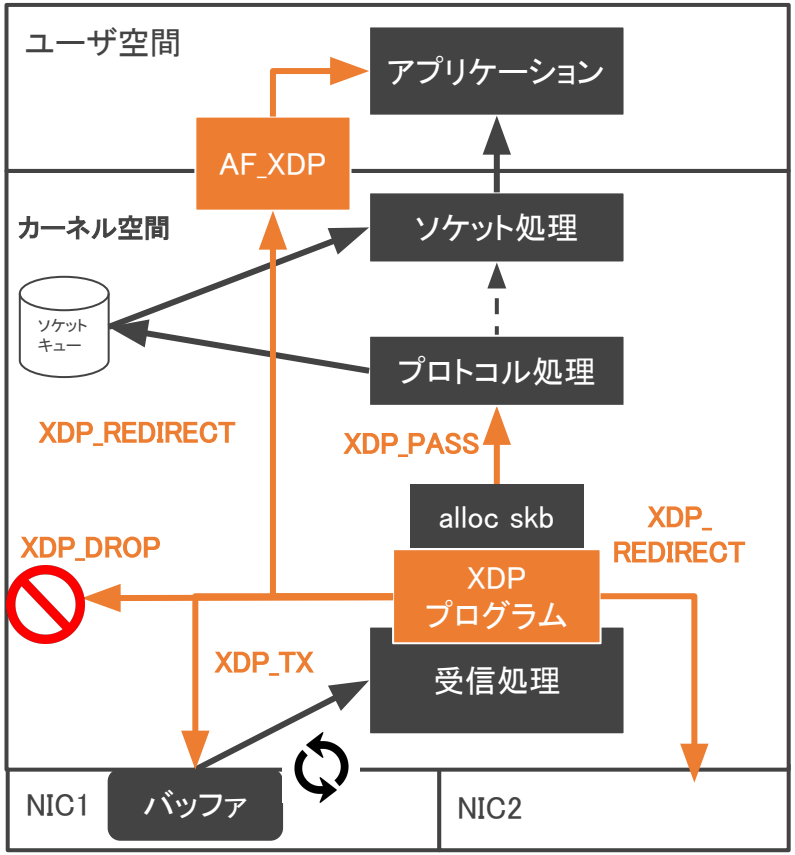
XDP(eXpress Data Path)

- **Linuxカーネル内に実装された高速パケット処理基盤**
  - Linuxカーネル4.8で導入
  - **14Mpps以上の処理性能を発揮**
    - 10GbEのワイヤーレート
- **NICドライバにeBPFプログラムをアタッチ**
  - **パケットがネットワークスタックに到達する前に**
    フィルタリングや転送などのパケット処理が可能
- XDP対応Smart NICを使うとハードウェアオフロードも可能
  - CPU負荷を軽減
- **Linuxの機能を活用したまま，高速パケット処理が実現**



XDPにおけるパケット処理

- **XDPプログラムはNAPIのポーリング処理にアタッチ**
  - **sk_buff割り当て前のパケットを改変・処理可能**
  - **sk_buffの割り当て・開放処理が不要**
- **AF_XDP**
  - 新しいソケットタイプ
  - **XDPで処理したパケットを**
    プロトコルスタックをバイパスしてユーザ空間に渡す
- **XDPのパケット処理アクション**
  - XDP_PASS：プロトコルスタックに流す
  - XDP_TX：受信したNICで送信
  - XDP_REDIRECT：別NICやAF_XDPなどにリダイレクト
  - XDP_DROP：パケットをドロップ



DPDK・XDP比較

OSカーネルの扱い

DPDK:ドライバレベルからバイパス

XDP:プロトコルスタックの処理をバイパス

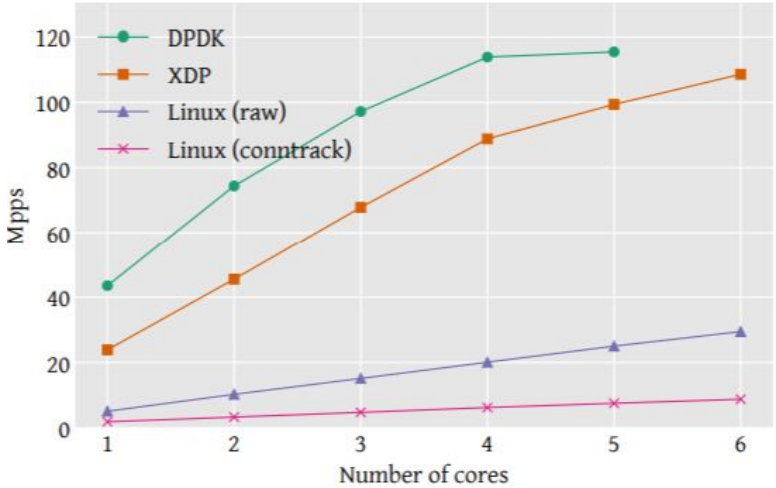
CPUリソースの扱い

DPDK:OS管理から分離して占有

XDP:専有せず全てのプロセス・処理と共有

どっちが優れている？:一長一短（個人の主観を含む）

- パケット処理性能ではXDP < DPDK
- **DPDKはプロトコルスタックも用意する必要**
  - **XDPはLinuxの機能・仕組みを活用可能**
- **どっちもパケット処理を自分で書かないといけない**
  - **皆さん一緒にOSとネットワークの勉強をしましょう！！！**



パケットドロップ性能[2]

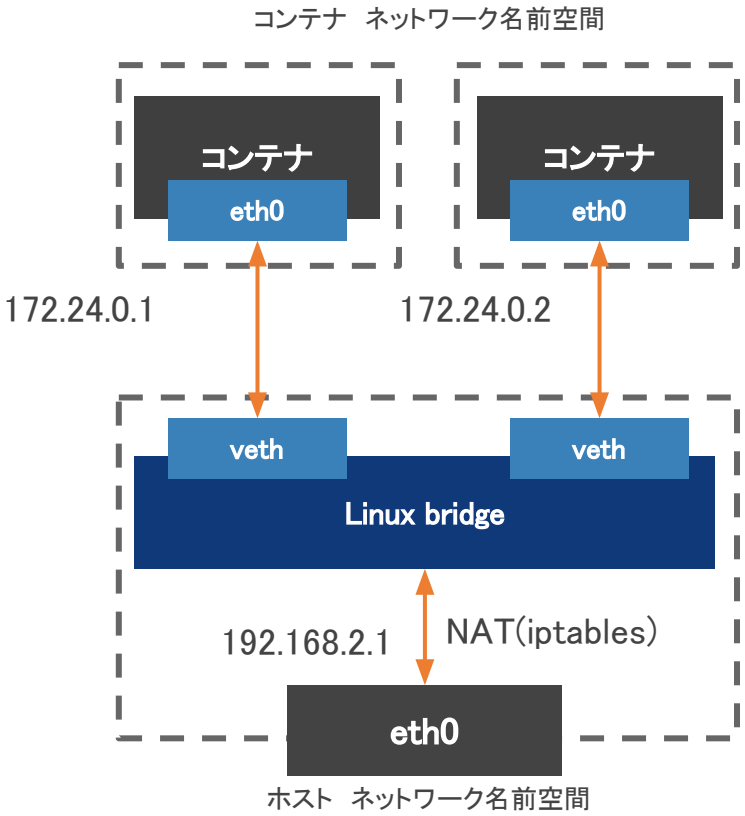
[2]Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. 2018. The eXpress data path: fast programmable packet processing in the operating system kernel. In Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '18). Association for Computing Machinery, 54–66.

高速パケット処理技術の応用

～コンテナネットワークに添えて～

前提:コンテナネットワークの仕組み

- **vethを使用して各コンテナに仮想NICを提供**
  - 仮想NICのペアを作成する機能
- **L2ブリッジとNATを用いてコンテナの接続性を提供**
  - Linux bridgeを使用してL2ブリッジを作成
  - ペアの片方をL2ブリッジに接続
  - 外部との通信はiptablesのNAT機能を用いて制御



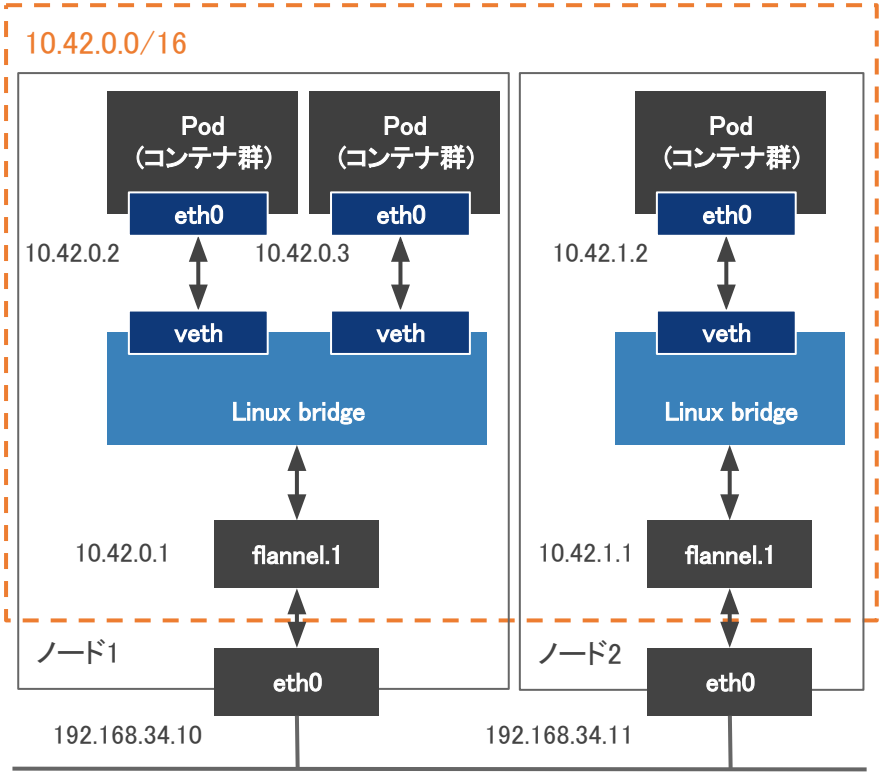
前提：Kubernetesにおけるネットワーク

CNIプラグイン：CNIという規格に準拠したネットワークを提供

- 様々なCNIプラグインが存在
- 要件に応じてネットワーク仕様を選択

Flannel：最もポピュラーなCNIプラグイン

- **VXLANによるオーバレイネットワーク**
  - クラスタ全体の論理ネットワークを構成
  - ノードをまたぐ通信はカプセル化
  - flannel.1がVXLANインタフェース
- 全Podの仮想NICはLinux bridgeで接続
  - 各ノード毎にサブネットを構成
- iptablesでPod間のロードバランシング

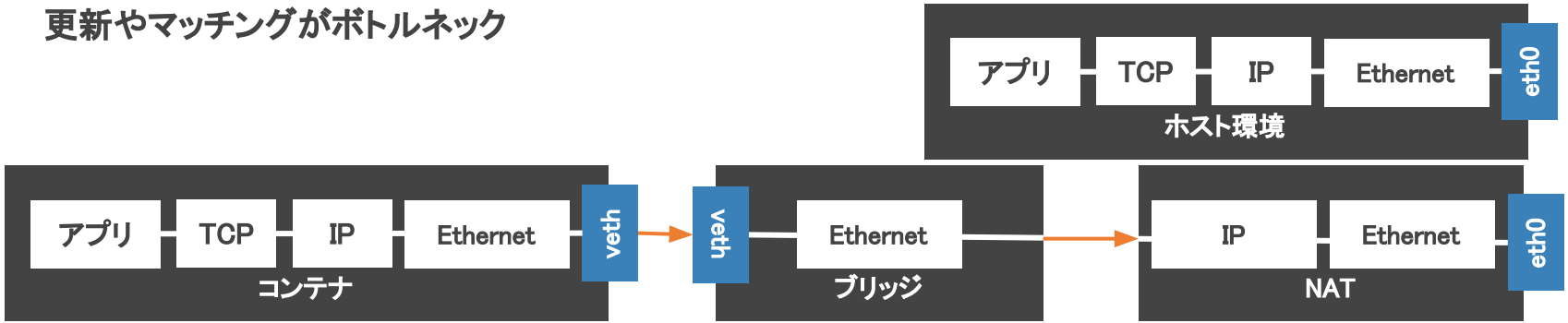


既存のコンテナネットワークにおける課題

- **データパス長大化による性能低下**[3,4]

  ホスト環境やホストのネットワークを使用した場合)と比べパケット送信までに多くの処理と時間が必要 [5]

- **iptables肥大化による性能低下**
  - kubernetesは，大量のコンテナを運用してコンテナを頻繁に起動・停止
  - **ルールが肥大化し，**
    更新やマッチングがボトルネック



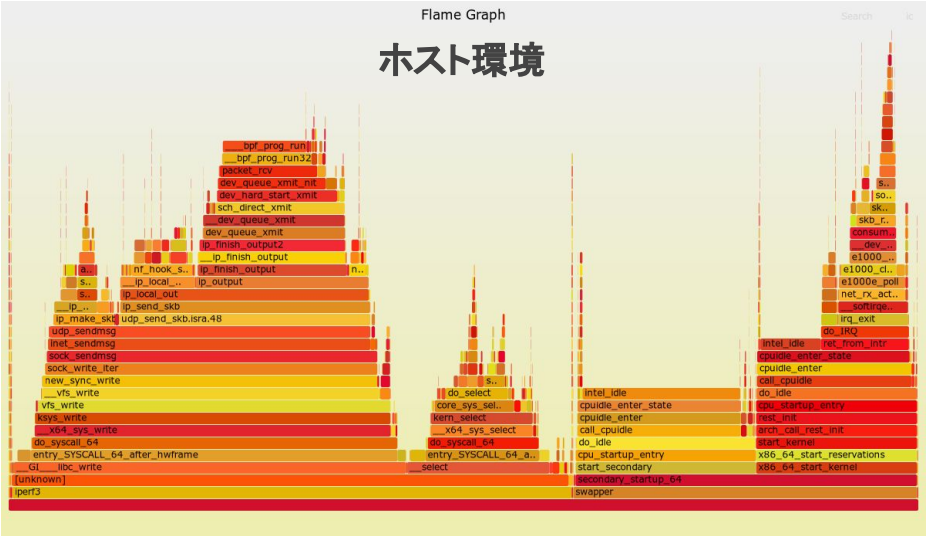
[3]Anderson, J., Hu, H., Agarwal, U., Lowery, C., Li, H. and Apon, A.: Performance considerations of network functions virtualization using containers, 2016 International Conference on Computing, Networking and Communications (ICNC), pp. 1–7 (2016).

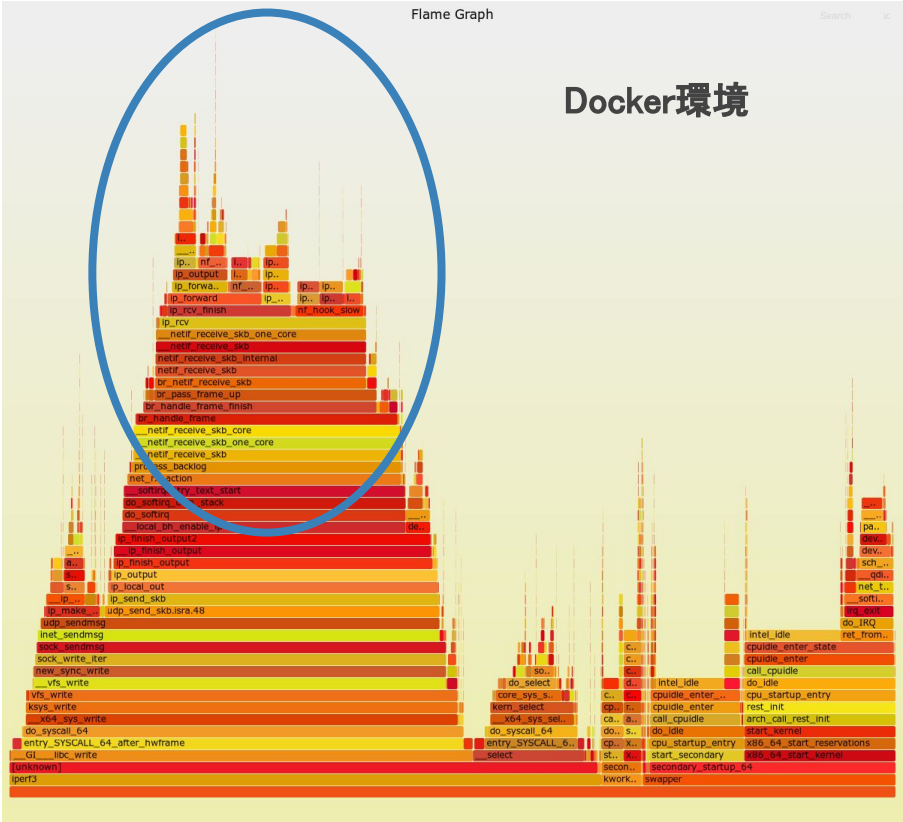
[4]Zhao, Y., Xia, N., Tian, C., Li, B., Tang, Y., Wang,Y., Zhang, G., Li, R. and Liu, A. X.: Performance of Container Networking Technologies, Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems, HotConNet '17, Association for Computing Machinery, pp. 1–6 (2017).

[5]Nakamura, R., Sekiya, Y. and Tazaki, H.: Grafting Sockets for Fast Container Networking, Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems, Association for Computing Machinery, pp. 15–27 (2018).

非コンテナ環境とコンテナ環境における OSカーネルのパケット処理呼び出しの差

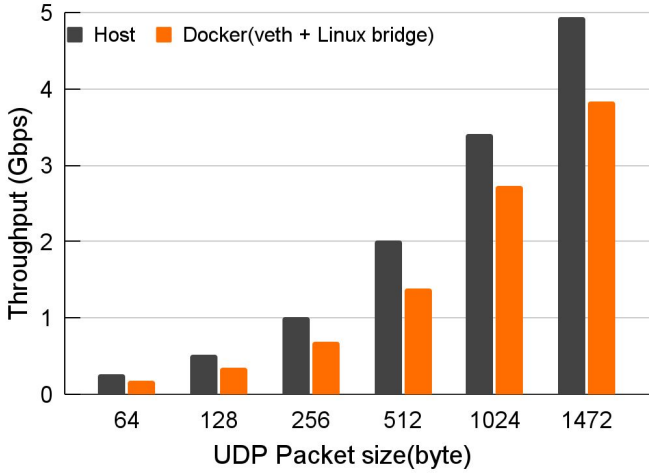
- iperf3(ネットワークベンチマーク)を使用して
  パケットを送信した際のフレームグラフ
  - perf(Linuxのパフォーマンスモニタリング)を使用
  - **ブリッジとvethの使用によって，**
    カーネル処理が多く発生





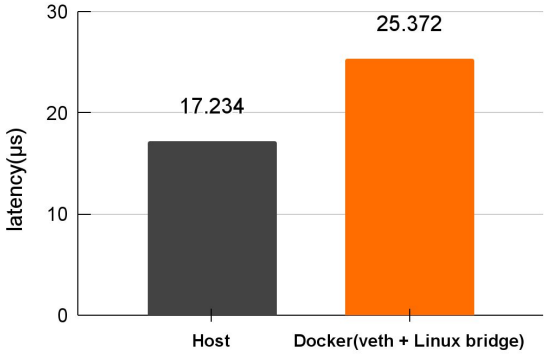
非コンテナ環境とコンテナ環境におけるネットワーク性能の差

- スループット(iperf3)，レイテンシ(sockperf)を評価
  - コンテナから別マシンへのUDPパケット送信
  - スループットはパケットサイズを変更しながら計測
- 1472バイトの時，スループットが約23%低下
- レイテンシは約48%増加



実験環境

OS	Ubuntu 20.04LTS (Linux Kernel 5.12.12)
CPU	Intel Xeon E-2286M @2.40GHz 8コア16スレッド
RAM	64GB
NIC	Aquantia AQtion AQC107 10GbE



性能低下の影響を受ける環境(1/2)

1. ネットワークインテンシブなアプリケーションのコンテナ実行

- NFV(Network Functions Virtualization)
  - 汎用OS上でルータやスイッチなどのネットワーク機器を実装・実行
- 分散処理・機械学習
  - 大規模なクラスタ間を低遅延・高速なネットワークで接続して演算速度の向上
- **CPUやネットワークリソースを最大限に使い切ることで，高性能なシステムを実現**

OSのパケット処理と，データパス長大化の2つのオーバヘッドによって

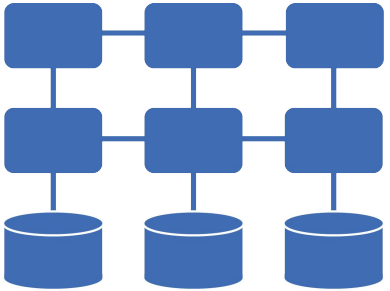
ネットワークリソースを使い切れない

性能低下の影響を受ける環境(2/2)

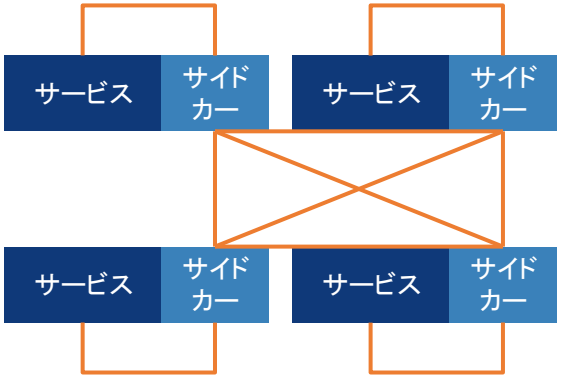
2. コンテナでマイクロサービス・アーキテクチャを実現したシステム

- スケーラビリティ・可用性を向上させるための手法
- システムを分割し，ネットワークで接続
- サービスメッシュの採用(e.g., Istio)
  - 動的に変化する接続情報の管理(サービスディスカバリ)や障害連鎖の防止(サーキットブレーカ)を実現
  - サイドカープロキシ(e.g., Envoy)コンテナを経由して通信
  - アプリケーションからサービス間通信を分離

多数のコンテナでコンテナ間通信が発生することで
データパス長大化・iptables肥大化が性能に大きく影響



マイクロサービス・アーキテクチャのイメージ図



サービスメッシュのイメージ図

コンテナネットワークを対象とした高速パケット処理技術の適用

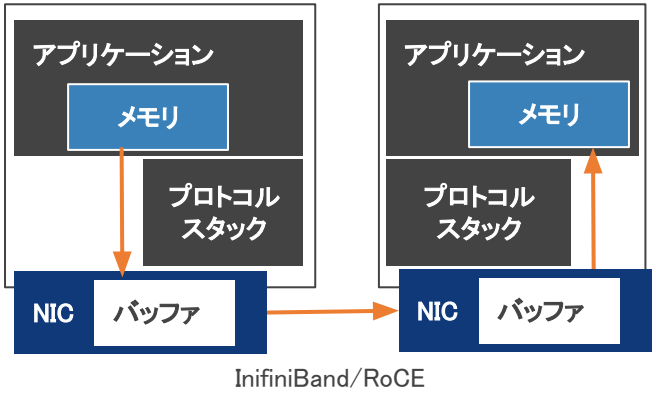
データパス長大化による性能低下の対策

1. NFVを対象としたDPDK適用

2. 分散処理・機械学習向け

   RDMA(Remote Direct Memory Access)適用

   - デバイスでアプリ内のメモリを読み，
     OSバイパスで転送先アプリのメモリに書き込む技術
   - ネットワーク適用例：
     InifiniBandやRoCE(RDMA over Converged Ethernet)



RDMAの一例

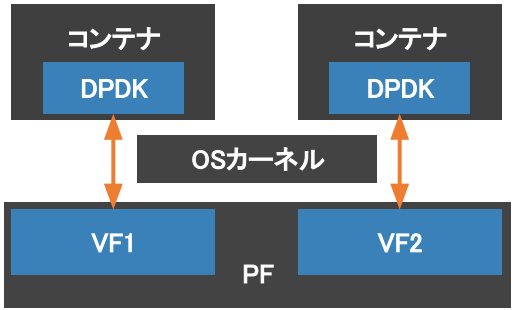
データパス長大化・iptables肥大化による性能低下の対策

3. Kubernetesを対象にしたeBPF・XDP適用

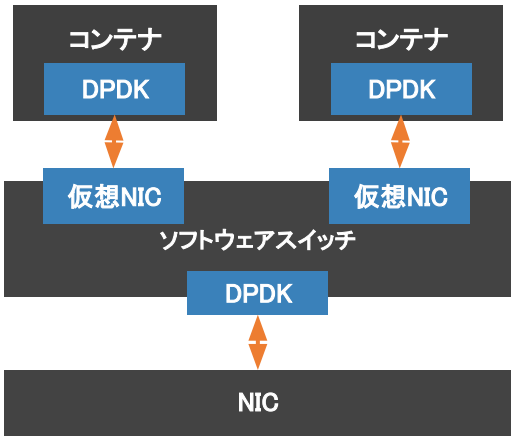
コンテナネットワークにおけるDPDKの適用パターン

1. スライシング(SR-IOV + DPDK)
   - **SR-IOV：ハードウェア機能で物理NIC（PF）上に仮想NIC（VF）を作成**
   - **各コンテナに対してVFを提供**
     - 提供されたVFでDPDKを使用してパケット処理

2. **アグリゲーション(ソフトウェアスイッチ + DPDK)**
   - DPDK使用可能ソフトウェアスイッチとコンテナを接続
   - **ソフトウェアスイッチとコンテナ双方でDPDKを使用することで，通信を集約・制御**



スライシング・パターン

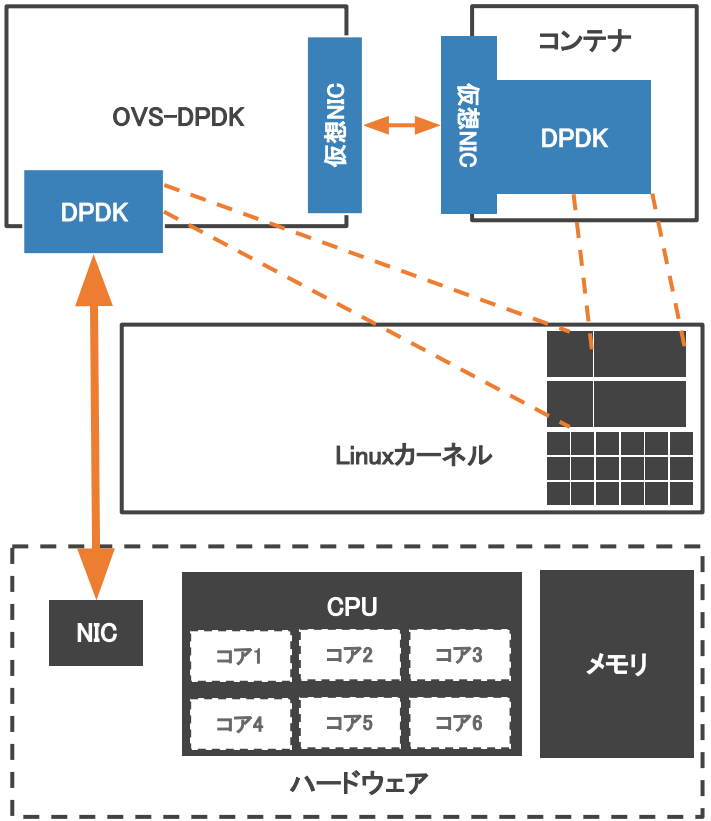


アグリゲーション・パターン

ソフトウェアスイッチ + DPDKを用いたコンテナネットワーク

Open vSwitch(OVS)

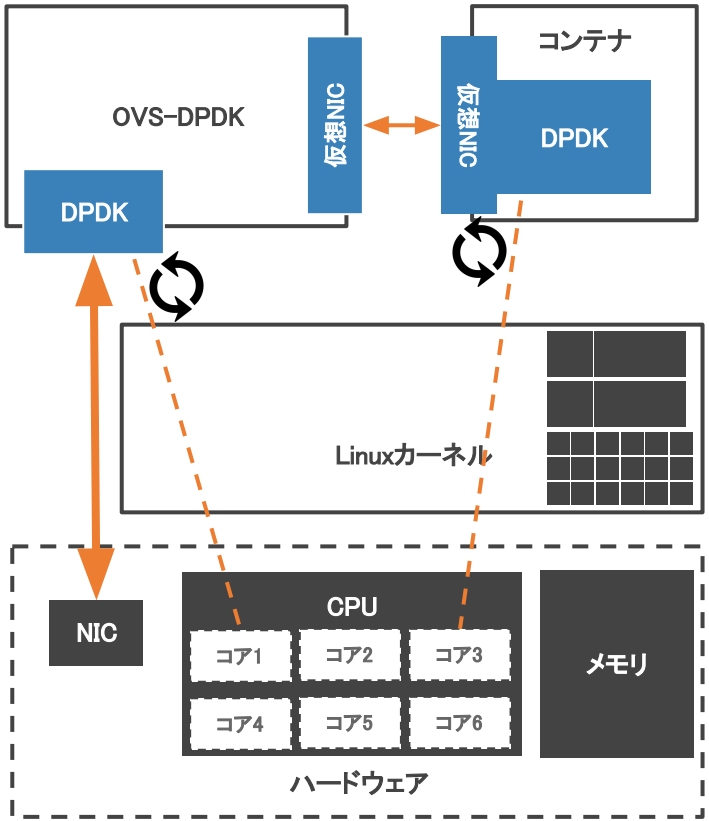
- 商用環境などで幅広く利用
- **OVS-DPDK：DPDKを用いたデータパス実装**
  - **DPDKをバインドしたNICをブリッジで使用可能**
- **利用方法**
  - DPDKが有効なOVSのパッケージ(apt:openvswitch-switch-dpdk)
    or --with-dpdkオプションを使ってビルド
  - OVS・コンテナ側でHugepagesや使用するCPUコアの指定
  - DPDKをバインドしたNICをブリッジに追加
- **特殊な仮想NICのペアを使用**
  - DPDKを利用できる高速パケット処理用途



ソフトウェアスイッチ + DPDKを用いたコンテナネットワーク

Open vSwitch(OVS)

- 商用環境などで幅広く利用
- **OVS-DPDK：DPDKを用いたデータパス実装**
  - **DPDKをバインドしたNICをブリッジで使用可能**
- **利用方法**
  - DPDKが有効なOVSのパッケージ(apt:openvswitch-switch-dpdk)
    or --with-dpdkオプションを使ってビルド
  - OVS・コンテナ側でHugepagesや使用するCPUコアの指定
  - DPDKをバインドしたNICをブリッジに追加
- **特殊な仮想NICのペアを使用**
  - DPDKを利用できる高速パケット処理用途



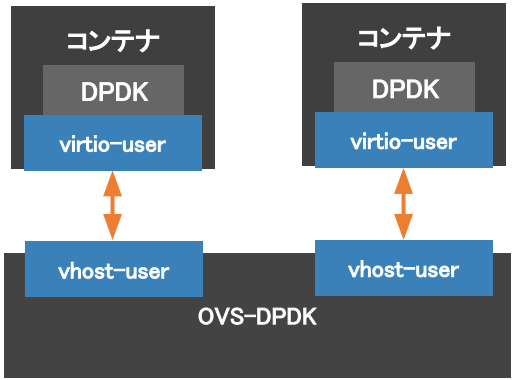
特殊なNICの割り当て：virtio-user/vhost-user

virtio-user（コンテナ側）

- ユーザ空間上でvirtio(準仮想化I/O方式)を用いた仮想デバイスを実現
  - コンテナでDPDKを使用した高速なネットワークI/Oを実現するために実装[6]
  - virtio-net（仮想マシン・ホスト間の通信）が元
    - コンテナでのDPDK利用要件が，デバイスエミュレート以外ほぼ同一
- バックエンドデバイス(vhost)と通信可能

vhost-user(OVS-DPDK側)

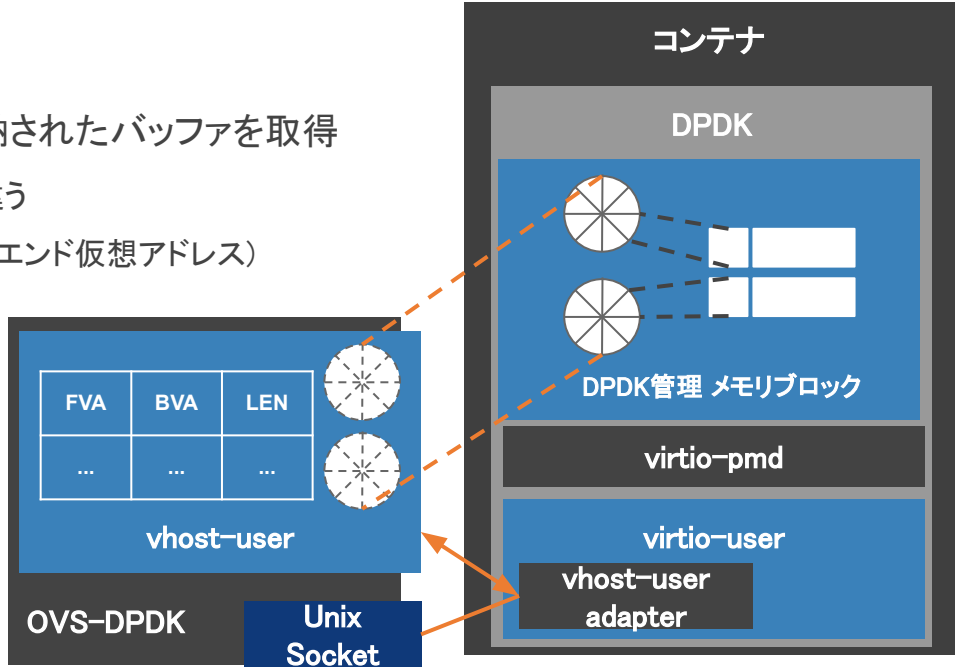
- ユーザ空間上でvirtioとのデータプレーンを構成
  - 元となったvhost-netは仮想マシン・ホスト間で使用するカーネルモジュール
- ユーザ空間に実装し，DPDKを用いるVM・コンテナとOVSの組み合わせを実現
  - virtio-user・vhost-user間は共有メモリベースの通信



[6] Jianfeng Tan, Cunming Liang, Huawei Xie, Qian Xu, Jiayu Hu, Heqing Zhu, and Yuanhan Liu. 2017. VIRTIO-USER: A New Versatile Channel for Kernel-Bypass Networks. In Proceedings of the Workshop on Kernel-Bypass Networks (KBNets '17). Association for Computing Machinery, 13-18.

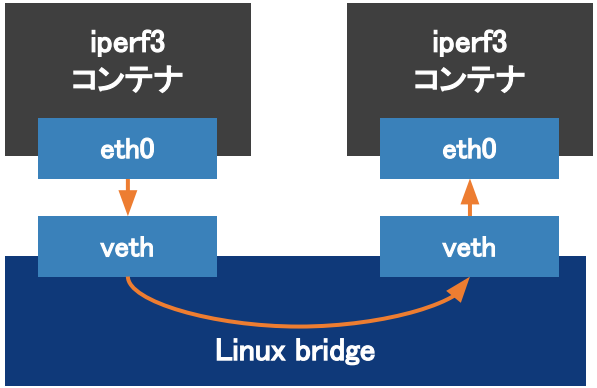
virtio-user・vhost-user間の通信

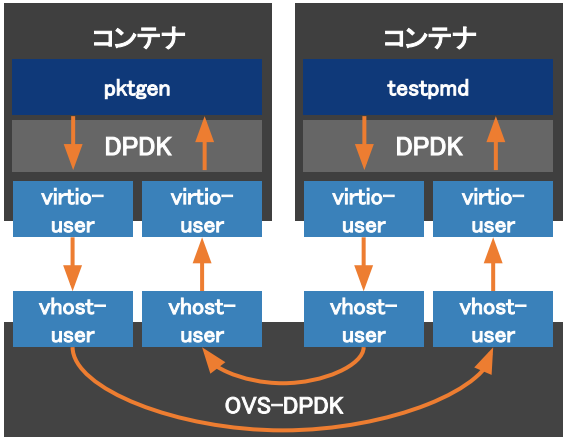
- virtioはvirtqueueと呼ばれるTX/RXキューを提供
  - DPDKで送信したパケットはこの中に入る
- virtio/vhost間は共有メモリベースの通信
  - virtqueueとDPDK管理のメモリブロックを共有
  - vhost-userはアドレス変換してパケットが格納されたバッファを取得
    - 異なるプロセスであり，仮想アドレス空間が違う
    - FVA(フロントエンド仮想アドレス)・BVA(バックエンド仮想アドレス)を変換するテーブルを持つ
- 制御情報はUnix Socketを用いて交換
  - virtqueueを共有するための
    メモリマッピング情報
  - データをvirtqueueに入れた場合に
    反対側にキックするための情報



性能評価:コンテナ間通信のスループット

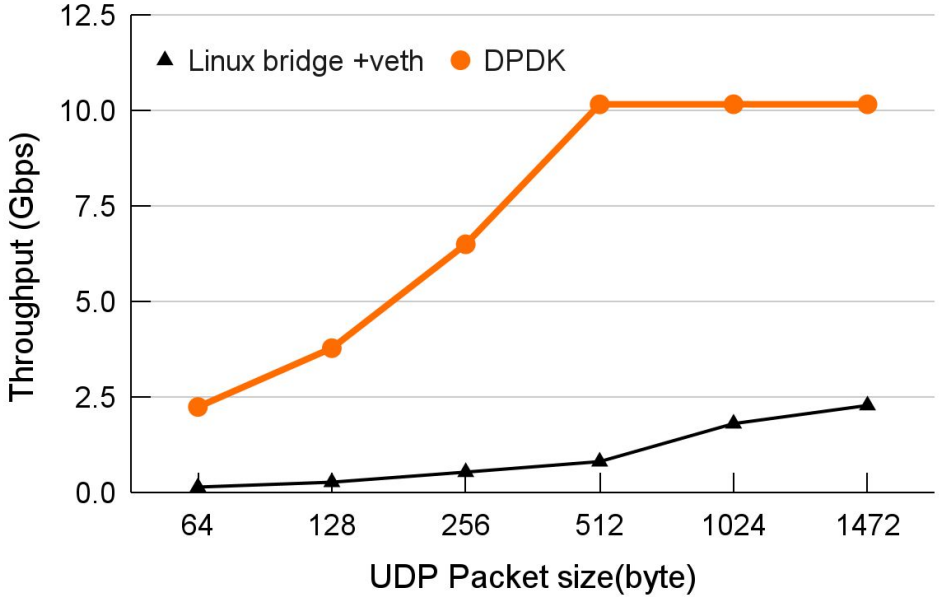
- **単一マシン内コンテナ間通信のUDPスループットを比較**
  - パケットサイズを増やしながら測定
- **Linux bridge + vethを用いた環境**
  - iperf3を使用
- **DPDKを用いた環境**
  - pktgen:DPDKを使用したパケットジェネレータ
  - testpmd:DPDKを使用したサンプルアプリケーション
    あるNICから来たパケットを別NICにフォワーディングできる





性能評価:コンテナ間通信のスループット

- 64バイトのとき，デフォルトのネットワークでは139Mbps，DPDKを用いた環境では2236Mbps
  - **約16倍の性能**
- 1472バイトでは，
  デフォルトのネットワークでは約2.2Gbps
  DPDKを用いた環境では約10.1Gbps
  - **約4倍の性能**
- **ショートパケットほど性能差が大きい**
  - 高スループットを達成するには
    大量のパケットを処理する必要
  - **カーネルの割り込みや**
    コンテナのデータパスの長大化の影響を大きく受ける



KubernetesにおけるDPDKの利用

汎用アプリケーションとNFVの特性の違い

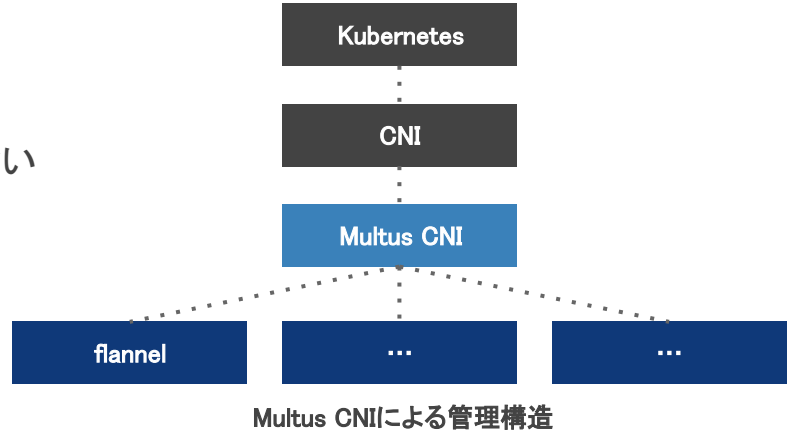
- 汎用アプリケーション
  - NICの割り当て：単一の仮想NIC(veth)で十分
  - L3やL2などクラスタ内のルーティング方式を意識しない
    - pod間の通信ができて，外部に出れれば良い
- **NFV**
  - 大量のトラフィックを低遅延で高速に処理
  - **vethだけではなく高速パケット処理用途のNIC(virtio-user)の使用**
  - **管理用・サービス用のNICとネットワークを分離**
    - kubernetesのCNIは，podに対して単一のNICのみ提供

シンプルな仮想NICだけではなく，標準のKubernetesネットワーク外で

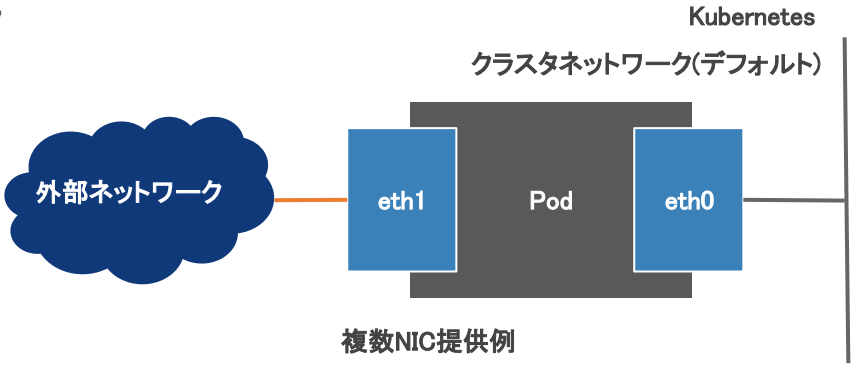
高速なL2・L3ネットワークに接続可能なNICを提供する方法が必要

Multus CNI：複数NIC・ネットワークの提供を実現

- **メタCNIプラグイン**
  - 複数のCNIプラグインを同時に使用・管理
  - Multus CNIがNICとネットワークを提供するわけではない
- **各CNIプラグイン毎にNICを提供**
  - **podが複数のNICを使用し，**
    複数のネットワークに接続することを実現
- 特定のNICをクラスタネットワークから分離しつつ，
  kubernetesのライフサイクルに則った
  自動的なNICのアタッチ・デタッチといった恩恵



Multus CNIによる管理構造



複数NIC提供例

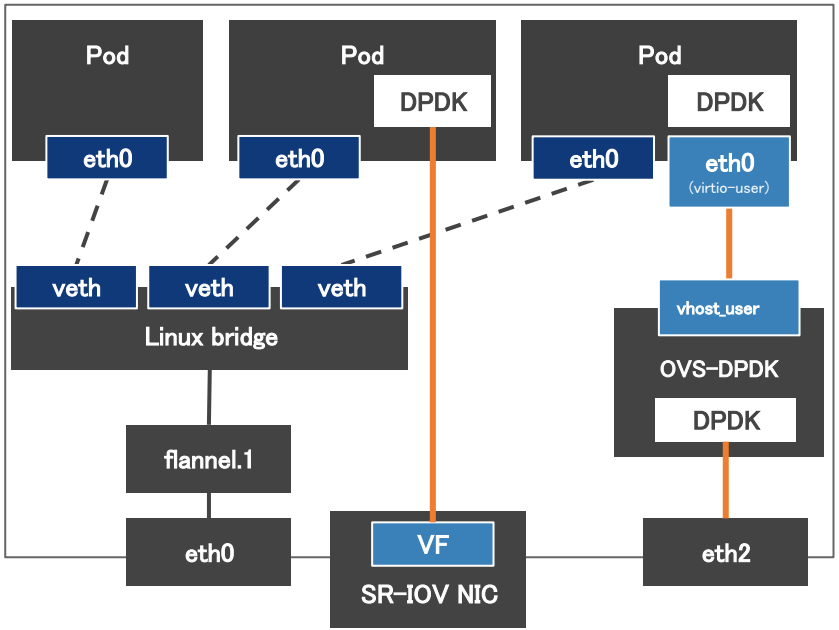
Multus CNIを用いたDPDK利用の実現

SR-IOV CNIプラグイン

- SR-IOV Network Devicプラグインと連携
  - ノード上で利用できるSR-IOV VFを検出
- **検出されたVFを**
  ノード上で実行されるVFを要求するpodに割り当て
- 割り当てられたVFを用いてDPDKを利用

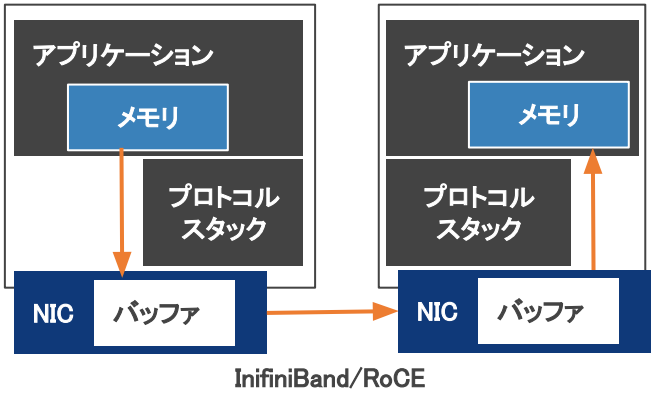
Userspace CNIプラグイン

- DPDKコンテナ・OVS-DPDKのような
  ユーザ空間上でのパケット処理を実現
- **pod作成時に，ノード上のOVS-DPDK bridgeに対して**
  vhost_userインタフェースを追加
  - **コンテナ側に対してもvirtio-userインタフェースを追加**



分散処理・機械学習向けRDMAの適用：FreeFlow[7](1/2)

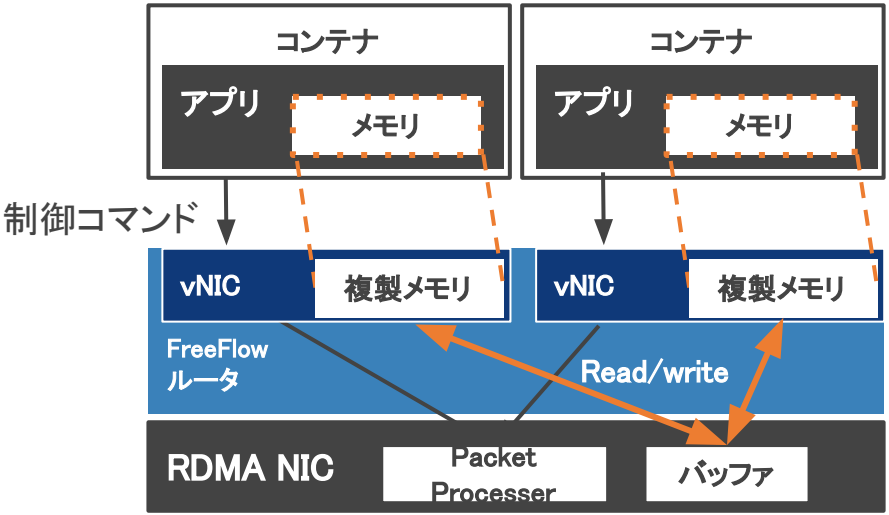
- 2019年のNSDI(ネットワーク系トップ国際会議)の発表
- コンテナのネットワークにRDMAを使うのは難しい
  - ネットワークスタックをバイパスし，
    ネットワーク処理を物理NICにオフロードすることで高性能
  - **複数のコンテナが同時に単一NICでRDMAを使用できない**
- ソフトウェアスイッチで高性能なまま通信を集約したい
  - アドレッシングやルーティングを柔軟に制御可能
  - QoSなどもデータプレーンを上で使用できる
  - **RDMAを使ったコンテナを対象にした**
    ソフトウェアスイッチはない



[7]Kim, D., Yu, T., Liu, H.H., Zhu, Y., Padhye, J., Raindel, S., Guo, C., Sekar, V. and Seshan, S.: FreeFlow: Software-based Virtual RDMA Networking for Containerized Clouds, In 16th USENIX Symposium on Networked Systems Design ar
Implementation (NSDI 19), pp. 113–126(2019).

分散処理・機械学習向けRDMAの適用：FreeFlow[7](2/2)

- **FreeFlowはアプリケーションが使用するメモリを複製し**
  物理NICが複製メモリに対して読み書き
  - アプリケーション・物理NIC間の
    読み書きを仲介することでポリシ制御等を実現
  - メモリの複製はアプリケーションに対して透過
- **SparkとTensorFlowを用いた評価では**
  ベアメタルRDMAとほぼ同じ性能を実現
  - **TCP over RDMAでは，**
    既存のコンテナネットワークと比較して，
    スループットが最大14倍向上

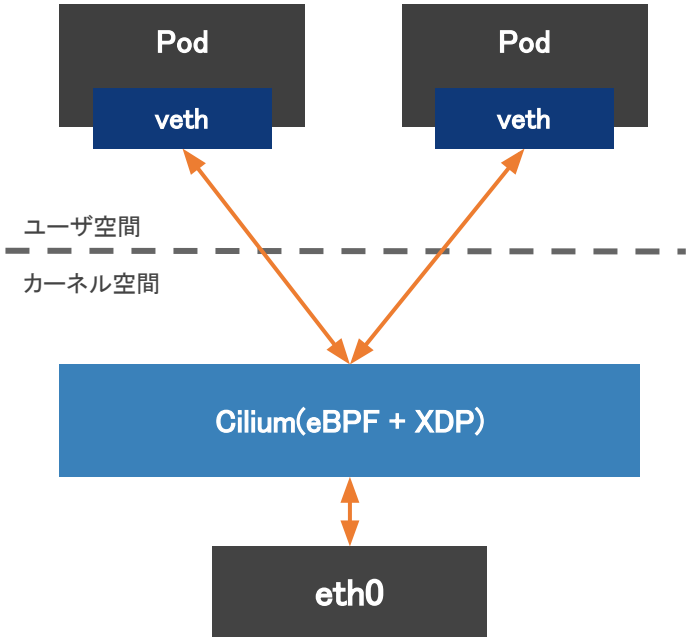


[7]Kim, D., Yu, T., Liu, H.H., Zhu, Y., Padhye, J., Raindel, S., Guo, C., Sekar, V. and Seshan, S.: FreeFlow: Software-based Virtual RDMA Networking for Containerized Clouds, In 16th USENIX Symposium on Networked Systems Design ar
Implementation (NSDI 19), pp. 113–126(2019).

Kubernetesを対象にしたeBPF・XDP適用：Cilium

eBPFを用いてスケーラブルなネットワークと可観測性，高度なセキュリティを実現

- iptablesを置き換え，データパスを最適化
  - iptablesは線形的な探索：O(n)
  - ciliumはeBPFのhashmap：O(1)
  - eBPFによるNAT・デバイス間リダイレクト
  - XDPによるロードバランシング
- Hubble：Cilium・eBPF上の監視ツール
  - 依存関係・フロー情報の可視化・アプリケーション監視
- L3/4/7ポリシー・透過的な暗号化を提供
  - vethでパケットをフックしてポリシー・暗号化を強制



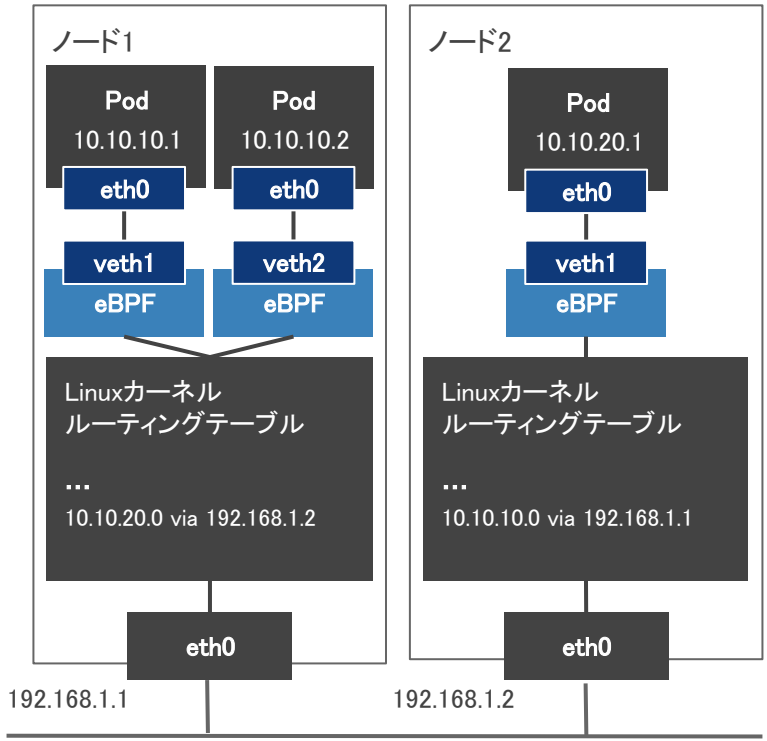
Ciliumにおけるルーティング方式

1. カプセル化(VXLAN，デフォルト)

- **クラスタネットワークがシンプル**
- カプセル化によってMTUが減少

2. ネイティブ

- 別のノード宛パケットを
  Linuxカーネルのルーティングにリダイレクト
  - ノード内プロセスがパケットを送信するようにルーティング
- **各ノードは，全ノードのpodやワークロードに**
  パケットを転送する方法を認識する必要
  - BGPなどを用いて他ノードのpodなどのIPを認識
  - 到達方法を知っているルータを配置



ネイティブ方式ルーティング

Ciliumにおけるデータパス最適化例：DSR(Direct Server Return)

- ノードから別ノードにリクエストがリダイレクトする事があるリソース

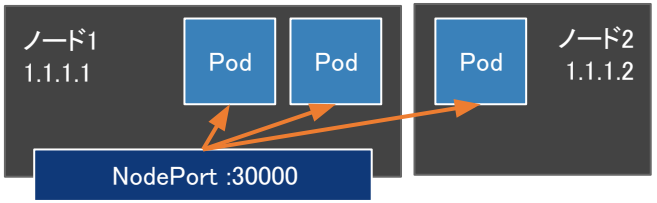
  e.g., NodePort, ExternalIP, LoadBalancer

  - 宛先がリクエストの送信先と異なるノードで実行される際に発生

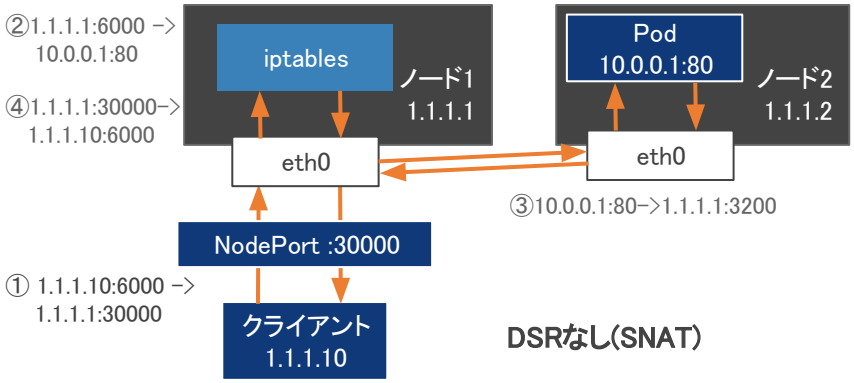
- **DSRなし：リダイレクト前にSNATで変換**

- **DSR：eBPFで応答時に送信元をリダイレクト元アドレスに変換**

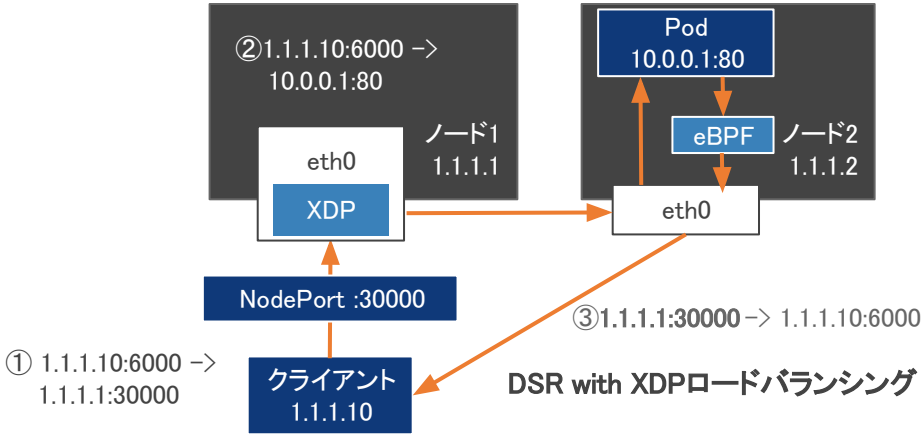
  - **別ノードへのリダイレクトはXDPを使用可能**



リダイレクト例：NodePort



DSRなし(SNAT)

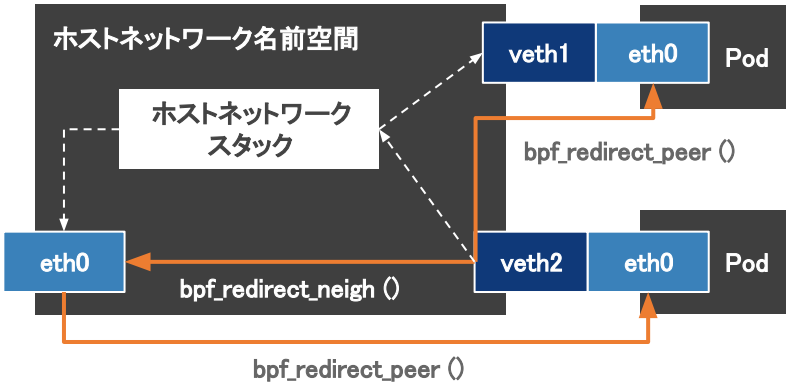


DSR with XDPロードバランシング

Ciliumにおけるデータパス最適化例：NIC間通信

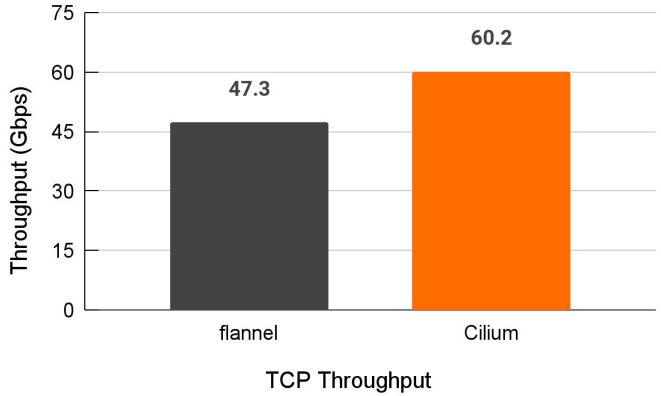
Linuxカーネル5.10(2020年12月)で追加されたeBPFのヘルパーを使用して
ホストのネットワークスタック処理を削減

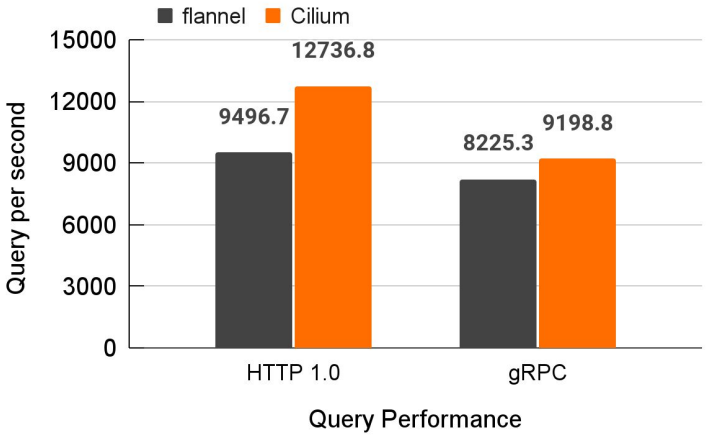
- bpf_redirect_peer
  - ノード内通信・別ノードからのパケット受信に使用
  - 指定したifindex(一意なNICのID)のvethデバイスのペアにパケットをリダイレクト
- bpf_redirect_neigh
  - 別ノードへのパケット送信に使用
  - 指定したifindexのデバイスにパケットをリダイレクト
  - ネクストホップの宛先を決定するために，
    skbのネットワークヘッダ情報からルーティングテーブルを検索



性能評価：単一ノード内におけるpod間通信性能

- **スループットとクエリ性能を評価**
  - iperf3を用いてTCPスループットを計測
  - fortioを用いて1秒あたりに処理できた HTTP1.0・gRPCのクエリ数を計測
- 比較対象：
  - flannel：0.12.0
  - Clium：1.9.8
- **CiliumはTCPのスループットを約27%向上**
- **HTTP1.0のクエリ性能は約34%, gRPCのクエリ性能は約11%向上**





まとめ

DPDK：カーネルをバイパスした高速パケット処理を実現

- UIO/VFIOを用いてユーザ空間からハードウェアを直接制御
- PMDによるポーリングでパケット受信を監視・CPUのコア占有
- HugepagesによるTLBヒット率向上
- **コンテナでのDPDK利用ではvirtio-user/vhost-user，SR-IOVを使用**

XDP：Linuxカーネル内に実装された高速パケット処理基盤

- NICドライバにeBPFプログラムをアタッチ
- Linuxの機能を活用したまま，高速パケット処理が実現
- **Cilium：eBPFとXDPを用いてiptablesを置き換え，データパスを最適化**

皆で今日から高速にパケットを処理して優勝！！！！