NS研究会
2019/6/21

NFVノードにおけるパケット I/O 性能向上のためのデータパス並列化

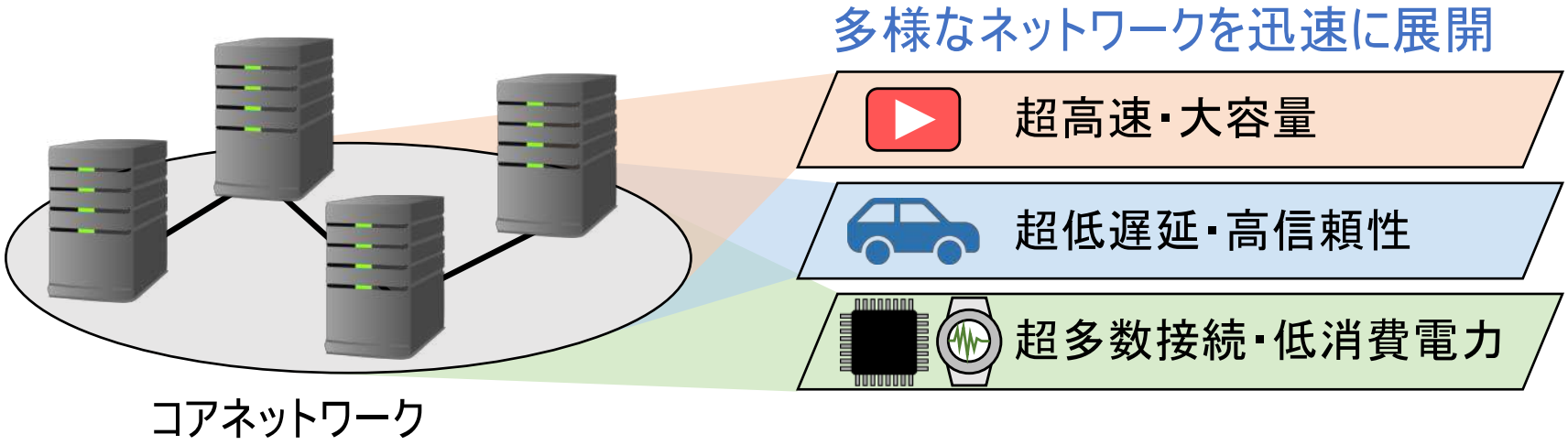
浅田 雅裕[†]　川島 龍太[†]
中山 裕貴[††]　林 經正[††]　松尾 啓志[†]

[†] 名古屋工業大学
[††]株式会社ボスコ・テクノロジーズ

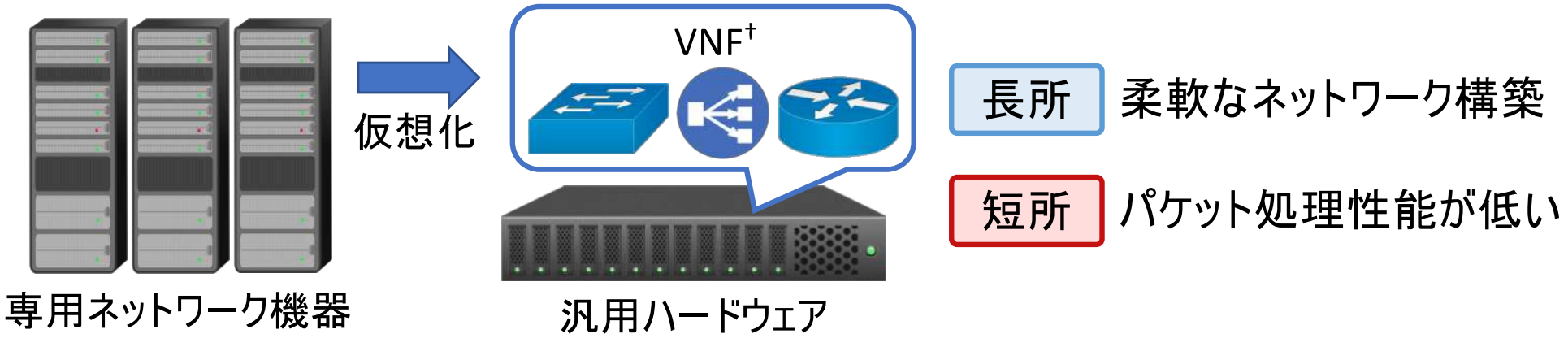
目次

5G / NFV

5G基幹網への要求



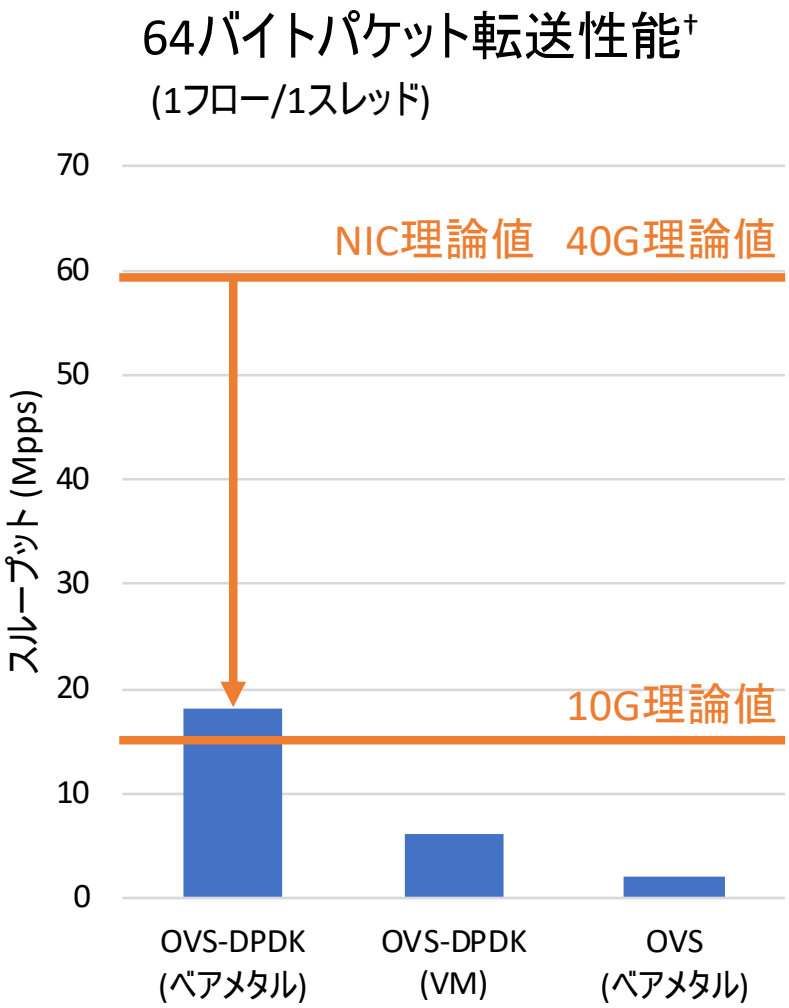
NFV



† Virtual Network Function

VNFのパケット処理性能 (1/2)

- >10 Gbpsの性能達成は困難



- 性能差はさらに深刻化
  - 100G～テラビット・イーサネットの登場

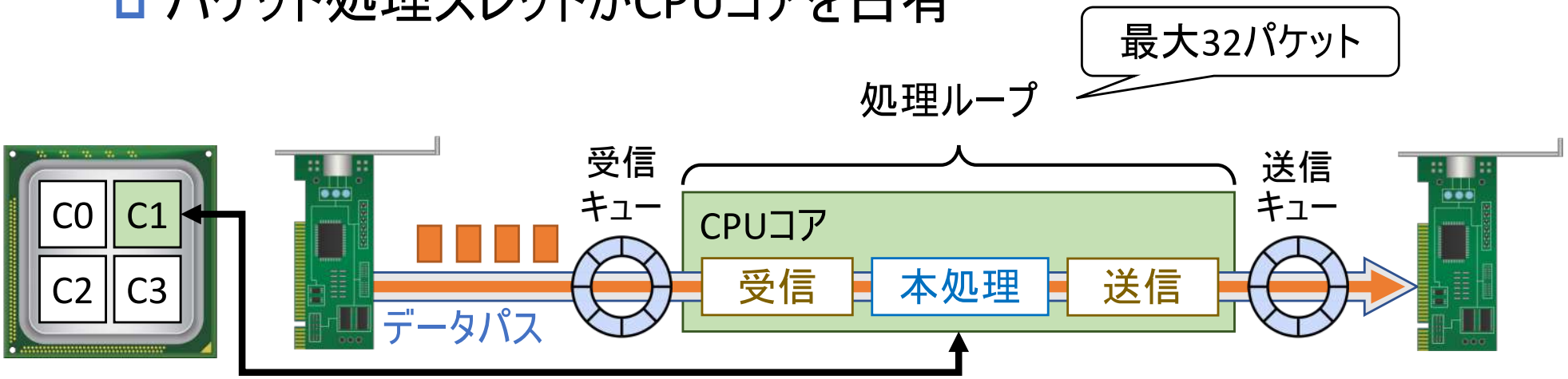
NFVノードの性能向上が不可欠

† Evaluation of Forwarding Efficiency in NFV-nodes toward Predictable Service Chain Performance.
R. Kawashima et al.
IEEE Transactions on Network and Service Management, 2017

DPDK: Data Plane Development Kit
OVS: Open vSwitch

VNFのパケット処理性能 (2/2)

- 一般的なパケット処理工程
  - パケット処理スレッドがCPUコアを占有



- 単一スレッドの性能向上は困難

64バイトパケット
100 Gbps

→

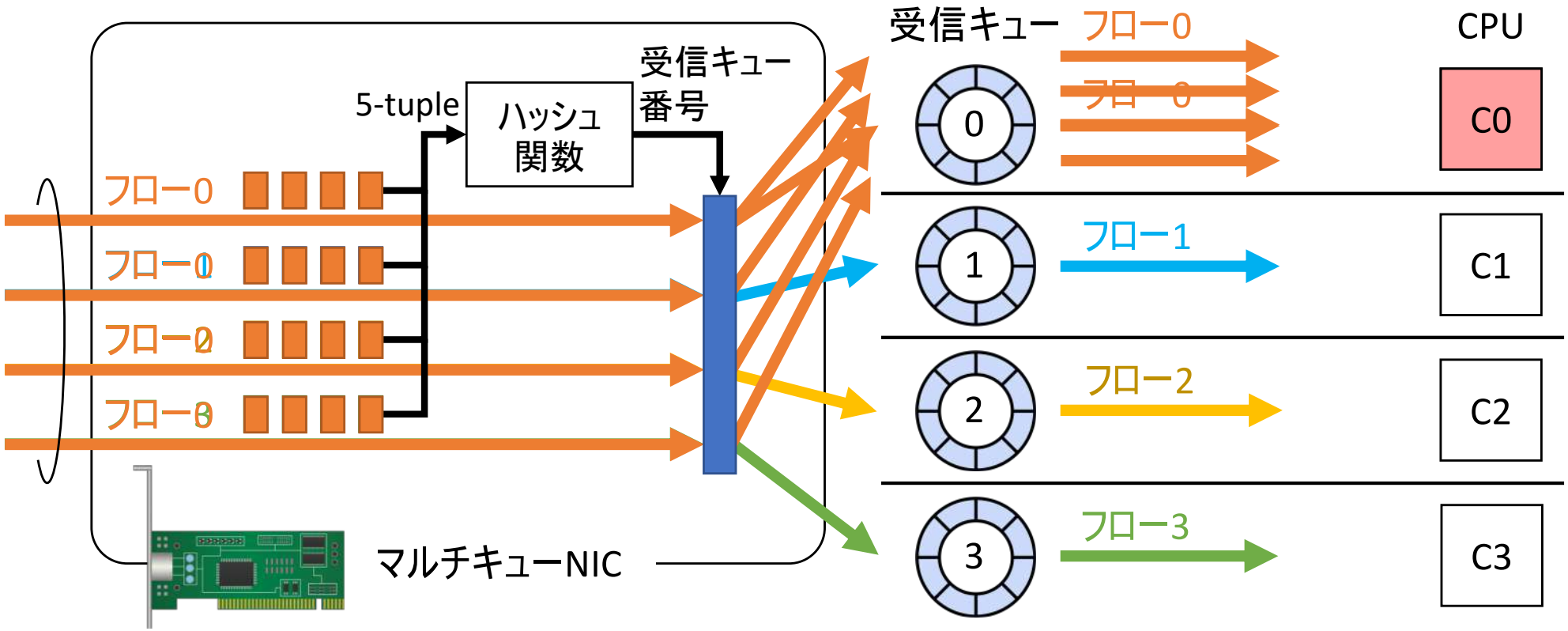
処理時間: 5.12 ns/パケット
CPUサイクル数: ≦17 cycles/パケット ††

†† Make the most out of last level cache in intel processors.
A. Farshin et al.
Proceedings of the Fourteenth EuroSys Conference, 2019

処理並列化が必須

フロー単位での処理並列化

- RSS: Receive Side Scaling
  - 5-tupleのハッシュ値を基に複数CPUコアに負荷分散



- ハッシュ衝突が多発 → 性能向上が不十分†
  - 単一フローの処理並列化不可

フローよりも細かい
粒度での並列化が必須

† A Case for Spraying Packets in Software Middleboxes.
H. Sadok et al.
Proceedings of the 17th ACM Workshop on Hot Topics in Networks - HotNets '18

目次

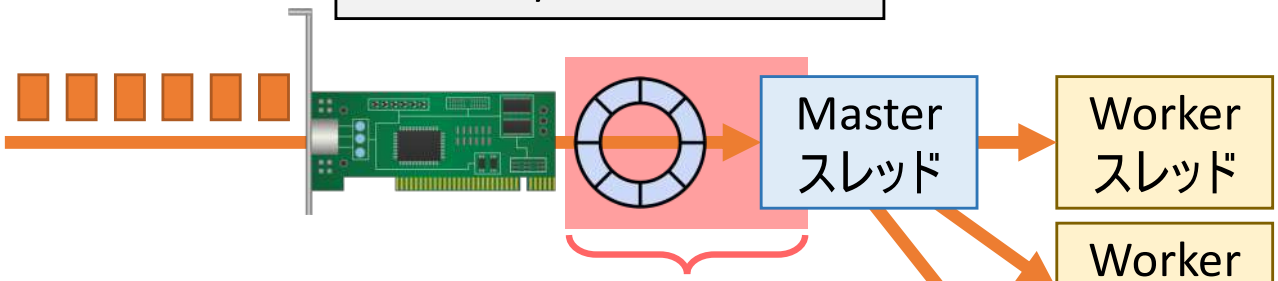
- 研究背景
- **関連研究**
- 提案手法・実装
- 評価
- 考察・まとめ

マルチコア環境におけるパケット処理 (1/2)

- 複数のパケット処理モデルを実装・評価†

RSSモデル　RSSを使用して並列化

Master/Workerモデル



受信処理が並列化されず

I/O部分がボトルネック

Masterが転送先Workerを決定

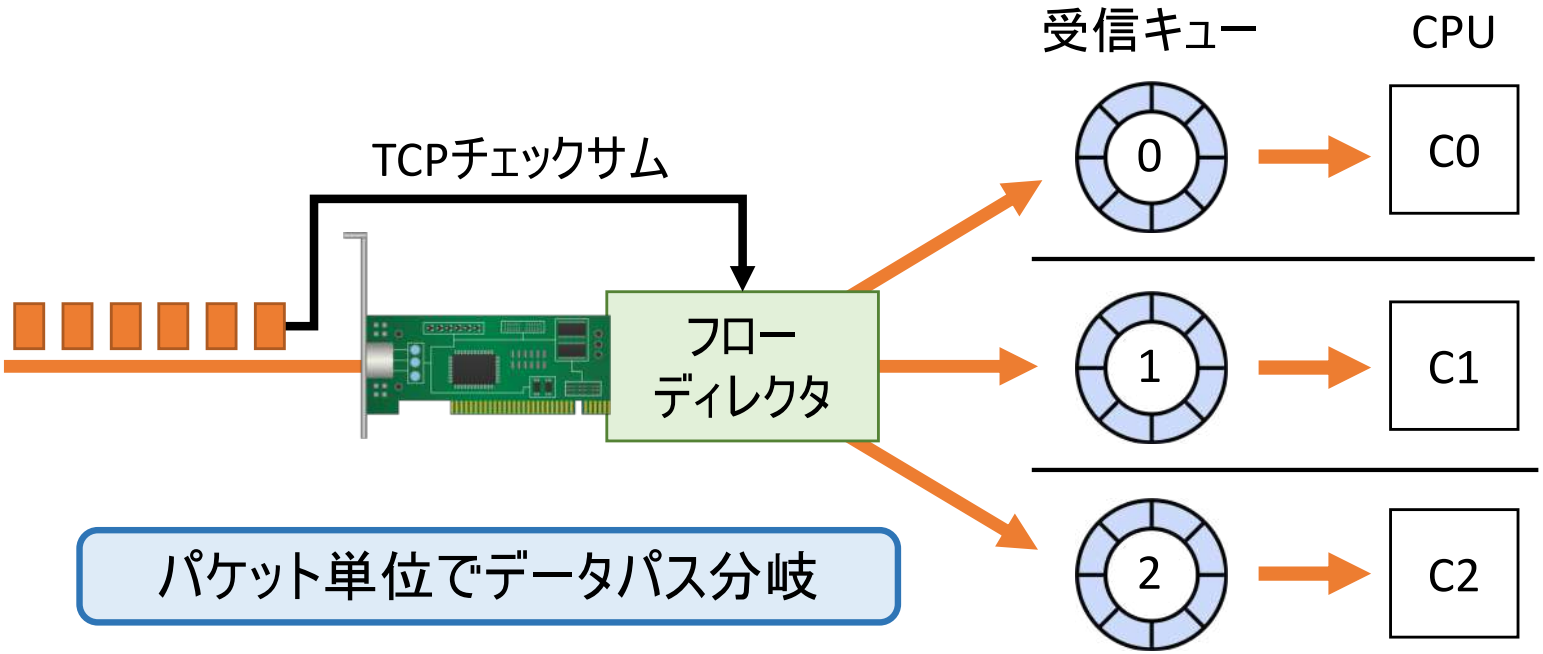
フロー非依存な
データパス分岐

I/O含む処理並列化が必須

† Designing Virtual Network Functions for 100 GbE Network Using Multicore Processors.
P. Li et al.
2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)

マルチコア環境におけるパケット処理 (2/2)

- TCPチェックサムを用いたデータパス分岐†
  - フローディレクタによる振り分け



パケット単位でデータパス分岐

並列化がTCPトラフィックに限られる

順序逆転に起因するオーバヘッド

特定のプロトコルに
依存しない設計

順序制御機構の付加

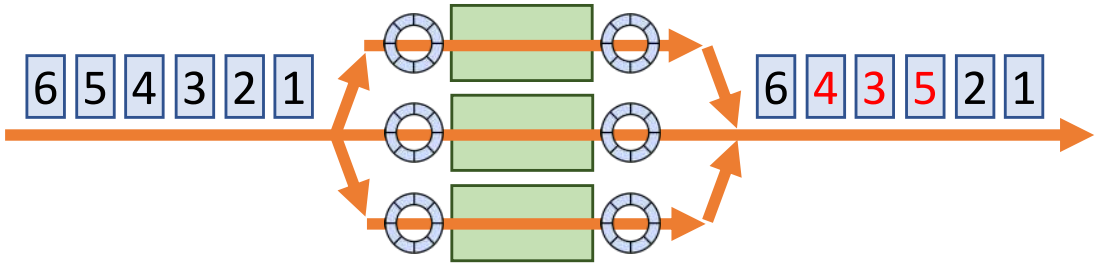
† A Case for Spraying Packets in Software Middleboxes.
H. Sadok et al.
Proceedings of the 17th ACM Workshop on Hot Topics in Networks - HotNets '18

目次

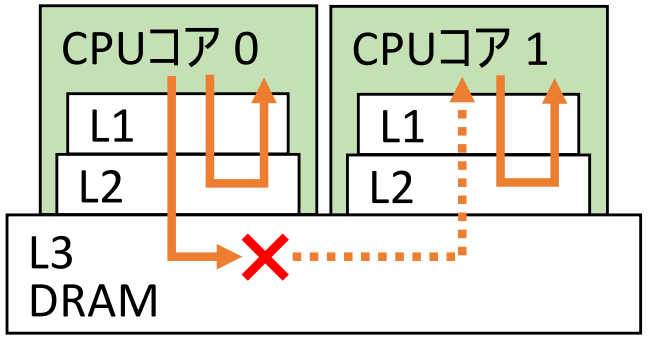
- 研究背景
- 関連研究
- **提案手法・実装**
- 評価
- 考察・まとめ

並列化要件

- 性能向上のための要件
  - I/Oを含む並列化
  - フロー・プロトコル非依存なデータパス分岐
    - RSSの制約を補完
    - 全てのトラフィックが並列化対象
    - パケット列の順序制御の必要性

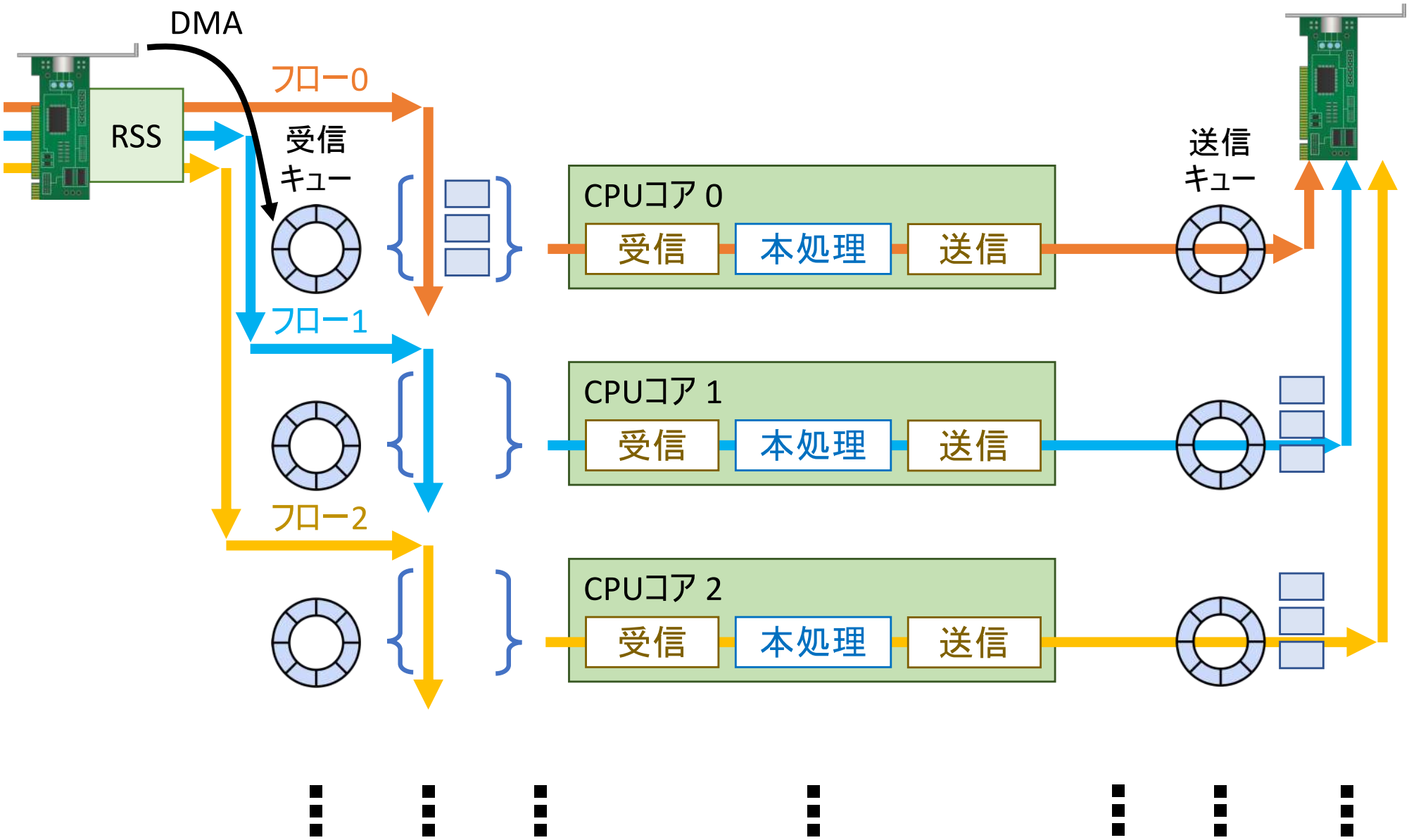


  - CPUコア間通信の排除
    - L3キャッシュ / DRAMアクセスは低速
    - 高速なL1/L2キャッシュを有効利用†

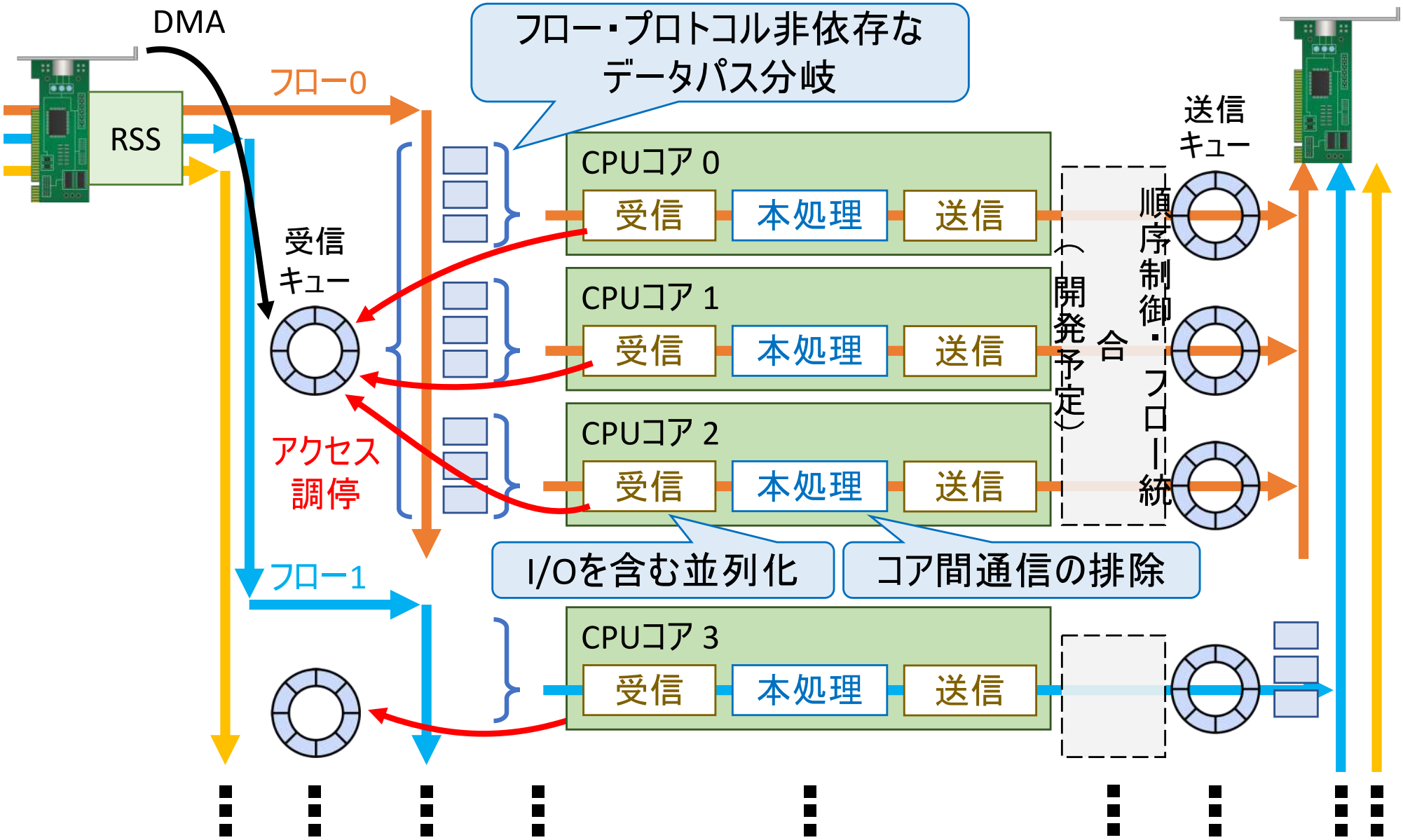


† Metron: NFV Service Chains at the True Speed of the Underlying Hardware.
G. Katsikas et al.
USENIX Symposium on Networked Systems Design and Implementation, 2018

従来の構成

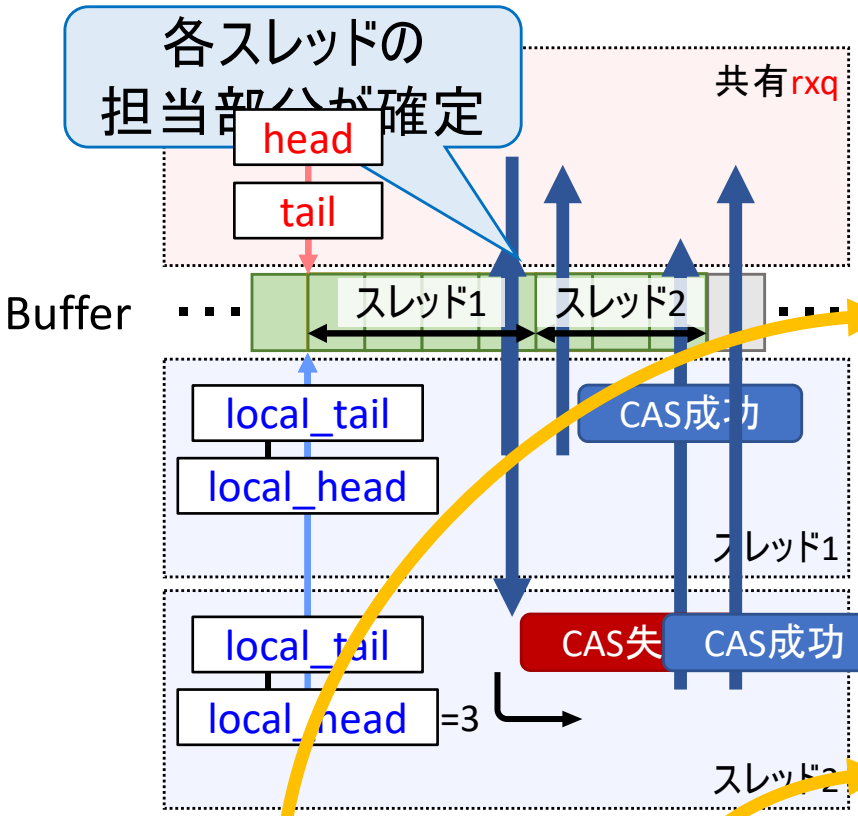

DMA
RSS
フロー0
受信
キュー
フロー1
フロー2
CPUコア 0
受信
本処理
送信
CPUコア 1
受信
本処理
送信
CPUコア 2
受信
本処理
送信
送信
キュー

提案手法の全体構成


DMA
RSS
フロー0
フロー1
フロー・プロトコル非依存な
データパス分岐
受信
キュー
アクセス
調停
CPUコア 0
CPUコア 1
CPUコア 2
CPUコア 3
受信
本処理
送信
I/Oを含む並列化
コア間通信の排除
（開発予定）
順序制御・フロー統合
送信
キュー

アクセス調停の実装

- Mellanox製NIC向けデバイスドライバを改変
  - パケット受信関数をスレッドセーフ化
- バッファ操作用の変数をCAS命令で更新



赤: スレッド間で共有
青: 各スレッドが
個別に保持

```
rxq = 操作対象の受信キュー;
do {
  local_tail = rxq->head;
  n = min(到着パケット数, 32);
  local_head = local_tail + n;
  success = cas(&rxq->head,
                local_tail, local_head);
} while (success == CAS_FAILED);

/* Buffer[local_tail:local_head]を処理 */
/*   従来実装と同様                    */

while (rxq->tail != local_tail) { }
rxq->tail = local_head
```

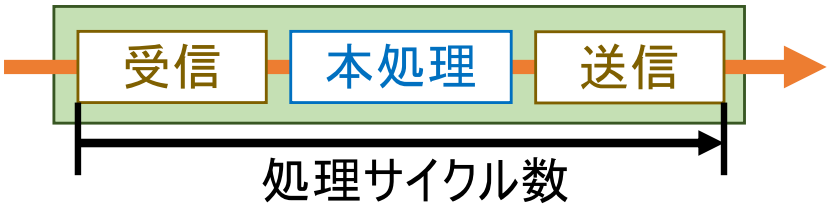
- CASループ , whileループ の実行時間 … アクセス競合指標

目次

- 研究背景
- 関連研究
- 提案手法・実装
- 評価
- 考察・まとめ

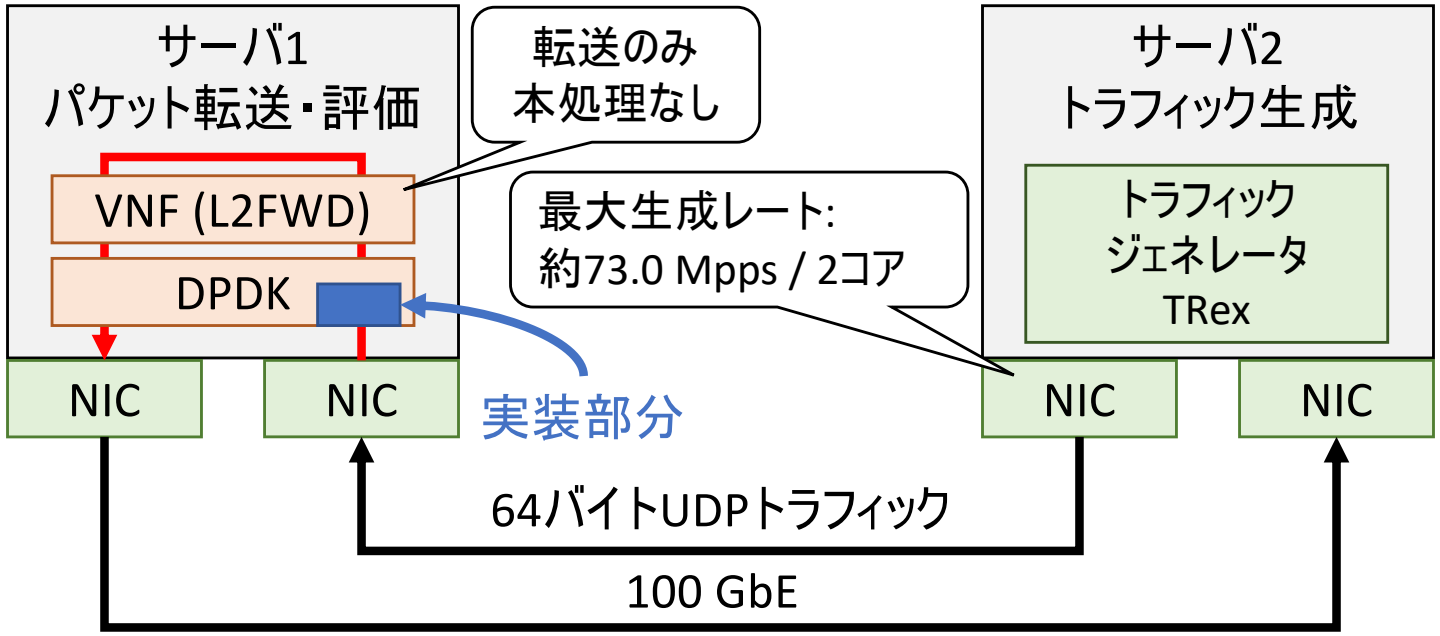
評価項目

- CAS，whileループの処理サイクル数
  - アクセス競合の度合いを確認
- スループット・パケット処理サイクル数
  - 並列化によるパケット処理性能への影響を確認



- 順序逆転
  - 並列化に起因する順序逆転度合いを確認

評価環境



	サーバ1	サーバ2
CPU	Intel Core i9-7940X @3.10GHz 14コア (HT無効)	Intel Core i5-4570 @3.20GHz 4コア (HT無効)
メモリ	8 GB × 4 DDR4	8 GB × 2 DDR3
NIC	Mellanox Technologies ConnectX-5 Ex 100 GbE Dual-Port	
OS	CentOS 7.6	CentOS 7.5
DPDK	v19.02	v18.08
TRex	—	v2.56

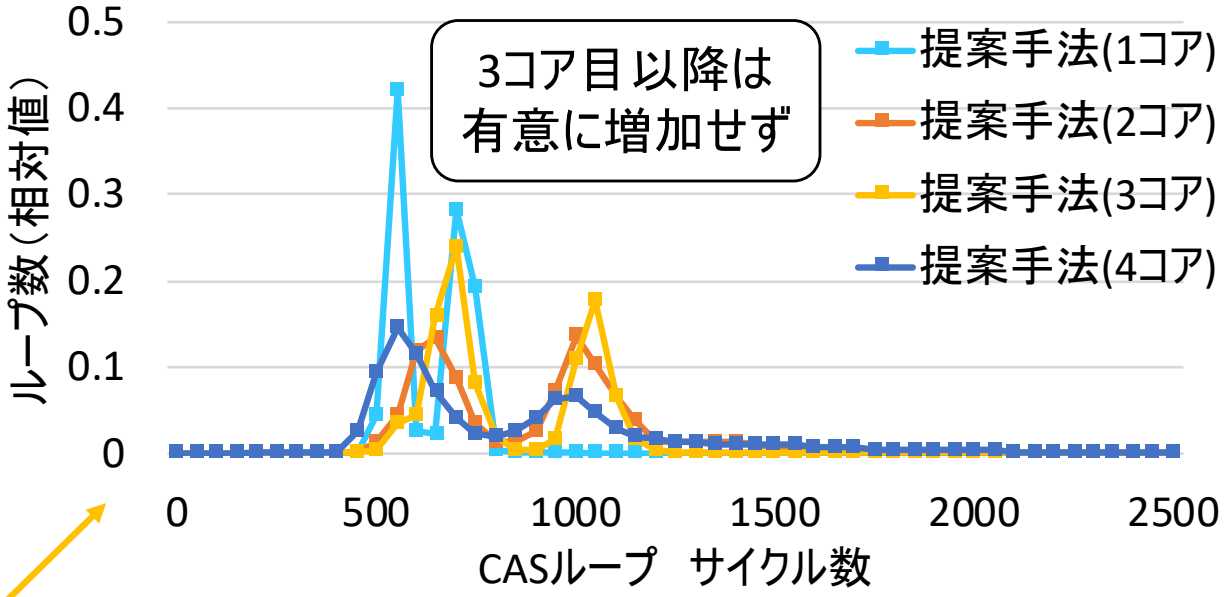
CAS，whileアクセス競合

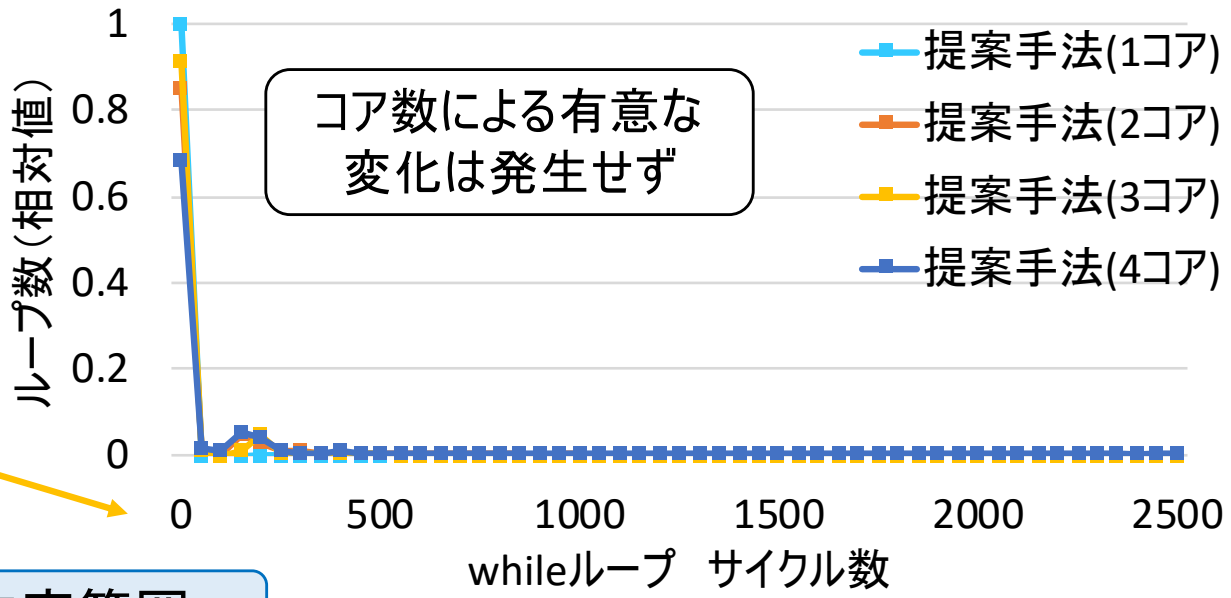
注) 全ループの合計回数を
1に正規化

CAS	1コア	2コア	3コア	4コア
平均	658.6	911.3	860.2	904.3
増加率	1.00	1.38	1.31	1.37

調停コード(再掲)

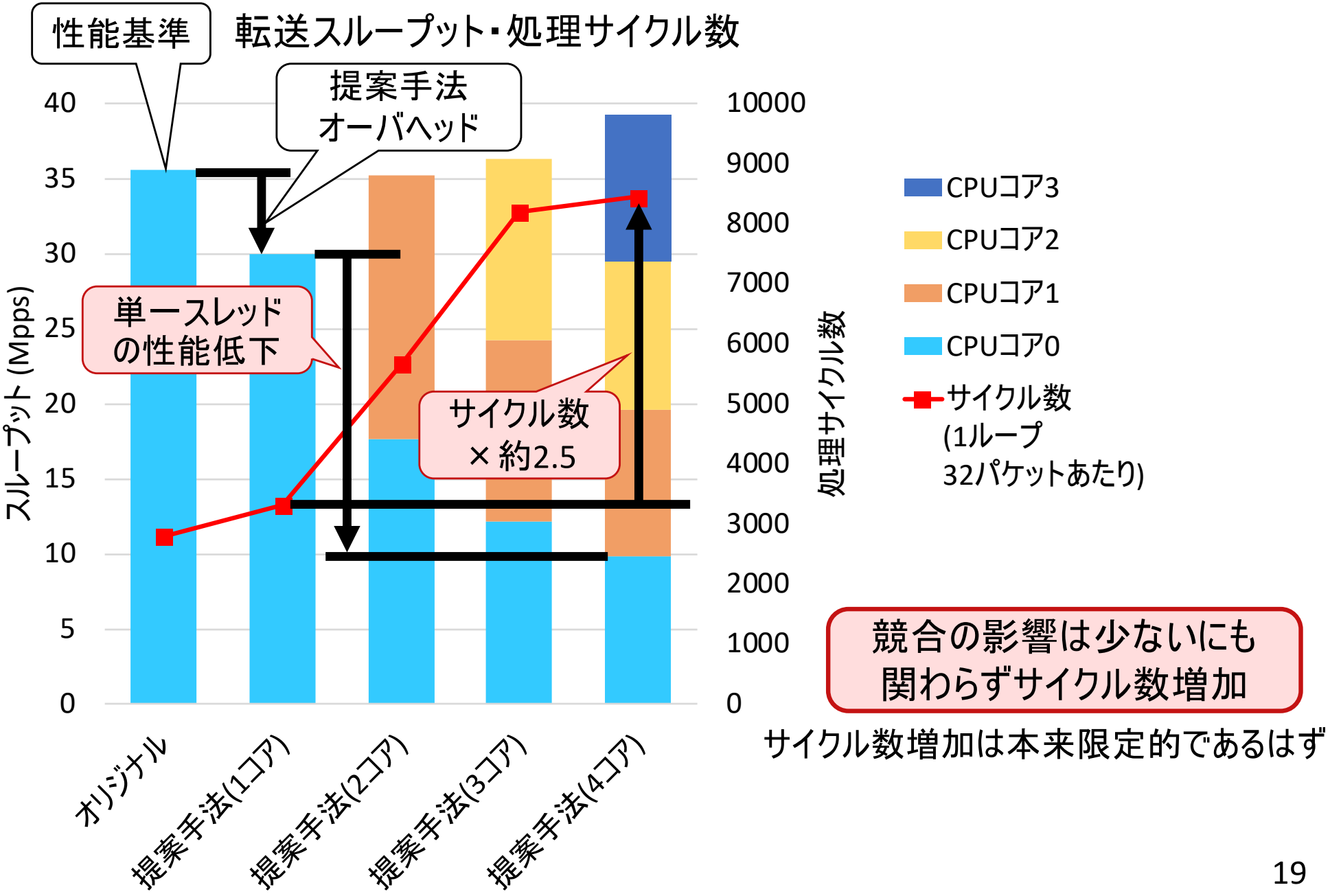
```
rxq = 操作対象の受信キュー;
do {
  local_tail = rxq->head;
  n = min(到着パケット数, 32);
  local_head = local_tail + n;
  success = cas(&rxq->head,
                local_tail,
                local_head);
} while (success == CAS_FAILED);

/* Buffer[local_tail:local_head] */
/* を処理                        */
/*   従来実装と同様              */

while (rxq->tail != local_tail) { }
rxq->tail = local_head
```





アクセス調停のオーバヘッドは許容範囲

スループット・処理サイクル数



競合の影響は少ないにも
関わらずサイクル数増加

サイクル数増加は本来限定的であるはず

目次

- 研究背景
- 関連研究
- 提案手法・実装
- 評価
- 考察・まとめ

考察 ― I/O性能

- スループット
  - 35.51 Mpps (オリジナル) → 39.26 Mpps (提案手法4コア)
  - スレッドあたりスループットが低下
  - 送受信キューあたりの性能頭打ちの可能性
    - DMA帯域不足など

トラフィック生成側では送信キューを2つ使用

- 提案手法の特性
  - 明確なサイクル数増加要因はアクセス調停部分のみ
    ←調停オーバヘッドは許容範囲
    コア数4倍に対しCAS競合1.37倍

- 自明ではないサイクル数増加要因
  - プログラム記述上は想定されない競合（干渉）の存在
    - 次スライド参照

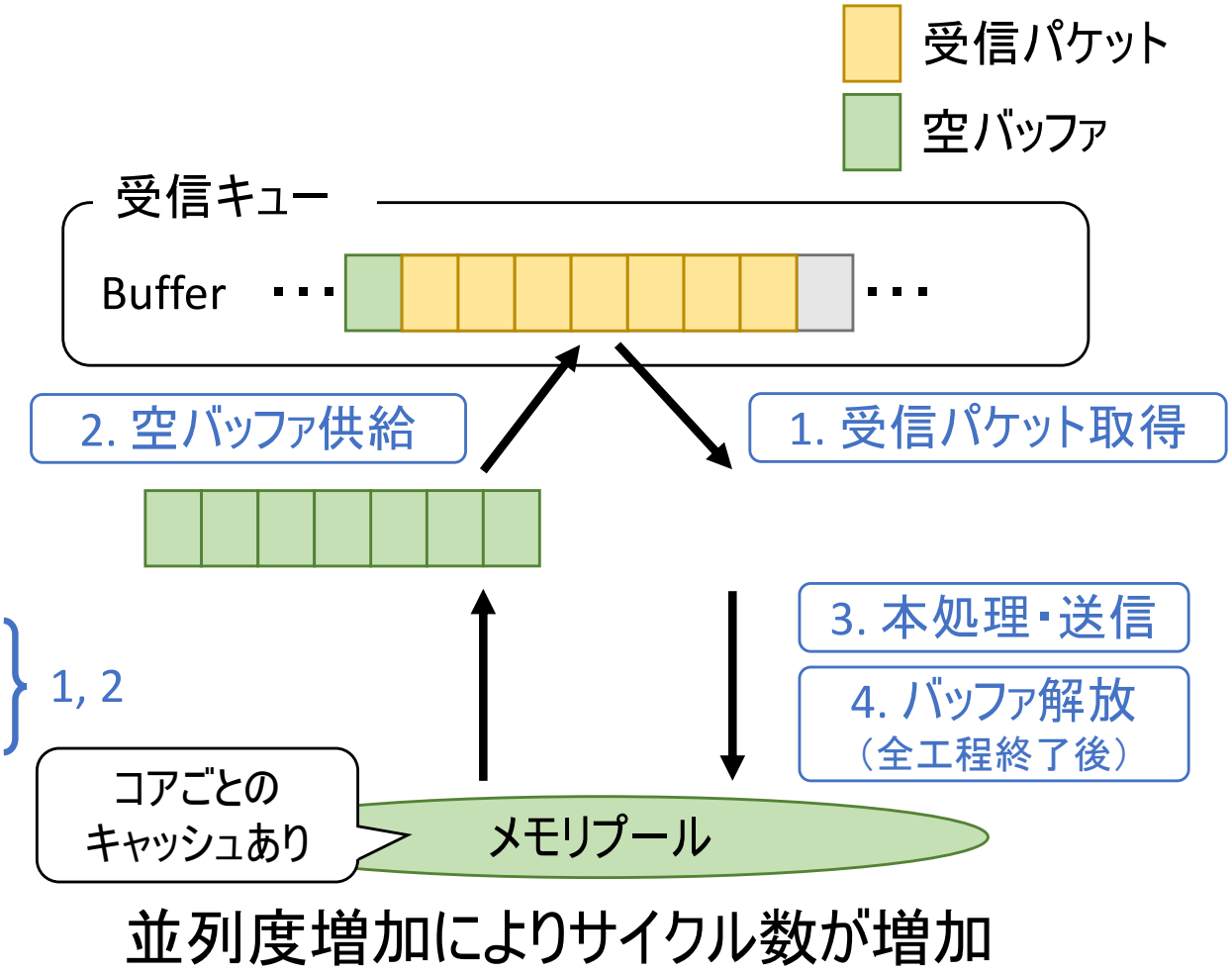
非自明な競合の存在可能性

- 記述上は競合なし

調停コード（再掲）

```
rxq = 操作対象の受信キュー;
do {
  local_tail = rxq->head;
  n = min(到着パケット数, 32);
  local_head = local_tail + n;
  success = cas(&rxq->head,
                local_tail,
                local_head);
} while (success == CAS_FAILED);

/* Buffer[local_tail:local_head] */
/* を処理                        */
/*   従来実装と同様              */

while (rxq->tail != local_tail) { }
rxq->tail = local_head
```

1, 2

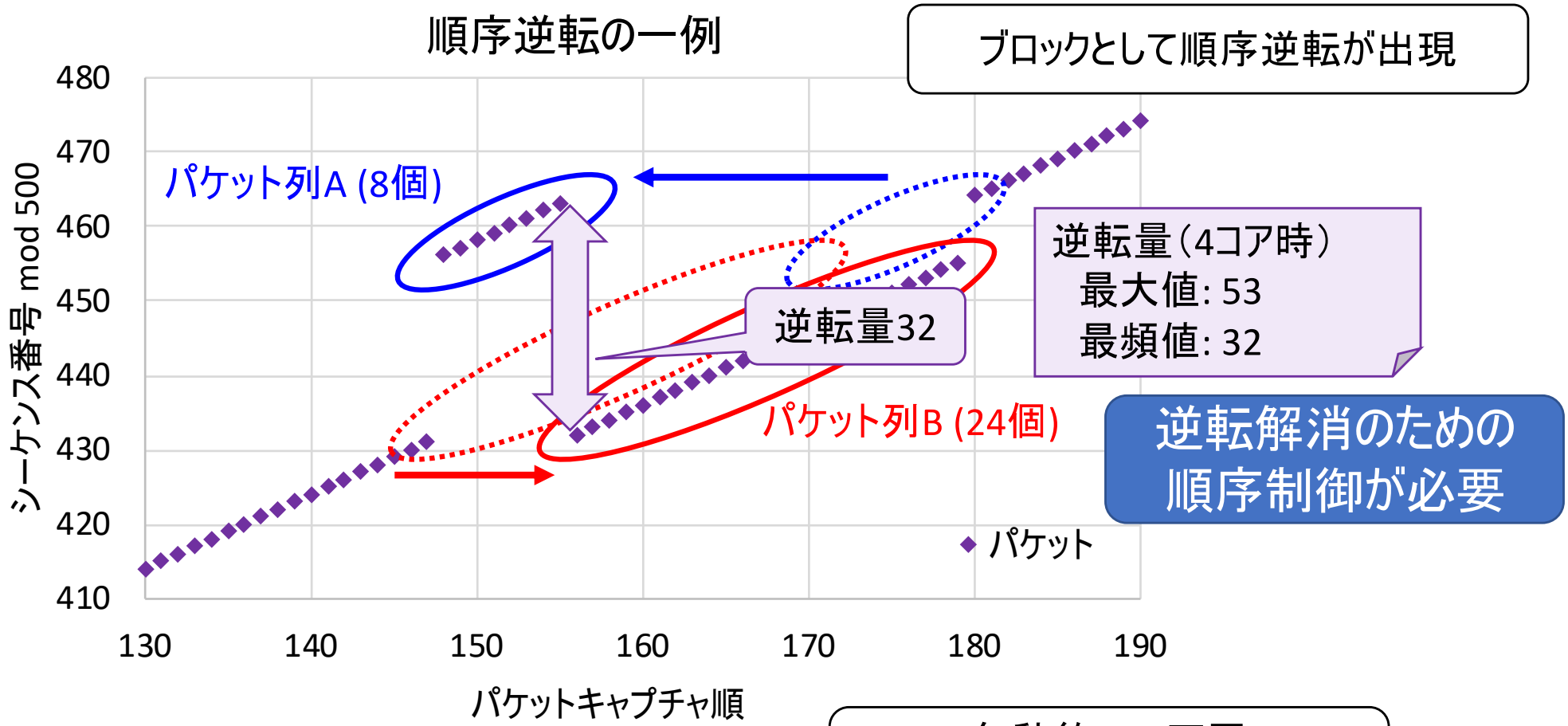


並列度増加によりサイクル数が増加

提案手法では処理サイクル数が性能に直結
→サイクル数増加要因を特定・改善すれば並列化効果は期待可能

順序逆転

- ペイロードにシーケンス番号を付加して巡回
  - パケットキャプチャし順序を観測



毎秒約22.7万回
(パケット列全体中の約3.4%)

考察 — 順序制御機構の実装にむけて

- 順序逆転はブロック単位
  - 順序制御の単位をブロックとする → 制御の効率化
- 順序の定義
  - タイムスタンプ
    - 受信時に各パケットに付加
  - 受信キューのインデックス
    - パケットは受信順でキューに格納される

約78.2 Mppsまではパケット単位で順序識別可能 (ConnectX-5)

- 実装手段・方針
  - ハードウェア支援の活用　—　FPGA, Smart NICなど
  - 順序の厳密さ < スループット　—　オーバヘッド削減が目的

まとめと今後の課題

まとめ

- データパス分岐によるパケットI/O並列化を提案
  - 受信キューへのアクセスをスレッドセーフ化
  - アクセス調停のオーバヘッドは許容範囲
  - 現状では性能向上に課題あり

今後の課題

- 処理サイクル数の増加要因の特定・改善
- 順序制御機構の具体的実装