P4による
GTP Packet Brokerの開発

2021年10月21日

ソフトバンク株式会社

熊谷　渉

目次

- GTPとPacket Brokerの概要
- Data Plane実装
- Control Plane実装
- まとめ

自己紹介

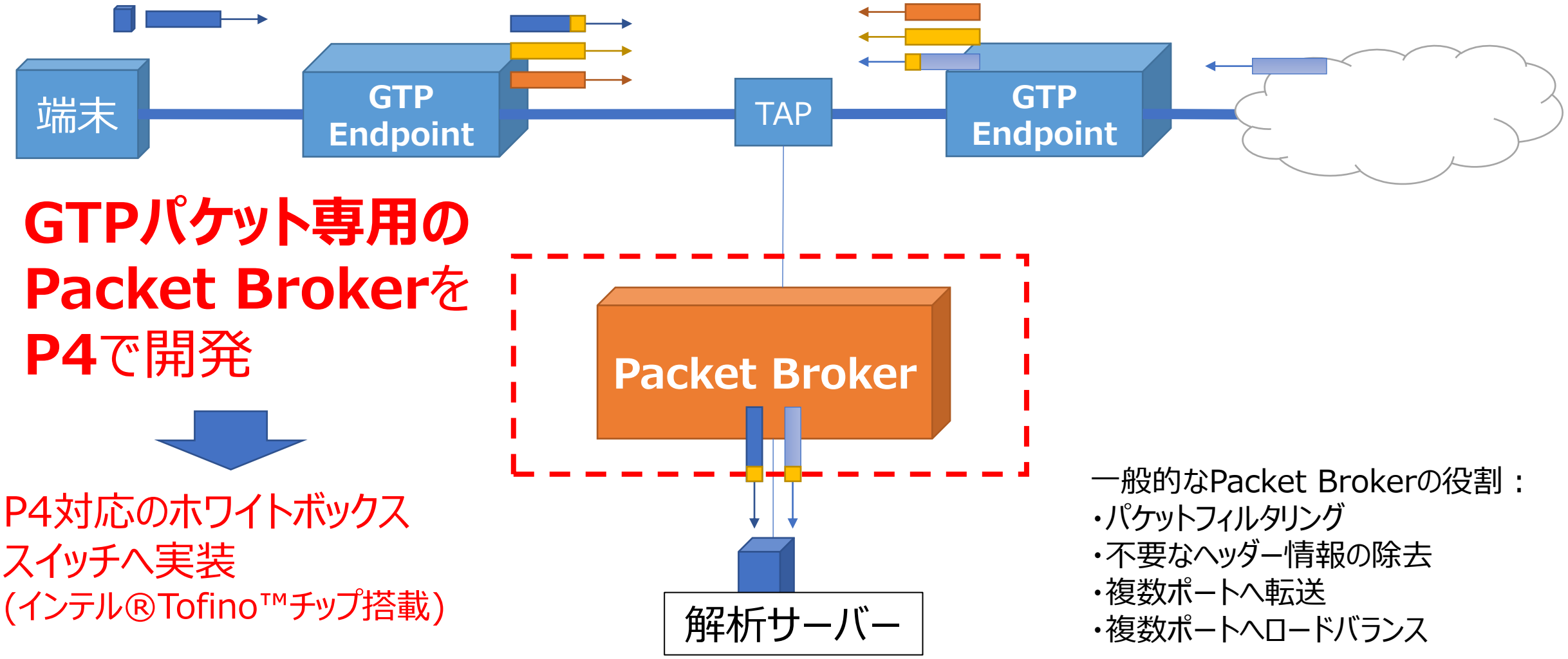
- 熊谷 渉(くまがい わたる)
- ソフトバンク株式会社 IT-OTイノベーション本部

- 経歴
  - 2005～2020年：NW機器ベンダーでスイッチ/ルーターのソフトウェア開発
  - 2021年～：モバイルNW向けNFVソフトウェアの開発(現職)
- P4に関する講演
  - 2018/07 JANOG45 P4使ってみた
    - https://www.janog.gr.jp/meeting/janog42/program/SP-P4
  - 2020/01 JANOG48 P4+SRv6
    - https://www.janog.gr.jp/meeting/janog45/program/srv6sfc

目次

- **GTPとPacket Brokerの概要**
- Data Plane実装
- Control Plane実装
- まとめ

GTP Packet Broker 概要



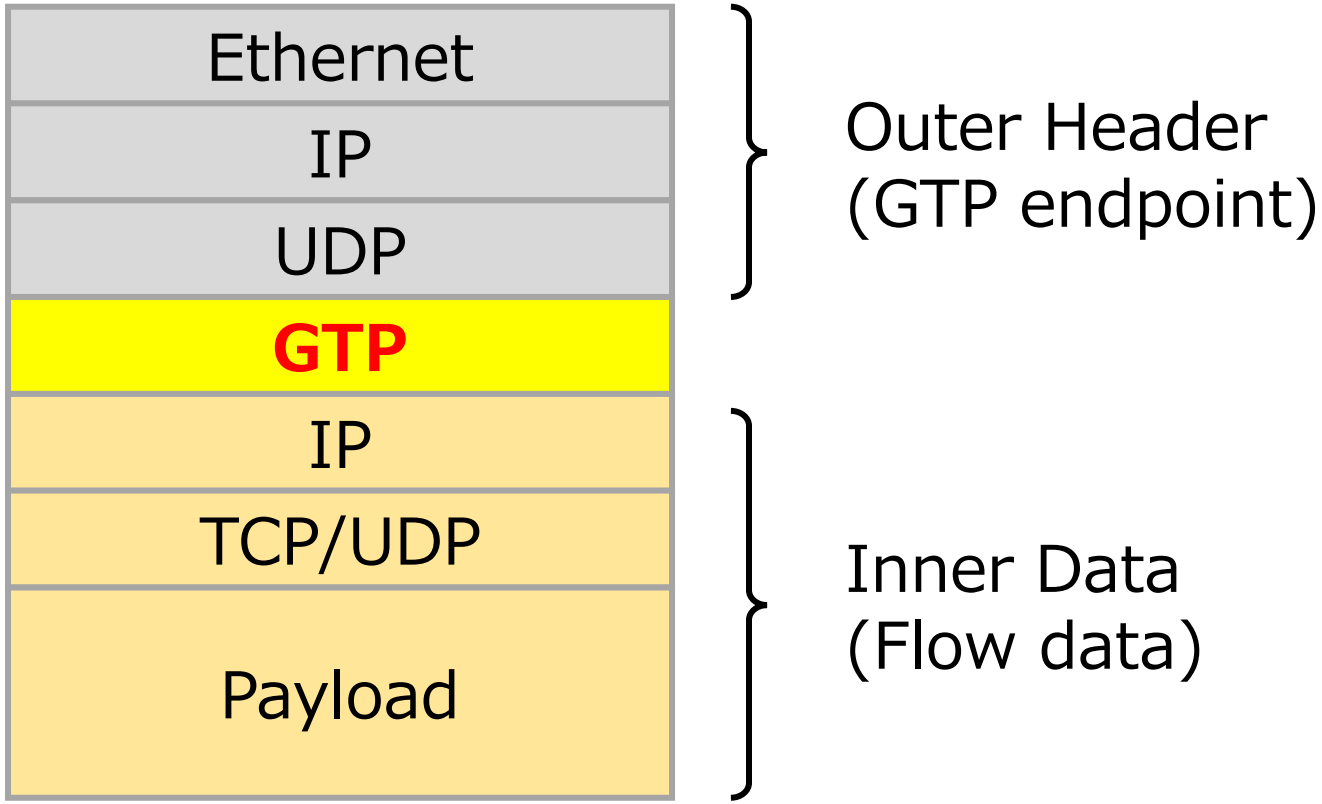
GTPパケット専用の Packet Brokerを P4で開発

P4対応のホワイトボックス スイッチへ実装
(インテル®Tofino™チップ搭載)

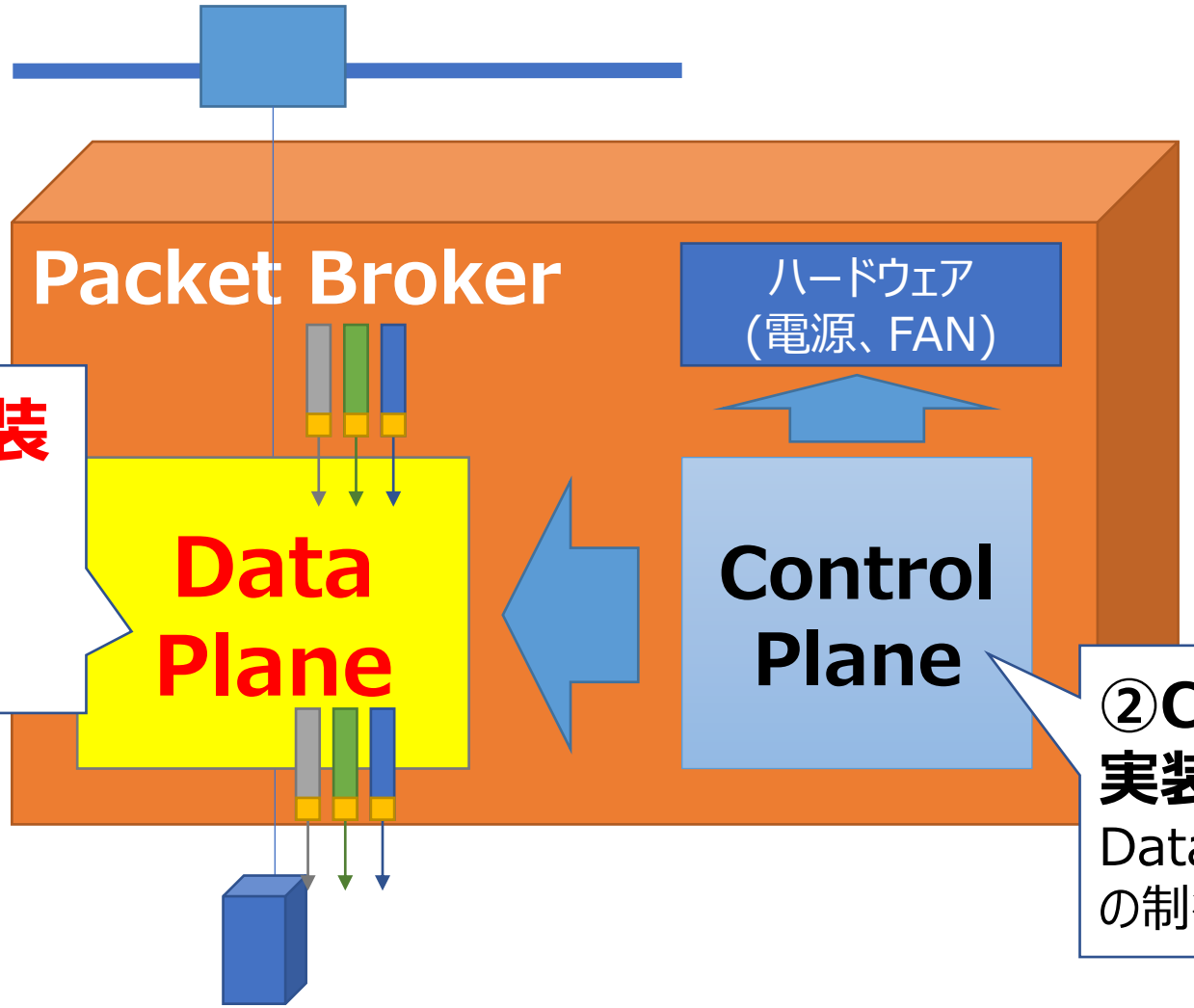
一般的なPacket Brokerの役割：
・パケットフィルタリング
・不要なヘッダー情報の除去
・複数ポートへ転送
・複数ポートへロードバランス

GTPプロトコル概要

- GTP(GPRS Tunneling Protocol)
- モバイル網で使用されるIPベースのトンネリングプロトコル



GTP Packet Brokerの開発



①Data Planeの実装
パケットのヘッダ解析および
転送処理
→**P4で実装**

②Control Planeの実装
Data Plane、ハードウェアの制御処理

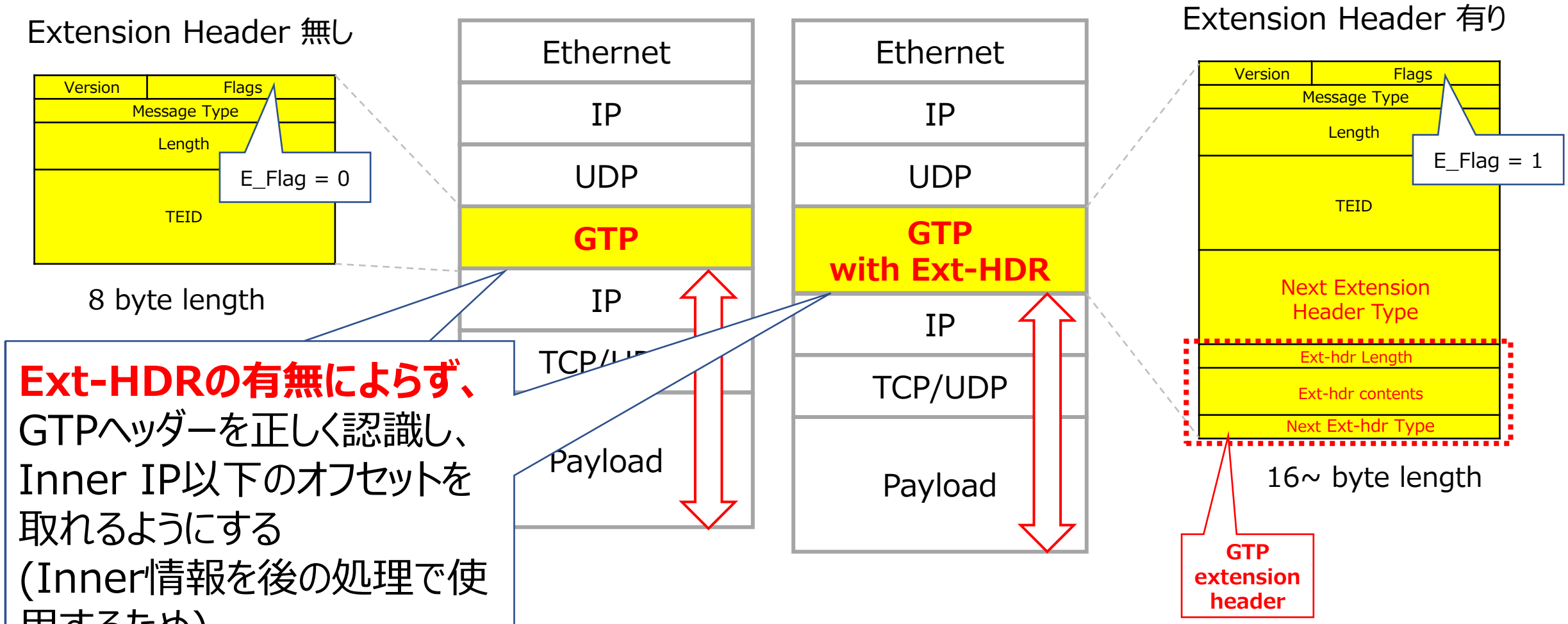
目次

- GTPとPacket Brokerの概要
- **Data Plane実装**
- Control Plane実装
- まとめ

GTP Packet Brokerの要件

1. **GTPヘッダーのParse処理**
2. **フロー単位トラフィック分散**
3. **サーバーの複数CPUコアへのフロー単位トラフィック分散**
4. Outer header でのフィルタ(ACL)
5. VLAN tag の除去
6. 受信時刻の Timestamp

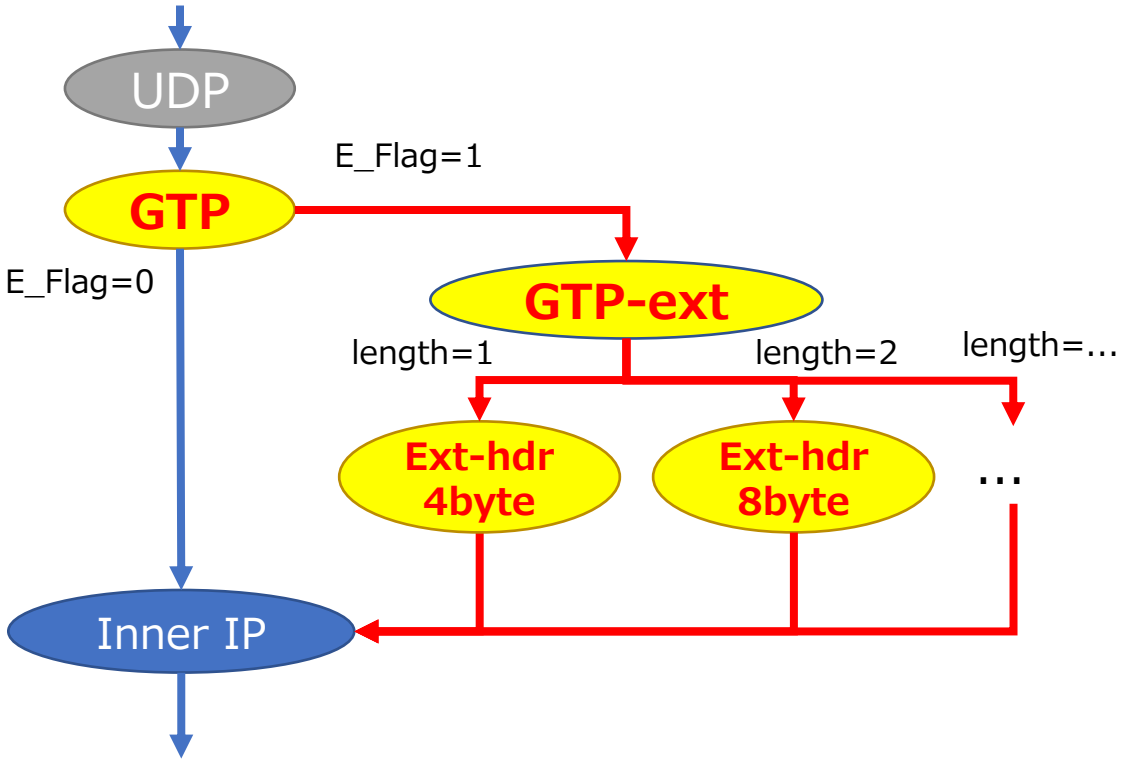
GTPヘッダーのParse処理(1)



Ext-HDRの有無によらず、
GTPヘッダーを正しく認識し、
Inner IP以下のオフセットを
取れるようにする
(Inner情報を後の処理で使
用するため)

GTPヘッダーのParse処理(2)

GTPのパーサー設計例



P4でのGTP ingress parser実装例 (P4_16)

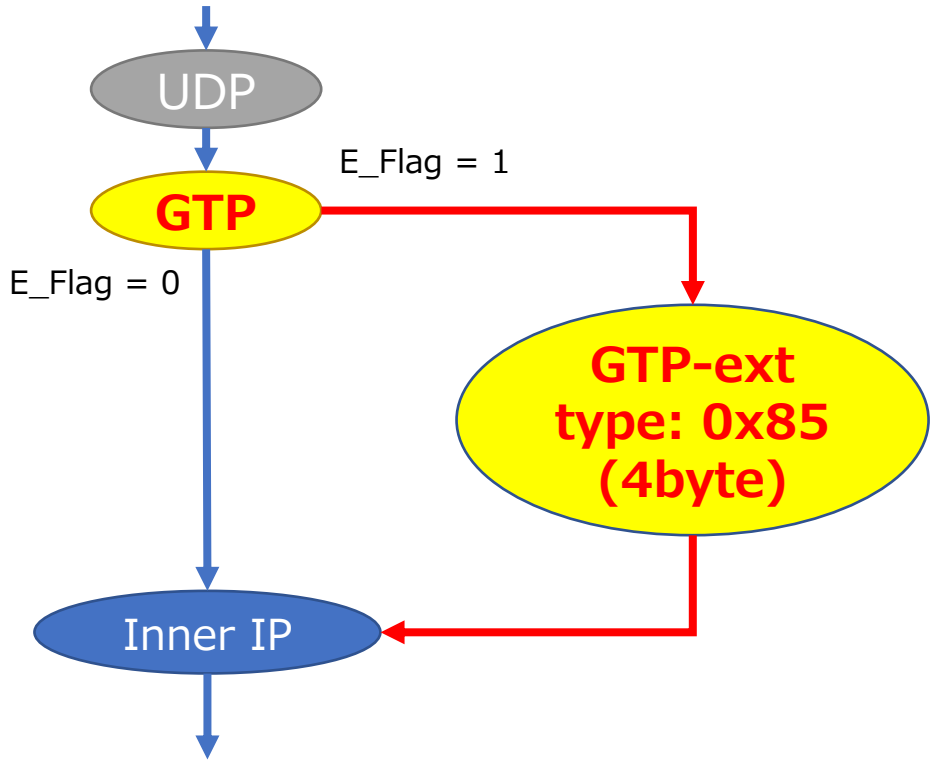
```
...
state parse_gtp {
    pkt.extract(hdr.gtp);
    transition select(hdr.gtp.e_flag) {
        0 : parse_inner_ip;
        1 : parse_gtp_ext_hdr;
    }
}

state parse_gtp_ext_hdr {
    bit<8> length = pkt.lookahead<bit<8>>();
    transition select(length) {
        1 : parse_gtpu_ext_header_4b;
        2 : parse_gtpu_ext_header_8b;
        ...
    }
}

state parse_gtpu_ext_header_4b {
    pkt.extract(hdr.gtpu_ext_4b);
...

state parse_gtpu_ext_header_8b {
    pkt.extract(hdr.gtpu_ext_8b);
...
```

E-flagのチェック

Ext-hdrのlength値に応じて、次のparseするheaderを選択

GTPヘッダーのParse処理(3)

Extension header typeが
決まっている場合の例 (0x85)



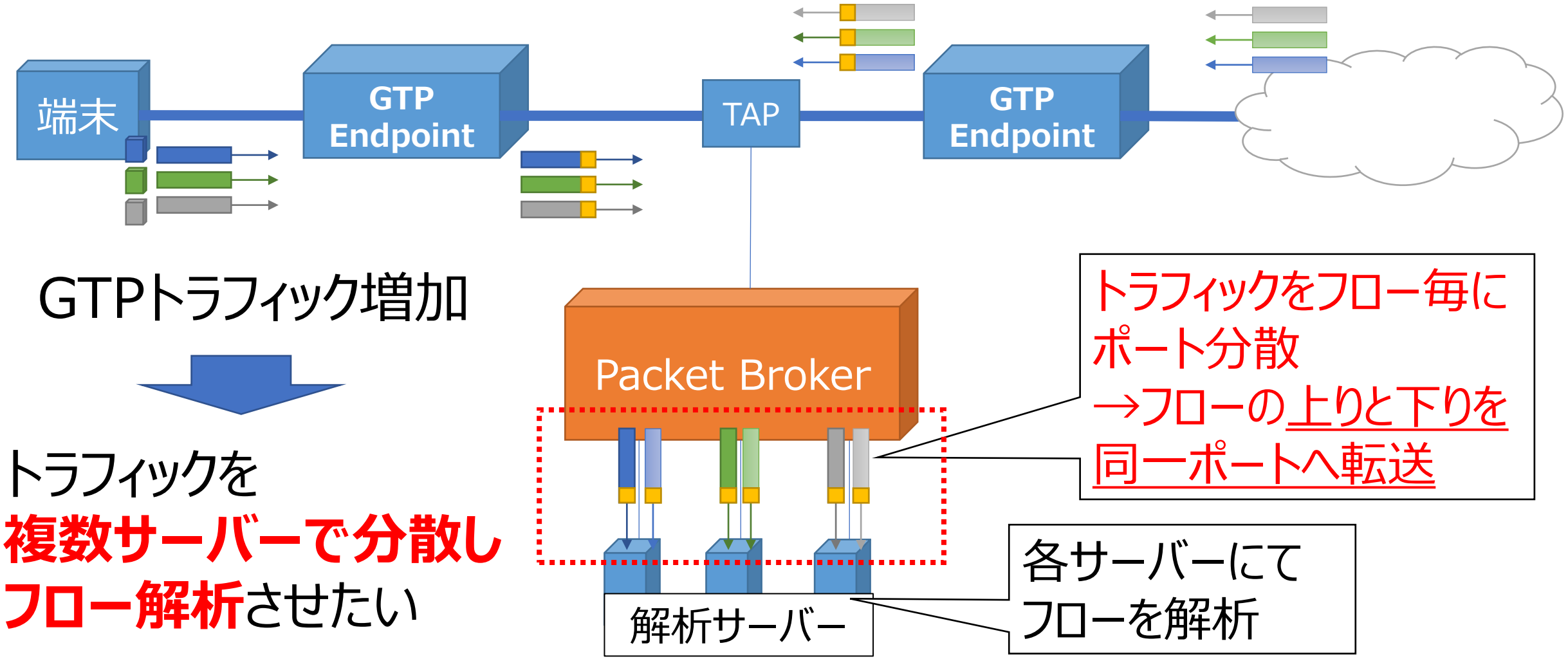
P4でのGTP ingress parser実装例 (P4_16)

```
...
state parse_gtp {
    pkt.extract(hdr.gtp);
    transition select(hdr.gtp.e_flag) {
        0 : parse_inner_ip;
        1 : parse_gtp_ext_hdr;
    }
}

state parse_gtp_ext_hdr {
    pkt.extract(hdr.gtp_ext);
    transition parse_inner_ip;
}

state parse_inner_ip {
    ...
```

E-flagのチェック

固定長ヘッダとみなし、extract

フロー単位トラフィック分散(1)



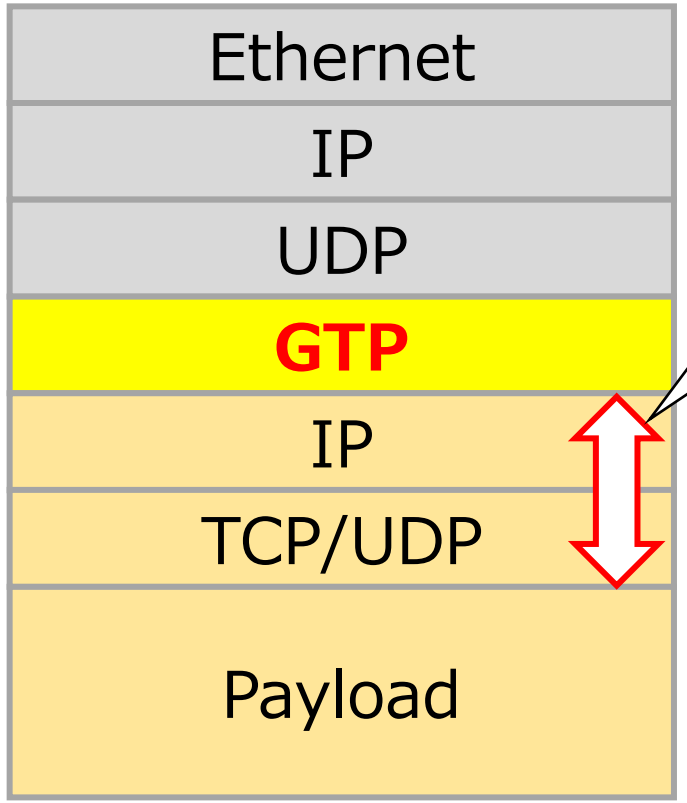
GTPトラフィック増加

トラフィックを
複数サーバーで分散し
フロー解析させたい

トラフィックをフロー毎に
ポート分散
→フローの上りと下りを
同一ポートへ転送

各サーバーにて
フローを解析

フロー単位トラフィック分散(2)



Flow単位で分散させるために、
Inner IPアドレス/L4ポート番号をキーにしてHash値を生成
(Symmetric Hash)

このHash値により、Egressポートを決定(Load-balance)

フロー単位トラフィック分散(3)

P4では、load-balanceの実装も可能

ポート分散で使用する要素を自由に選択可

P4での GTP LB 実装例 (P4_16)

```
@symmetric("hdr.inner_ipv4_src", "hdr.inner_ipv4_dst")
@symmetric("hdr.inner_l4_src", "hdr.inner_l4_dst")
Hash<bit<32>>(HashAlgorithm_t.CRC32) loadbalance_hash;
...
table load_balance {
        key = {
            ingress_port        : exact;
            loadbalance_hash    : selector;
        }
        actions = {
            set_egress_port;
        }

        implementation = loadbalance_selector;
}

apply {
...
        hash = loadbalance_hash.get({hdr.inner_ipv4_src,
hdr.inner_ipv4_dst, hdr.inner_l4_src, hdr.inner_l4_dst});
        load_balance.apply(hash);
...
```

symmetric hashの定義

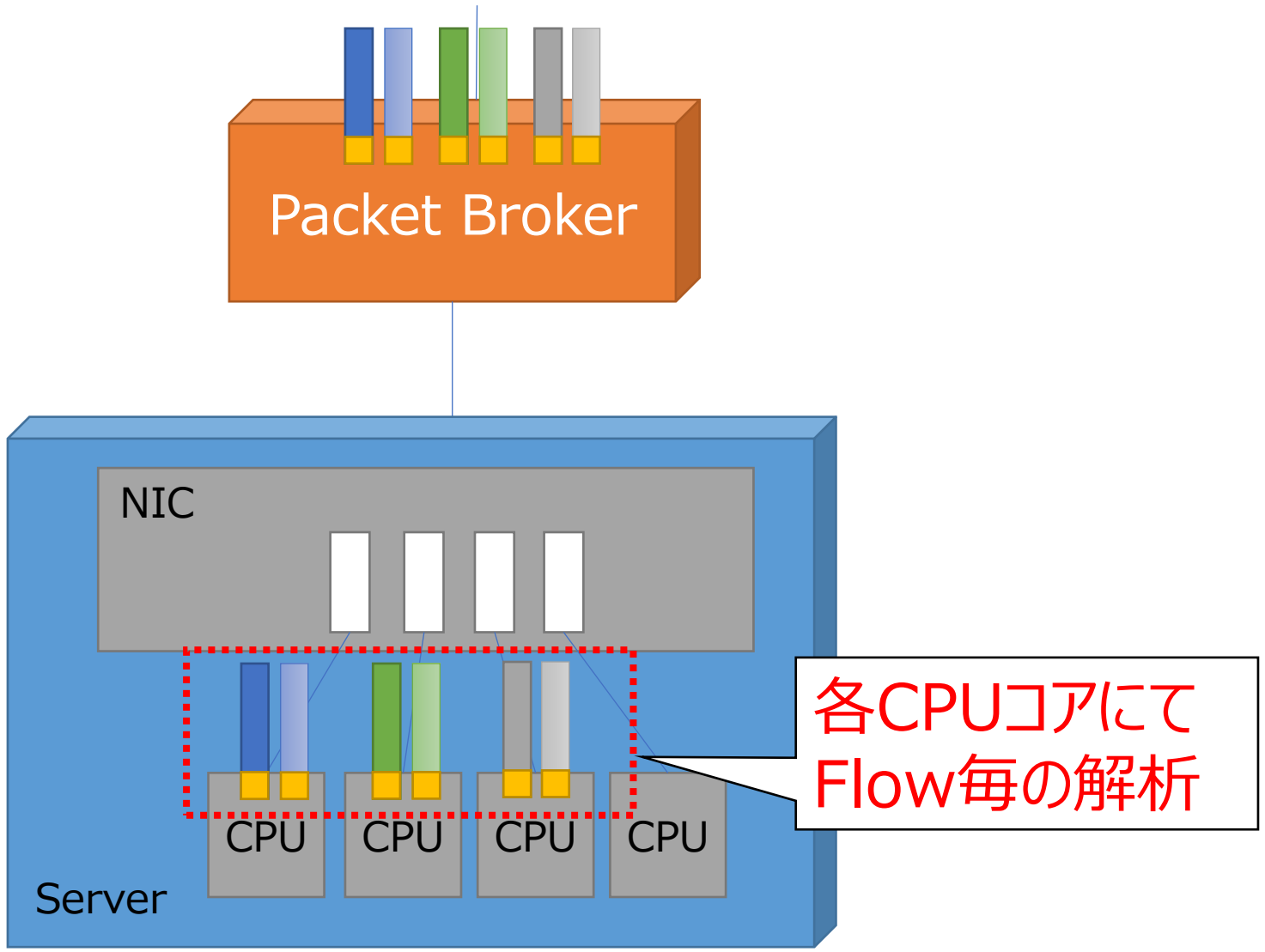
生成したHash値により、Load balanceの送信先ポートを選定

Inner情報よりHash値を生成

複数CPUコアへのトラフィック分散(1)

サーバーへ転送される
GTPトラフィックがさらに
増加

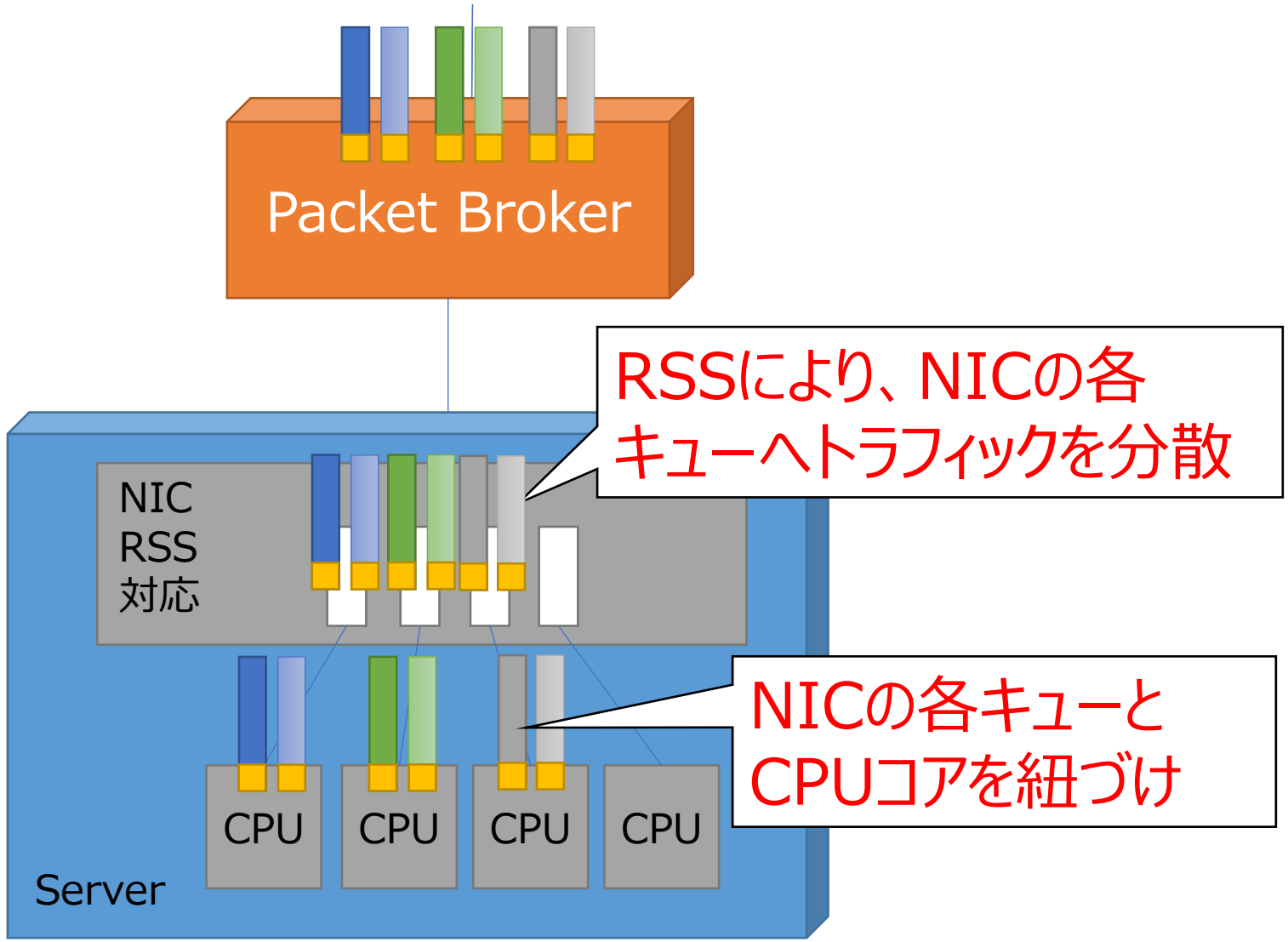
**トラフィックを
各サーバーの複数の
CPUコアに分散し
フロー解析したい**



複数CPUコアへのトラフィック分散(2)

NICの**RSS**機能を使用
(Receive Side Scaling)

トラフィックを複数の
CPUコアへ分散処理可能

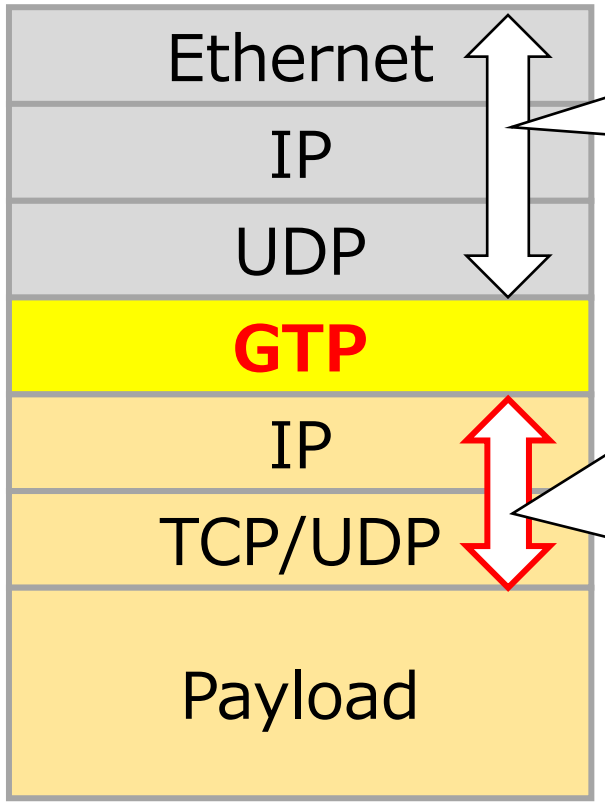


複数CPUコアへのトラフィック分散(3)

ロードバランスのとき同様
フローの上りと下りを同一の
CPUコアへ転送したい

しかし問題が・・・

RSSのHash値の計算には、
Innerヘッダの値は使えない



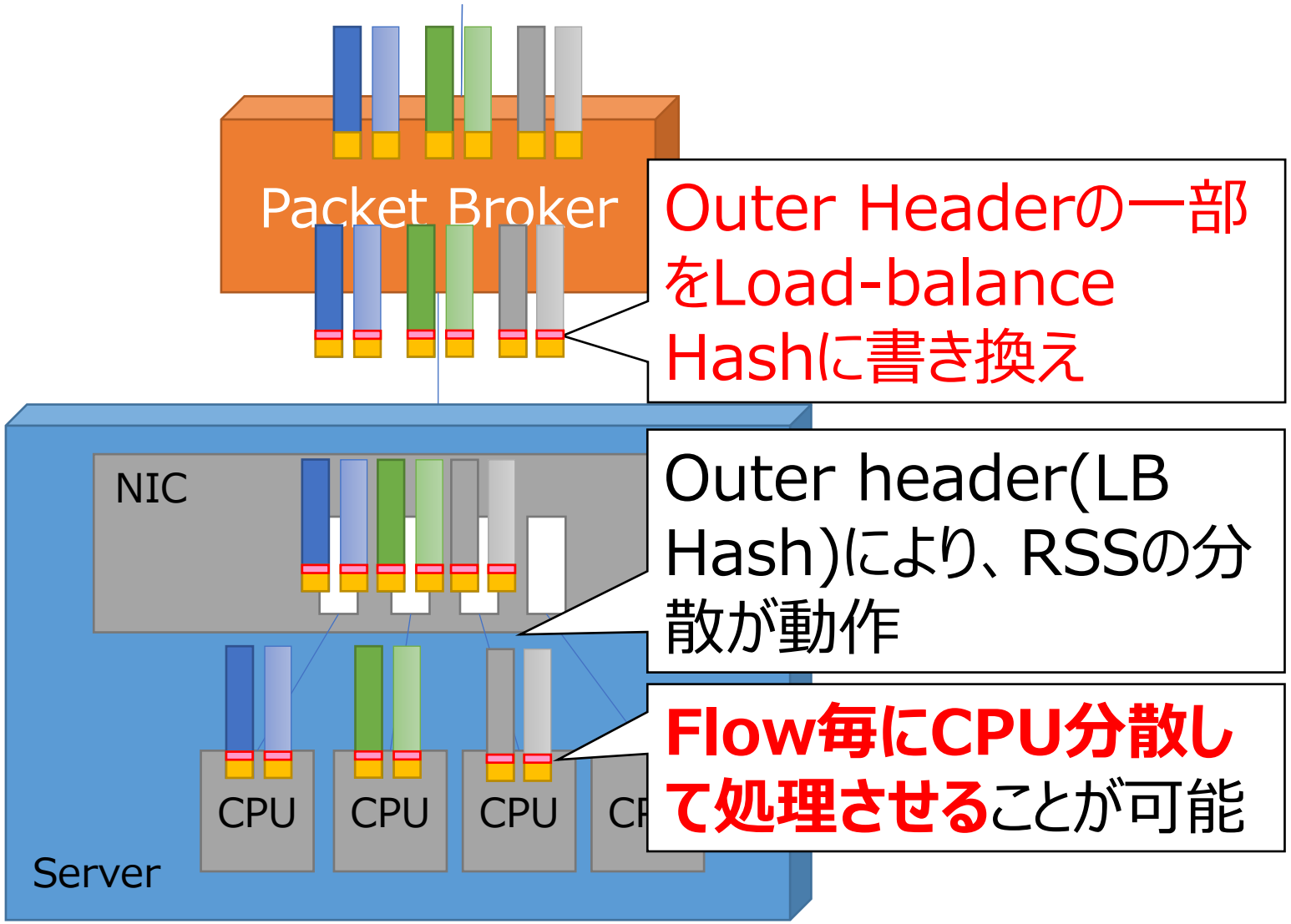
Outer headerのアドレス/ポート番号はRSSのHash計算で**使用可**

Inner headerのアドレス/ポート番号はHash計算**使用不可**
→Flow毎にCPU分散させるため、何とか**Innerの値で計算したHash値を使用したい**

複数CPUコアへのトラフィック分散(4)

P4では、任意のフィールド値を書き換えることが可能

→Packet Brokerにて**パケットのフィールド値をLoad-balanceで使用したHash値に書き換えることが可能**



複数CPUコアへのトラフィック分散(5)

P4での Hash stamp 実装例 (P4_16)

```
..
apply {
        hash = loadbalance_hash.get({hdr.inner_ipv4_src,
hdr.inner_ipv4_dst, hdr.inner_l4_src, hdr.inner_l4_dst});
        load_balance.apply(hash);
...
        hdr.ipv4.dst_addr = hash;
...
```

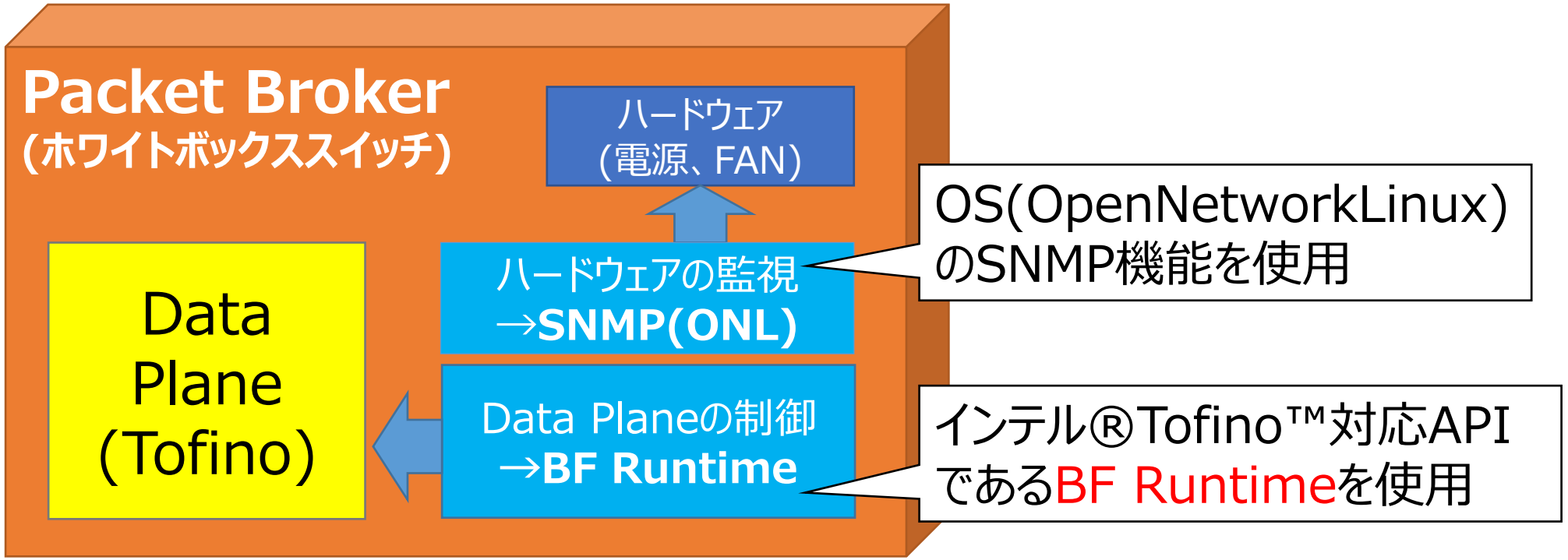
Dst IPフィールドを
Load-balance hash値
に書き換え

目次

- GTPとPacket Brokerの概要
- Data Plane実装
- **Control Plane実装**
- まとめ

Control Planeの実装

- 実運用、保守管理を考えると、Control Planeの実装も非常に重要



目次

- GTPとPacket Brokerの概要
- Data Plane実装
- Control Plane実装
- **まとめ**

発表のまとめ

- GTPトラフィック用のPacket Brokerを開発
- P4により、GTPトラフィックに最適化された仕様を実現
  - GTP extension headerに対応したParser
  - Flow毎のLoad-balance
  - 解析サーバーとの連携による負荷分散

Packet BrokerをP4で開発するメリット

Packet Brokerの特徴		P4で開発のメリット
シンプルな動作 L2/L3処理不要、ステートレス	→	**低コストで開発、運用可能**
一般的でない要件 特殊なプロトコル 解析サーバーとの連携	→	**P4による柔軟な実装により要件を実現**
大容量トラフィック	→	**負荷分散をP4で実装 ハードウェア処理対応**

EOF