



P4/Stratum活用に向けた プログラマブルスイッチの実装・検討

2018年10月19日
日本電信電話株式会社
NTTネットワークサービスシステム研究所
武井 勇樹

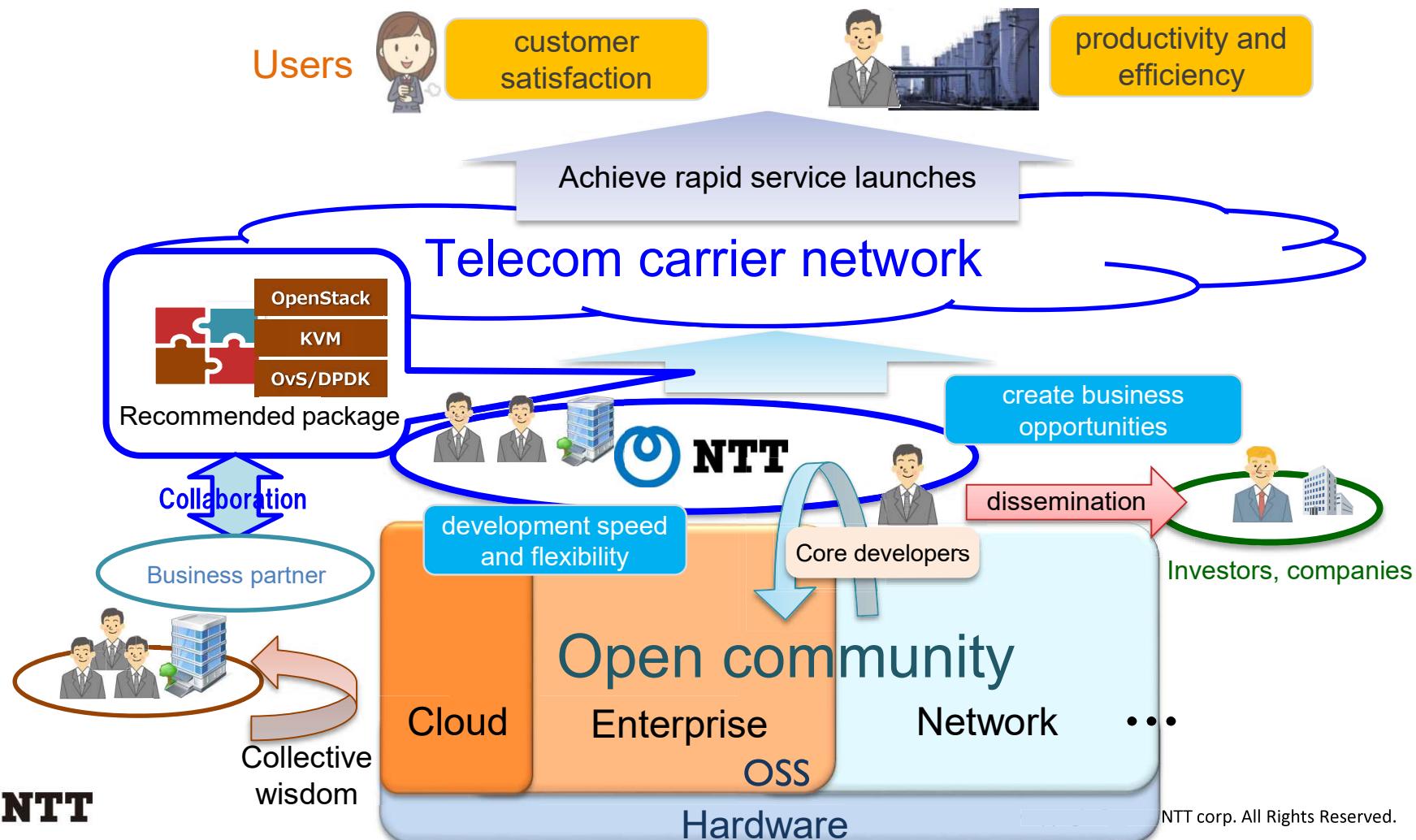
本日のアジェンダ

- 背景
- P4/Stratumの概要
- NTTのユースケース
- P4スイッチ/P4Runtimeの検討状況
- 今後の取り組み
- まとめ



NTTネットワークのR&Dビジョン

NTTは、「プロダクトのオープン化」と「OSSへの要件インプット」の両面からコミュニティ活動を実施し、通信キャリアネットワークへの適用を検討している。



ONF Reference Designs

- NTTではONF (Open Networking Foundation)に参加し、議論を実施している。
- ONFの新たな活動の方向性として、以下の技術分野を中心に取り組む方針を発表(2018.6)。我々はRD-UPANにおいて、P4/Stratumの通信キャリアにおけるユースケースを議論している。

	概要	バースとなるONFの取り組み	オペレータ
RD-SEBA (SDN Enabled Broadband Assess)	Lightweight reference design supporting a multitude of virtualized access technologies at the edge of the carrier network, including PON, G.Fast, DOCSYS and more.	R-CORD	AT&T, Deutsche Telekom, NTT , Turk Telekom
RD-Trellis	SDN-native spine-leaf data center fabric optimized for edge applications.	Trellis	Comcast
RD-UPAN (Unified, Programmable & Automated Network)	Next generation SDN reference design, leveraging P4 to enable flexible data plane programmability and network embedded VNF acceleration	Stratum	China Unicom, Deutsche Telekom, Google, NTT , Turk Telekom
RD-ODTN (Open Disaggregated Transport Network)	Open multi-vendor optical networks	ODTN	China Unicom, Comcast, NTT

我々の取り組み領域

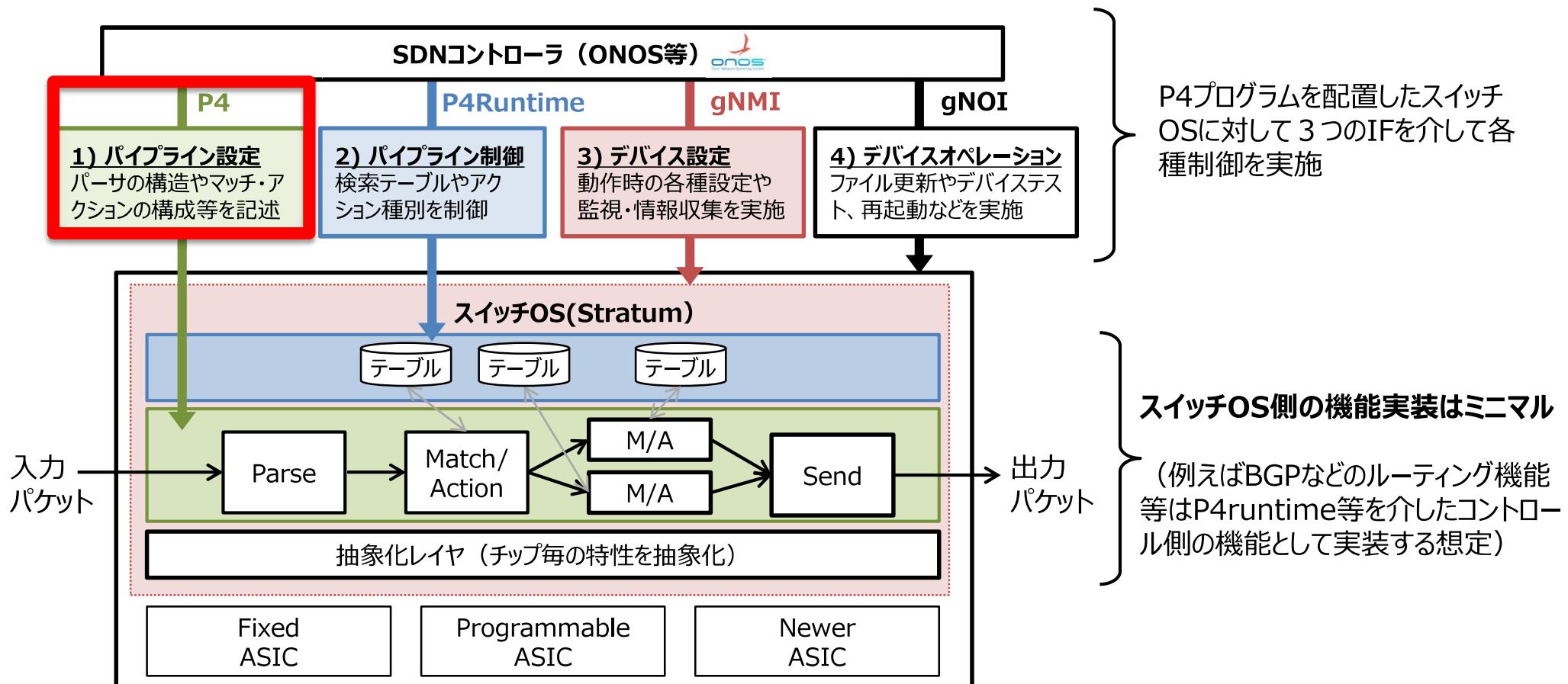
NTTグループからはRD-SEBA(vOLT)、RD-UPAN(エッジ/P4/Stratum)、RD-ODTN(ODTN)に参加



P4/Stratumの概要

Stratumとは

- Stratumは、SDN集中制御を前提としたミニマルな機能実装を指向したスイッチOSである。
- 異なるベンダチップを搭載したスイッチであっても統一的な手段で制御するためにP4Runtime・gNMI・gNOIと呼ぶインタフェースを規定。



P4とは

- P4 :
High-level Language for
Programming Protocol-independent Packet Processors の略
- パケットを受け取ってから、その中身に応じて処理を行い送り出すまで(データプレーン)
の動作を記述するための**プログラミング言語**
- 『**特定のプロトコルに依存せず、
パケット処理の独自仕様を実現できる**』
(Protocol-independent)
- 『**多様なハードウェアの仕様を意識せず、
パケットの処理の内容を実装できる**』
(Target Independent)



<https://p4.org/>より引用



P4対応デバイス

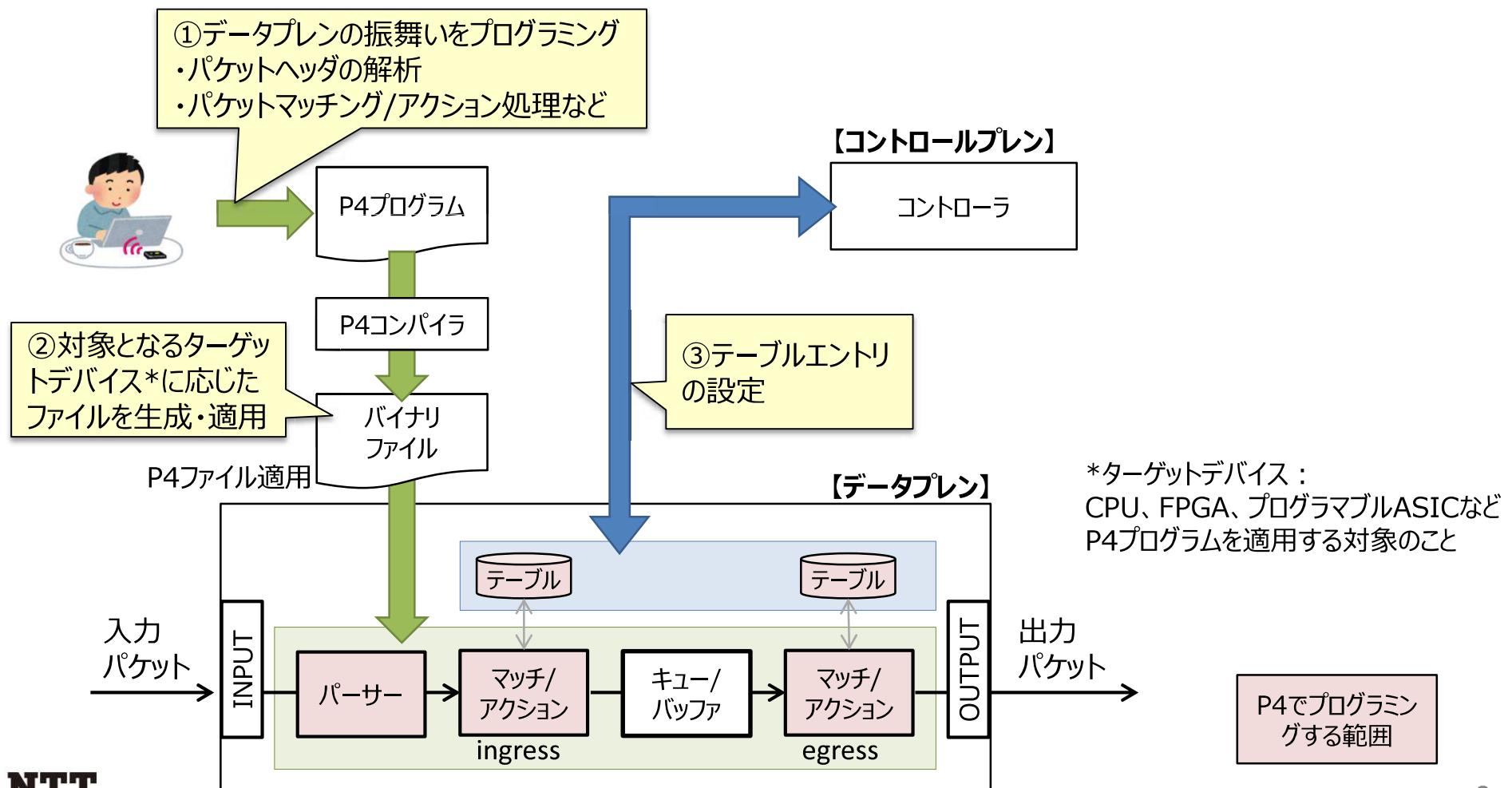
- CPU、FPGA、NPU、スイッチASICといった多様なデバイス種別に対応している。
- P4には現在 2つの仕様バージョン(P4₁₄ / P4₁₆)が存在し、どちらもメンテナンスが継続。

デバイス種別	フレームワーク例	コミュニティ/ベンダ例	サポートP4バージョン
CPU	BMv2	P4.org	P4_14/16
Smart NIC	FPGA	SDNet	Xilinx
		NP4	Netcope
	NPU	Agilio	Netronome
Switch-ASIC	Tofino	Barefoot	P4_14/16

※2018.10時点

P4でパケット転送するまでの基本的な流れ

- データプレンの振舞いを定義するプログラム（P4プログラム）を準備し、コンパイル（①②）
- パケット転送に必要なテーブルエントリを設定し、パケットを転送（③）

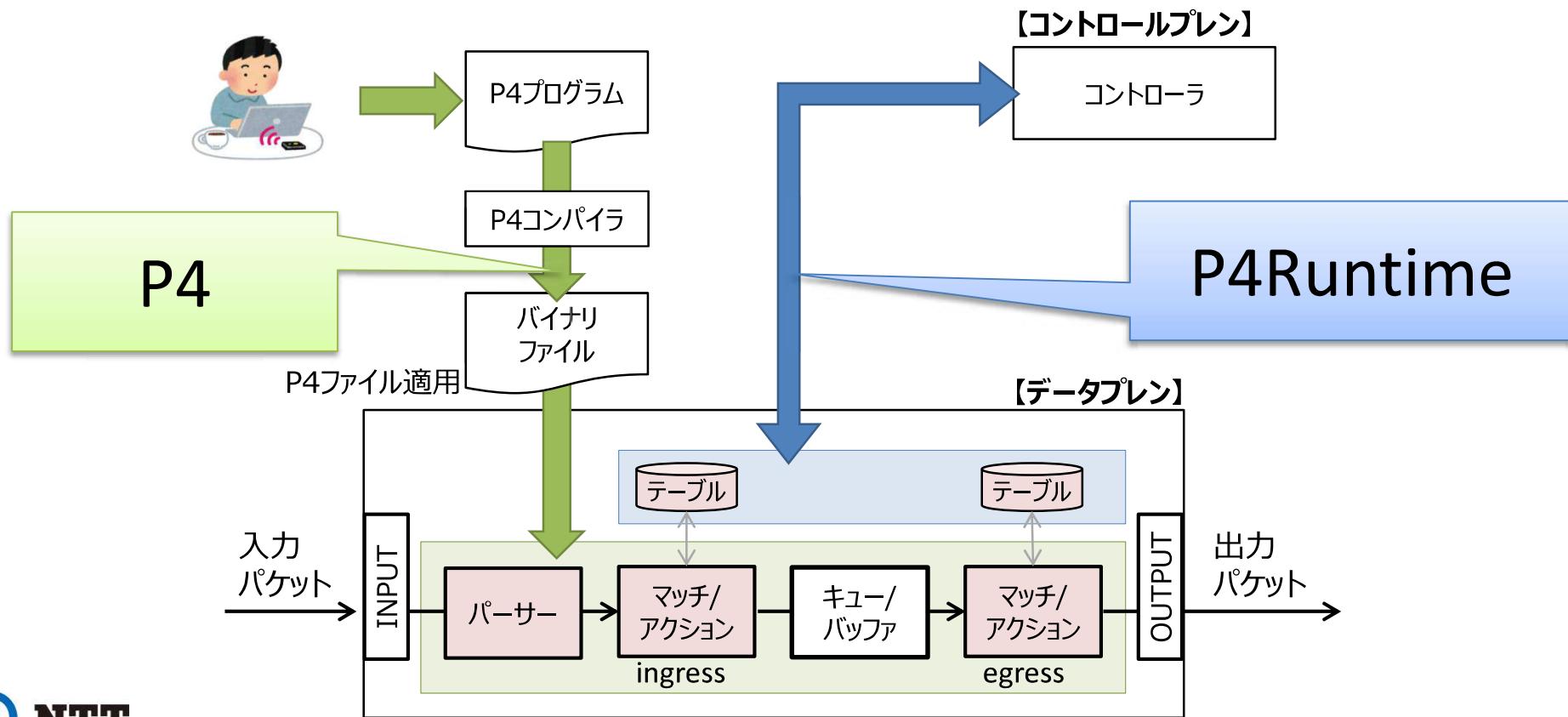


P4とP4 runtime

P4 : デバイスを抽象化し、スイッチがどのようにパケットを処理するか定義。

パイプラインの指定(どのフィールドが一致するか、どんなアクションを実行するか等)。

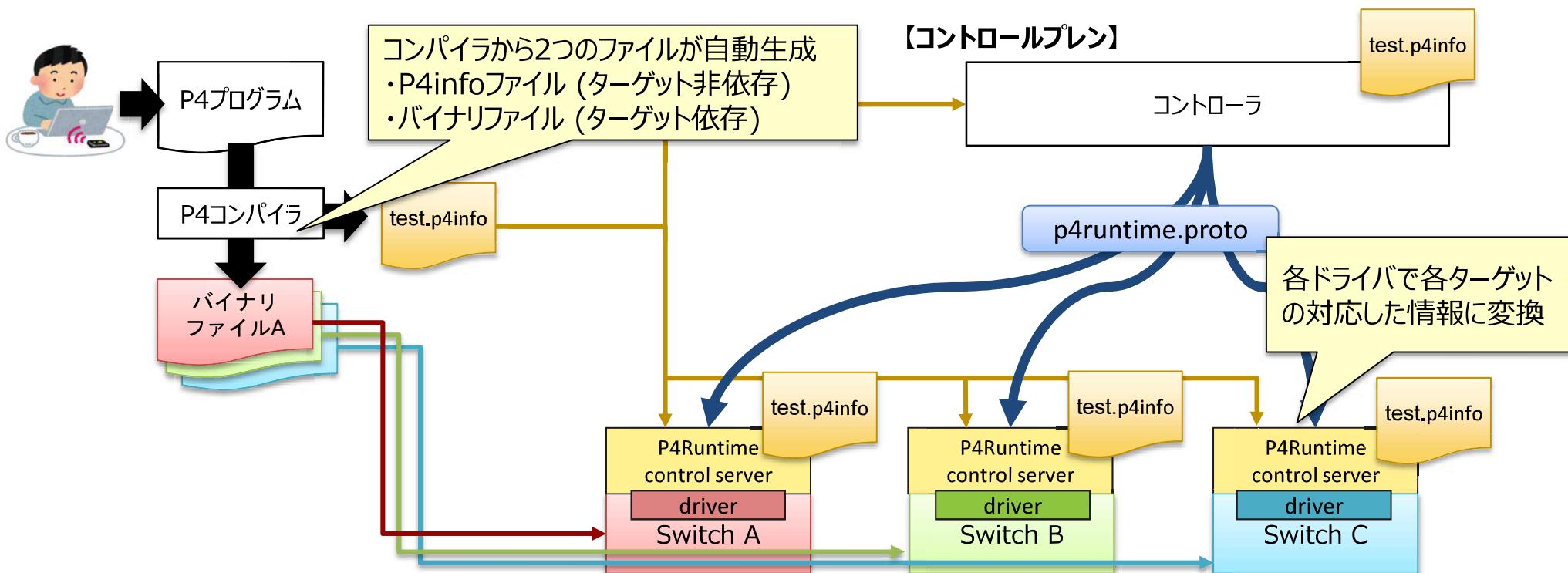
P4Runtime : 固定的な機能のスイッチであるか、プログラマブルであるかに関係なく、P4言語で動作が指定されている機能を制御するために使用するAPI。



P4処理の流れについて

コンパイラから2つのファイルが自動生成。

- ①P4infoファイル (ターゲット非依存) : コントロールプレーンから制御するために必要なテーブルIDなどの対応関係が格納。外部コントローラはこのファイルを参照しながらターゲットデバイスを制御。
- ②バイナリファイル (ターゲット依存) : 対象となるターゲットデバイスに応じたデータプレーンファイル。





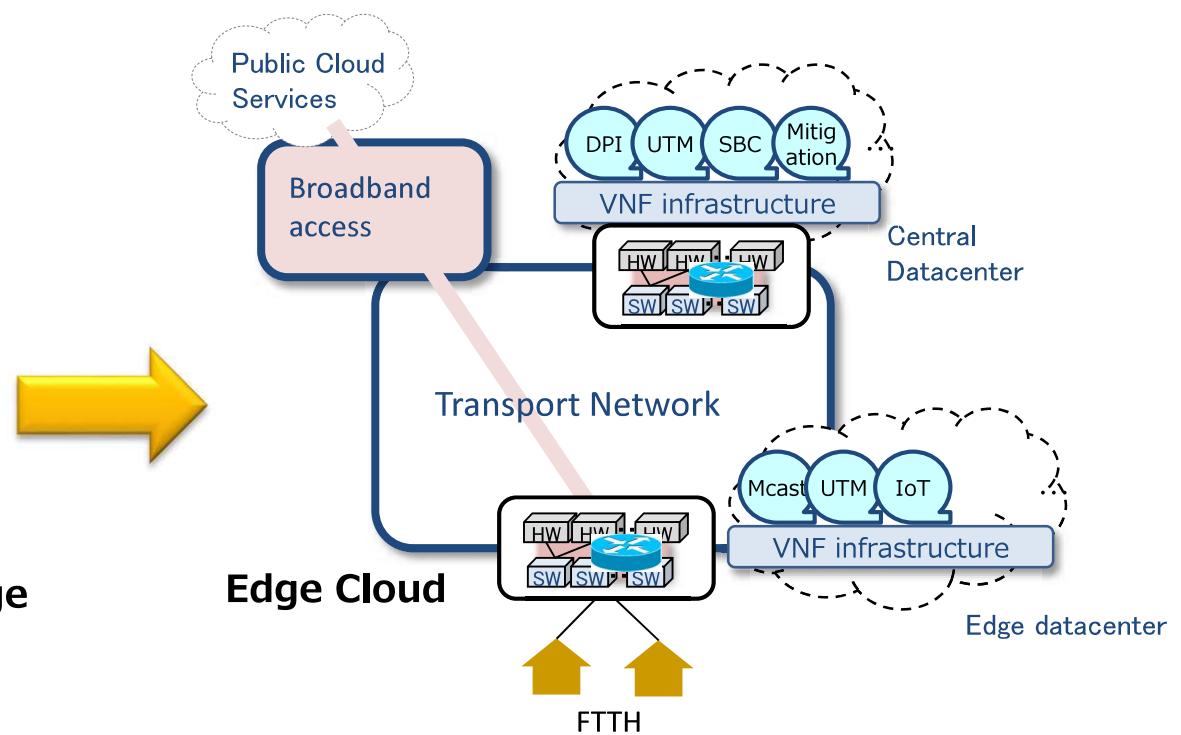
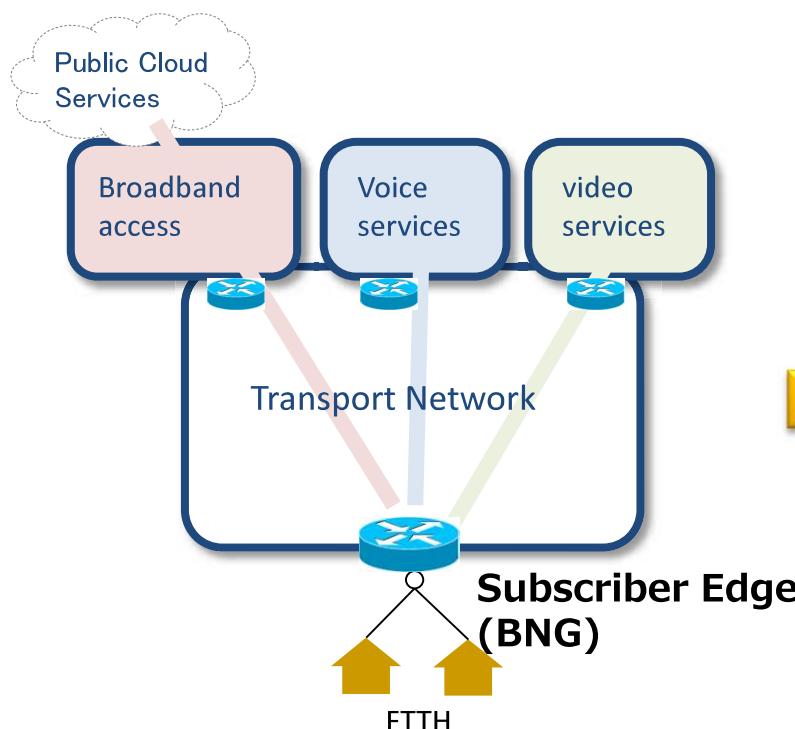
Innovative R&D by NTT

NTT Confidential

NTTのユースケース

将来ネットワークのイメージ

- 将来のネットワークにおいては、サービスの変更・拡張に対して、経済的かつ柔軟に対応するため、プログラマビリティが要求されると考えられる。
- この実現に向けて、P4/Stratumを活用できなか検討している。



- トリプルプレイヤーサービス
- 加入者向けに最適化

- サービスの変更・拡張にあわせ、ネットワーク機能を迅速に提供
- 新しい事業を始めたい事業者や企業に応じたカスタマイズが可能



P4/Stratumのユースケース

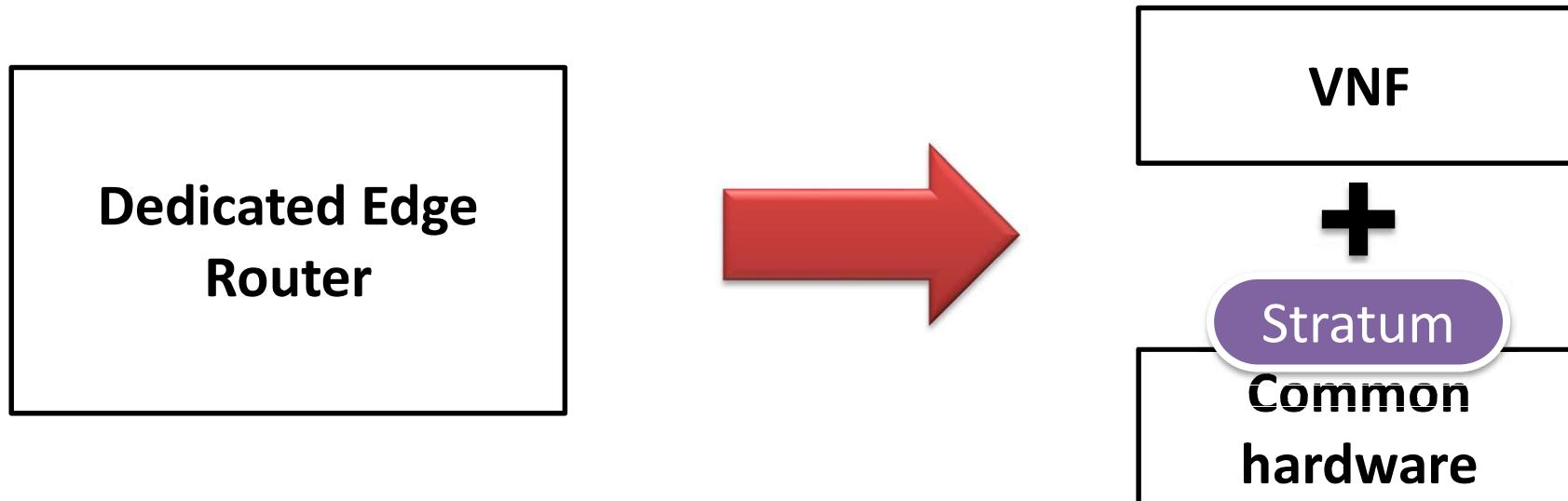
- 我々はP4/Stratumを活用のユースケースとして、2つのアプローチを検討。
 - BNGのディスアグリゲーション (VNFオフロード)
 - 通信品質の可視化 (In-band Network Telemetry)

効用	BNGのディスアグリゲーション (VNFのオフロード)	通信品質の可視化 (In-band Network Telemetry)
効用	NWの経済化	新サービス提供
解決する課題	現状の仮想化NWでは、高度な性能を出すためにはサーバ増設が必要あり、CAPEXが増加してしまう	現状のNW基盤では、低遅延等の新サービスに向けた、厳密な通信品質の管理ができない
実現できること	プログラマブル技術により、サーバ上の機能の一部をスイッチへオフロード可能となり、サーバの増設台数を抑えることができる	プログラマブル技術により、細かい単位で情報を取得可能



ユースケース 1 BNGのディスアグリゲーション(1/2)

- ネットワークの柔軟性・スケール性の向上のため、既存の大型ルータを汎用装置とVNF技術による実現を目指している。

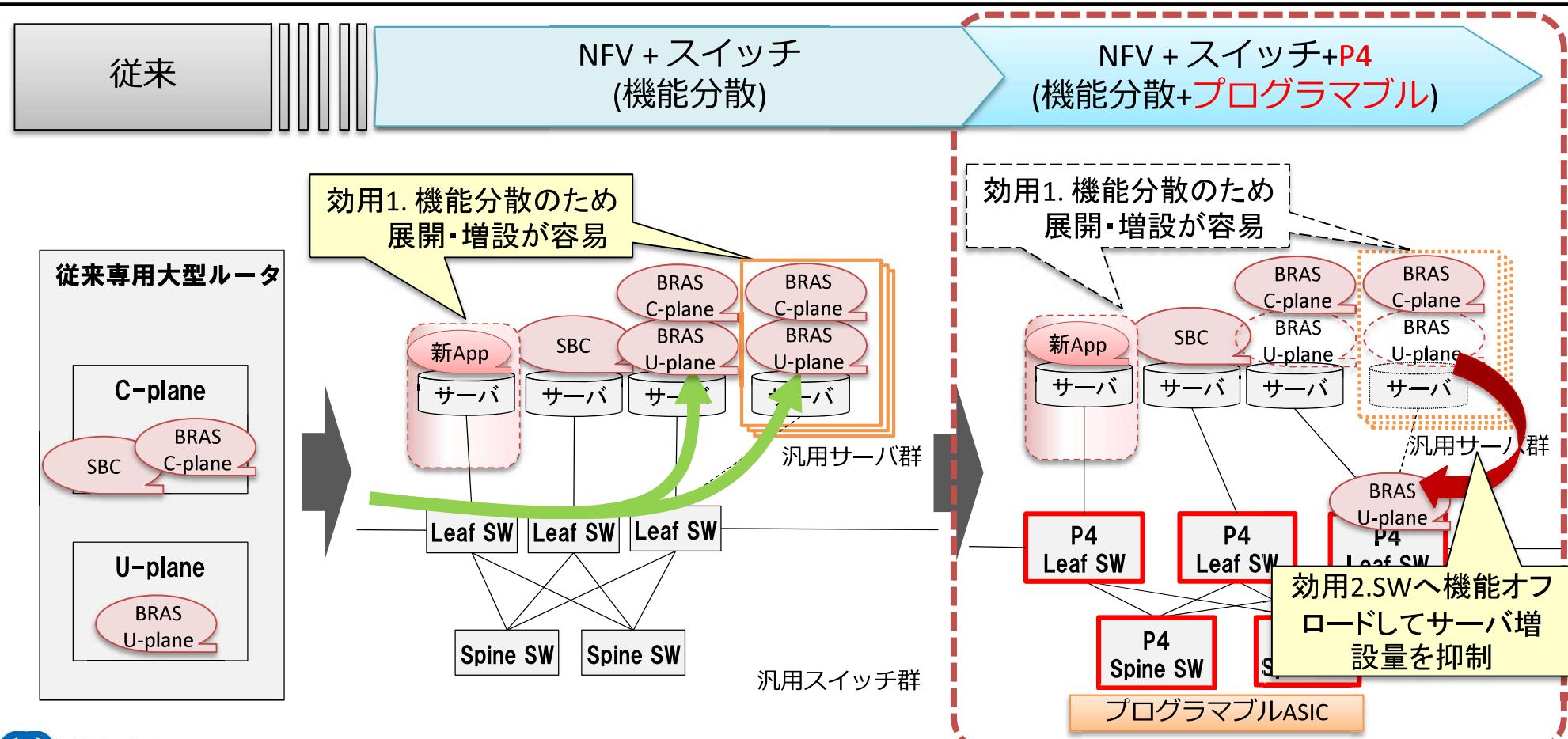


ユースケース1 BNGのディスアグリゲーション(2/2)

汎用サーバだけではなくホワイトボックス側にも自らが定義した機能を自在に分散配備できる。

効用1. 必要な設備を必要なだけ増設すれば良いので、新たなサービスを迅速かつ安価に展開可能。

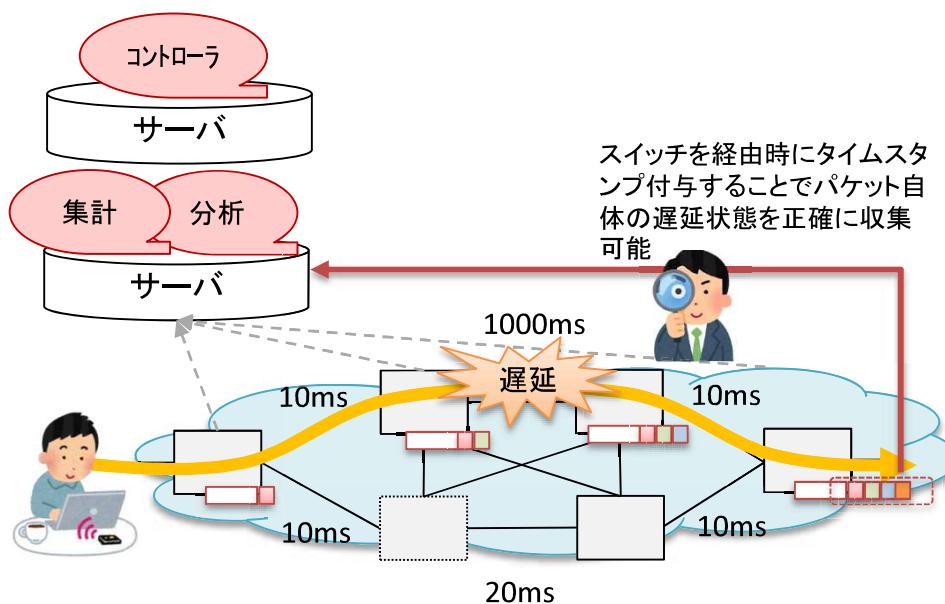
効用2. 汎用サーバ上で仮想化している機能の内、ボトルネックだけをスイッチ側へ移行可能。



ユースケース2 通信品質の可視化

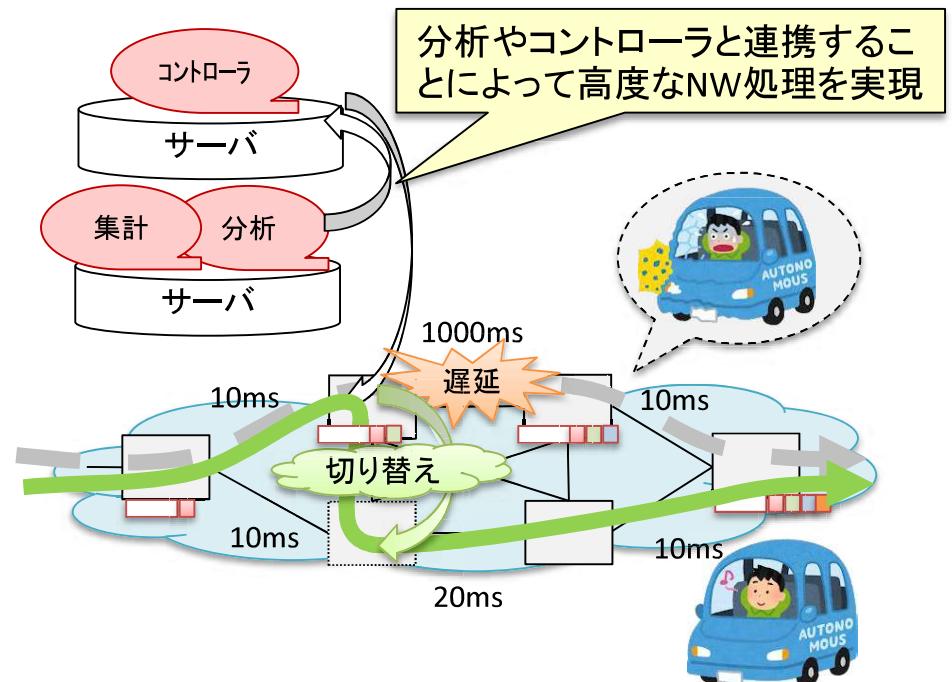
- 情報取得を装置単位からパケット単位にすることで、通信品質を満たしているかを明確に可視化。
- 細かい単位で情報を取得できるので、精緻な制御を実施可能なインフラを用意できる。

INTの適応イメージ



遅延情報をパケット単位で可視化
(低遅延アプリで遅延が発生していないか? 等)

INTの応用例



情報を元にパケット単位の緻密な制御を実現
(遅延発生を予測して経路を迂回する等)

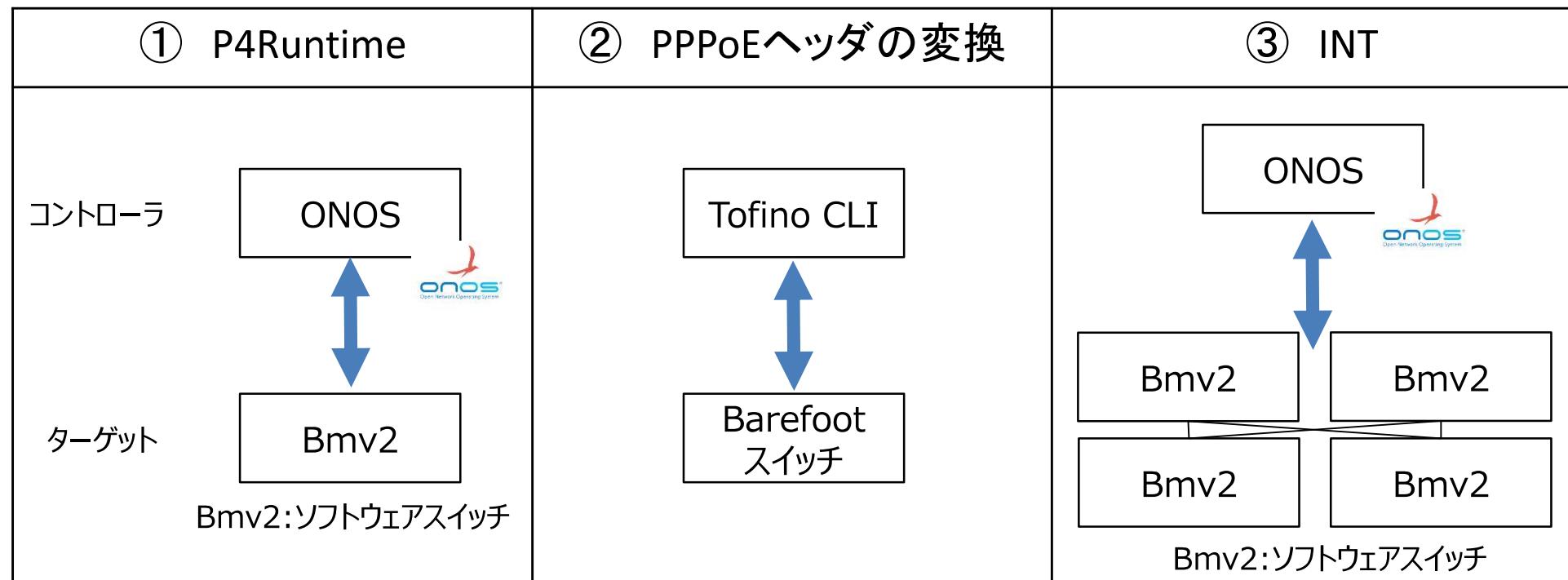


P4スイッチ/P4Runtimeの検討状況

取り組みの全体像

P4/Stratumを用いたユースケース実現に向けて、現在下記の評価・検討を実施。

- ① P4Runtimeの実装状況の確認
- ② P4によるハードウェアオフロードの実現性の確認 (PPPoEヘッダの変換)
- ③ P4によるINTの実現性の確認



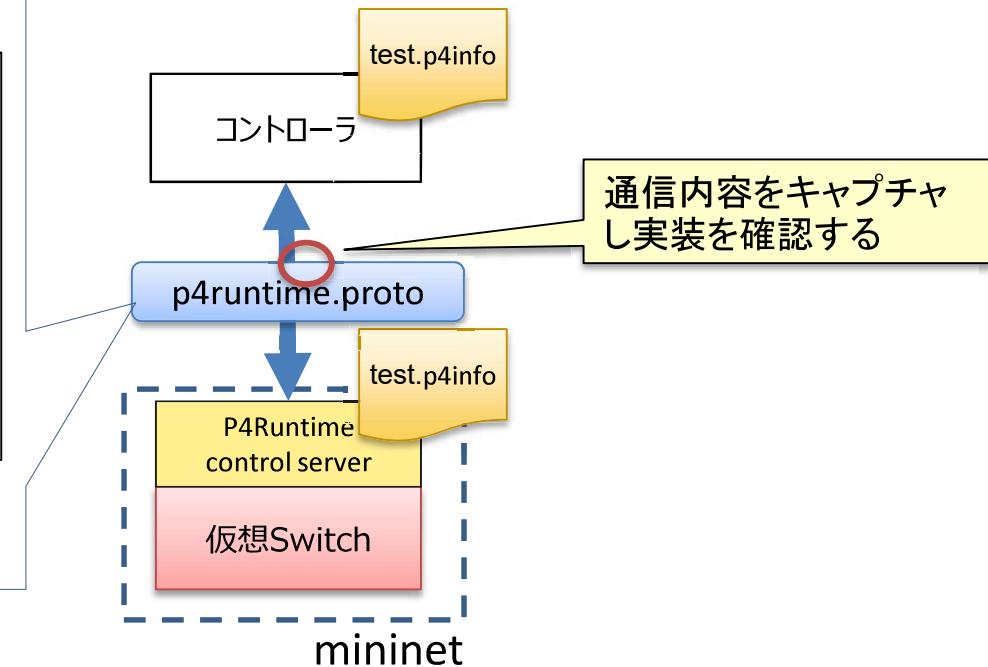
① P4Runtimeの実装状況 (1/2)

- P4Runtimeは、2018.6に仕様ドラフトがFIXされた。
- ONOSコントローラとデバイス(ソフトウェアスイッチ)間の通信をキャプチャ・Wiresharkで解析することで、P4Runtimeの実装状況を調査する。
- ONOS wikiのチュートリアル(※)を実施。

p4runtime.protoによる通信内容例

```
service P4Runtime {
    rpc Write(WriteRequest) returns (WriteResponse) {}
    rpc Read(ReadRequest) returns (stream ReadResponse) {}
    rpc SetForwardingPipelineConfig(SetForwardingPipelineConfigRequest)
        returns (SetForwardingPipelineConfigResponse) {}
    rpc GetForwardingPipelineConfig(GetForwardingPipelineConfigRequest)
        returns (GetForwardingPipelineConfigResponse) {}
    rpc StreamChannel(stream StreamMessageRequest)
        returns (stream StreamMessageResponse) {}
}
```

<https://github.com/p4lang/p4runtime/blob/master/proto/p4/v1/p4runtime.proto>より



※ONOS wikiのチュートリアルより

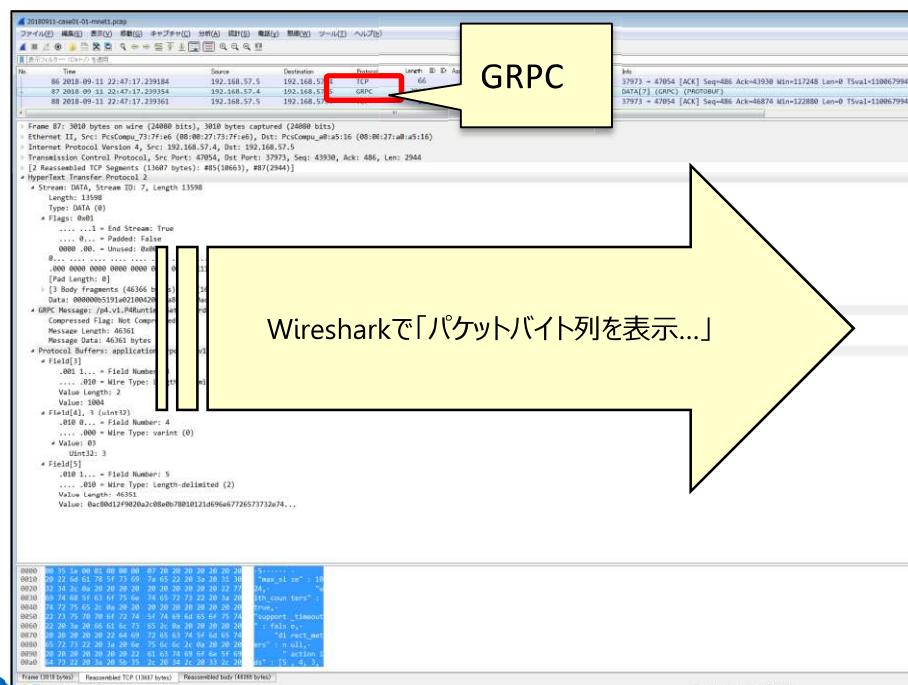
<https://wiki.onosproject.org/display/ONOS/P4Runtime+support+in+ONOS>

① P4Runtimeの実装状況 (2/2)



- Bmv2環境におけるP4Runtimeの実装状況。
 - ONOS \leftrightarrow bmv2間でのgRPCベースの通信は実装済みであることを確認。
 - SetForwardingPipelineConfig RequestではBmv2のjsonファイルを送信など。
 - ONOSのGUIからネットワークの構成およびデバイスの情報が表示できる。
 - 現在、ONOSとbarefootのスイッチを接続し、同様に通信を解析中。

【SetForwardingPipelineConfig Requestの例】

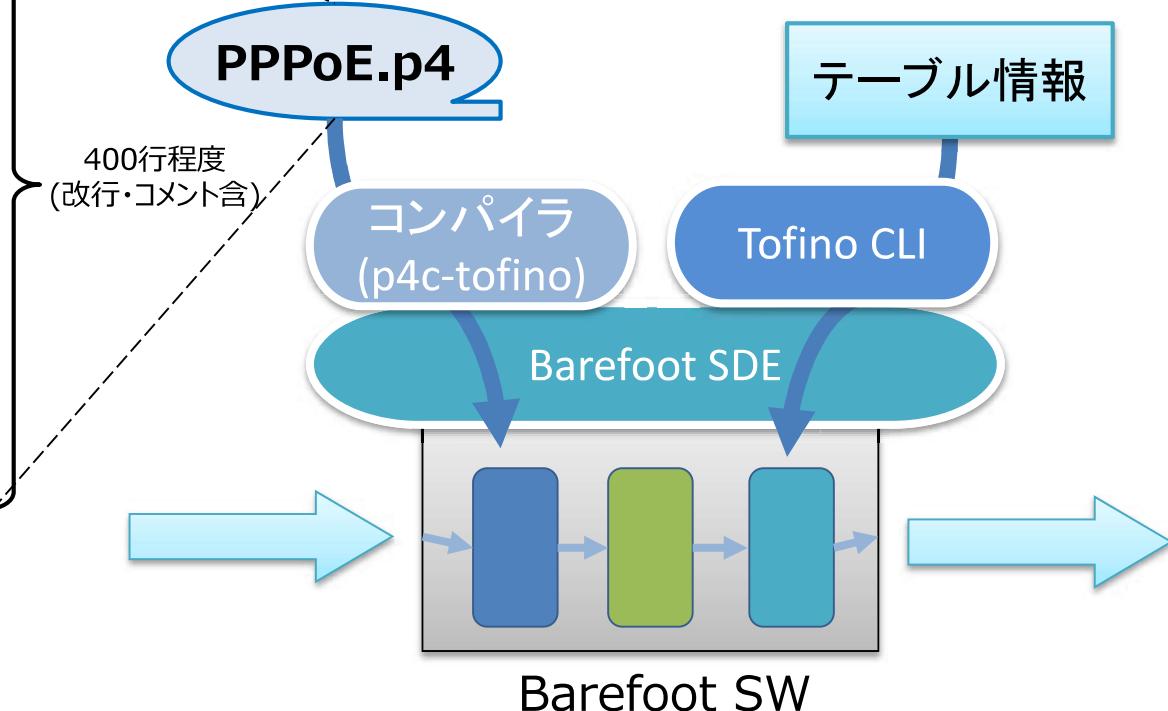


```
1     "max_size" : 1024,
2     "with_counters" : true,
3     "support_timeout" : false,
4     "direct_meters" : null,
5     "action_ids" : [5, 4, 3, 0],
6     "actions" : ["ingress.table0_control.set_egress_port",
7 "ingress.table0_control.send_to_cpu", "ingress.table0_control.set_next_hop_id",
8 "_drop"],
9         "base_default_next" : "tbl_act_2",
10        "next_tables" : [
11            "ingress.table0_control.set_egress_port" :
"tbl_act_2",
12            "ingress.table0_control.send_to_cpu" : "tbl_act_2",
13            "ingress.table0_control.set_next_hop_id" :
"tbl_act_2"
14        ],
15    },
16    "intrinsic_metadata.recirculate_flag",
17    ["standard_metadata", "recirculate_flag"]
18],
19],
20],
21],
22],
23],
24],
25],
26],
27],
28],
29],
30],
31],
32],
33],
34],
35],
36],
37],
38],
39],
40],
41],
42],
43],
44],
45],
46],
47],
48],
49],
50],
51],
52],
53],
54],
55],
56],
57],
58],
59],
60],
61],
62],
63],
64],
65],
66],
67],
68],
69],
70],
71],
72],
73],
74],
75],
76],
77],
78],
79],
80],
81],
82],
83],
84],
85],
86],
87],
88],
89],
90],
91],
92],
93],
94],
95],
96],
97],
98],
99]
```

② P4によるPPPoEヘッダの変換の実装（1/2）

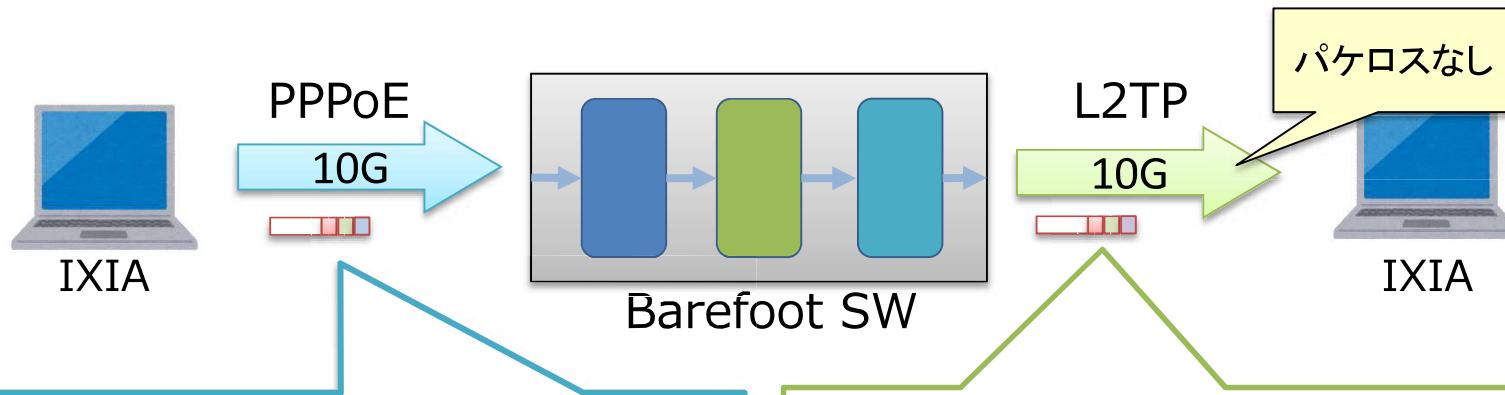
- PPPoEパケットをL2TPパケットに変換するオリジナルコードをP4_14で作成。
→ 非常に簡単に実装が可能（400行程度）
- Barefoot SDEを用いてスイッチ上に実装。

```
PPPoE.p4 (P4_14)
/***** headers *****/
...
header_type pppoe_t {
    fields {
        ver      : 4;
        pktType : 4;
        code    : 8;
        sessionId : 16;
        payloadLength : 16;
    }
}
...
/***** parser *****/
...
/***** action *****/
...
/***** table *****/
...
/***** control */
...
```



② P4によるPPPoEヘッダの変換の実装 (2/2)

- 試験機より疑似パケットを印加し、実装した機能が動作するか確認。
- 下記のように、ハードウェアスイッチ上でパケットの変換動作が可能。



```

> Frame 1: 974 bytes on wire (7792 bits), 974 bytes captured (7792 bits) on interface 0
> Ethernet II, Src: MS-NLB-PhysServer-06_0a:0e:ff:f3 (02:06:0a:0e:ff:f3), Dst: Rivertre_0
> 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 1001
* PPP-over-Ethernet Session
  0001 .... = Version: 1
  .... 0001 = Type: 1
  Code: Session Data (0x00)
  Session ID: 0x0001
  Payload Length: 946
> Point-to-Point Protocol
> Internet Protocol Version 4, Src: 2.2.2.2, Dst: 1.1.1.1
> User Datagram Protocol, Src Port: 60002, Dst Port: 60001
> Data (916 bytes)

```

PPPoE

```

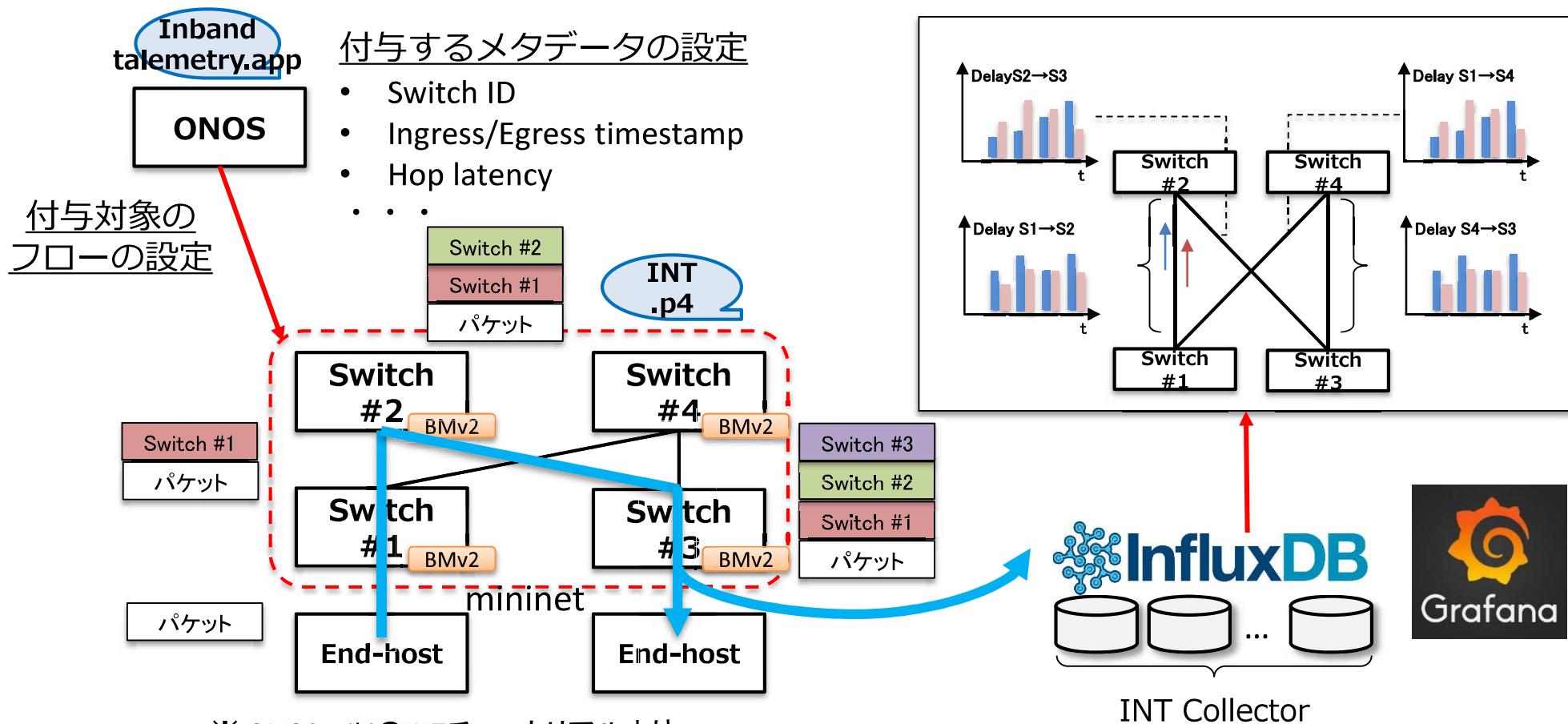
> Frame 1: 1000 bytes on wire (8000 bits), 1000 bytes captured (8000 bits) on interface 0
> Ethernet II, Src: Cisco_00:00:01 (00:12:01:00:00:01), Dst: AlaxalaN_e3:0b:bb (00:12:e2:e)
> Internet Protocol Version 4, Src: 10.3.0.2, Dst: 100.0.0.1
> User Datagram Protocol, Src Port: 1701, Dst Port: 1701
* Layer 2 Tunneling Protocol
  > Packet Type: Data Message Tunnel Id=61847 Session Id=64798
    Tunnel ID: 61847
    Session ID: 64798
  > Point-to-Point Protocol
  > Internet Protocol Version 4, Src: 2.2.2.2, Dst: 1.1.1.1
  > User Datagram Protocol, Src Port: 60002, Dst Port: 60001
  > Data (916 bytes)

```

L2TP

③ P4によるINTの実現性の確認 (1/3)

- INT(In-band Network Telemetry) とONOSを用いたINT実装を確認。
- ONOS wikiのINTチュートリアルを使用。

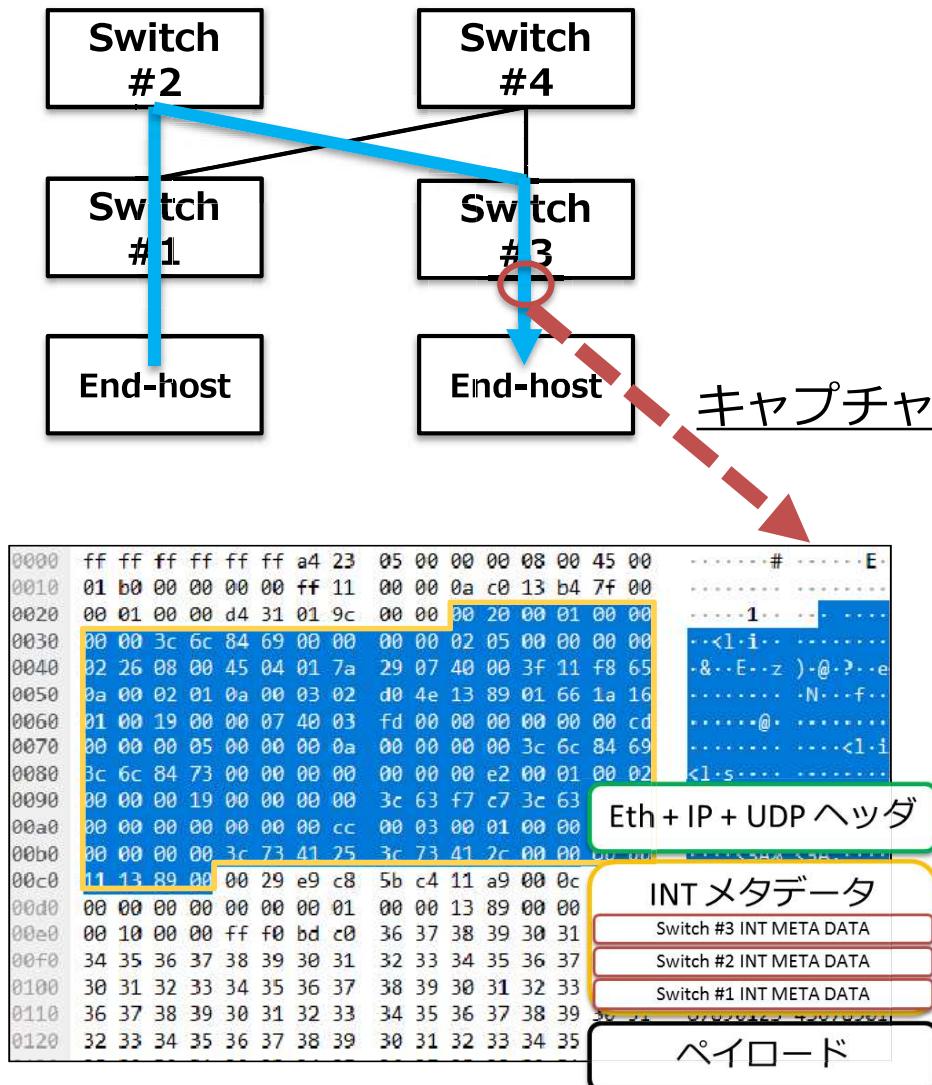


※ ONOS wikiのINTチュートリアルより

<https://wiki.onosproject.org/display/ONOS/%28INT%29+In+Band+Network+Telemetry+with+ONOS+and+P4>

Copyright©2018 NTT corp. All Rights Reserved.

③ P4によるINTの実現性の確認 (2/3)



```

Frame 1: 488 bytes on wire (3904 bits), 488 bytes captured (3904 bits)
> Ethernet II, Src: OpenNetw_00:00:00 (a4:23:05:00:00:00), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Internet Protocol Version 4, Src: 10.192.19.180, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 0, Dst Port: 54321
  INT-Report Protocol Data
    > Telemetry Report Fixed Header
    Encapsulated Header
    > INT Shim Header for TCP/UDP
    > INT metadata Header
    > INT metadata
      > Switch metadata #3
        > Switch metadata #2
          Switch ID: 226
          Ingress port ID: 1
          Egress port ID: 2
          Hop Latency: 25
          0000 0000 ..... .... .... = Queue ID: 0
          .... .... 0000 0000 0000 0000 0000 = Queue Occupancy: 0
          Ingress Timestamp: 1013184455
          Egress Timestamp: 1013184480
          0000 0000 ..... .... .... = Queue ID: 0
          .... .... 0000 0000 0000 0000 0000 = Queue Congestion: 0
        > Switch metadata #1
          Switch ID: 204
          Ingress port ID: 3
          Egress port ID: 1
          Hop Latency: 7
          0000 0000 ..... .... .... = Queue ID: 0
          .... .... 0000 0000 0000 0000 0000 = Queue Occupancy: 0
          Ingress Timestamp: 1014186277
          Egress Timestamp: 1014186284
          0000 0000 ..... .... .... = Queue ID: 0
          .... .... 0000 0000 0000 0000 0000 = Queue Congestion: 0
      > INT Tail Header

```

Switch #3
Switch #2

Switch #1



③ P4によるINTの実現性の確認 (3/3)

ONOSのINT設定画面

Collector IP: 127.0.0.1 | Collector Port: 54321

Src Address: 10.0.0.11 | Dst Address: 10.0.0.22 | Src Port: | Dst Port: 5001 | Protocol: UDP

Switch Id Port Id Hop Latency Queue Occupancy Ingress Timestamp Egress Timestamp Egress Port Tx Utilization

取得情報の選択が可能

Installed INT Intents (1 total)

ID	SRC ADDRESS	DST ADDRESS	SRC PORT	DST PORT
1	10.0.0.11/32	10.0.0.22/32	N/A	5001

ONOS上のトポロジー図

Topology details: device:bmv2:s21 to device:bmv2:s11 (28.65 ms), device:bmv2:s11 to device:bmv2:s12 (108.4 ms), device:bmv2:s12 to device:bmv2:s22 (31.55 ms), device:bmv2:s21 to device:bmv2:s22 (29.41 ms).

Grafana画面

Metrics displayed: Flow Latency, Hop Latency (s11), Hop Latency (s21), Hop Latency (s12). All metrics show high volatility between 0 and 400ms.



P4検証を通して、わかつたこと

- P4Runtimeの実装が進んできている。
- P4プログラミングは簡単。
 - チュートリアルが整っている。
 - 数百行程度のコーディングで任意の機能がスイッチで実現できる。
- スイッチの環境構築に苦労
- 現在は、デバイス依存箇所が存在
 - ポート番号、リソース量等のデバイス依存要素を考慮しなければいけない場合あり
 - 検証時、BarefootスイッチはP4_14にのみ対応 (現在はP4_16にも対応)

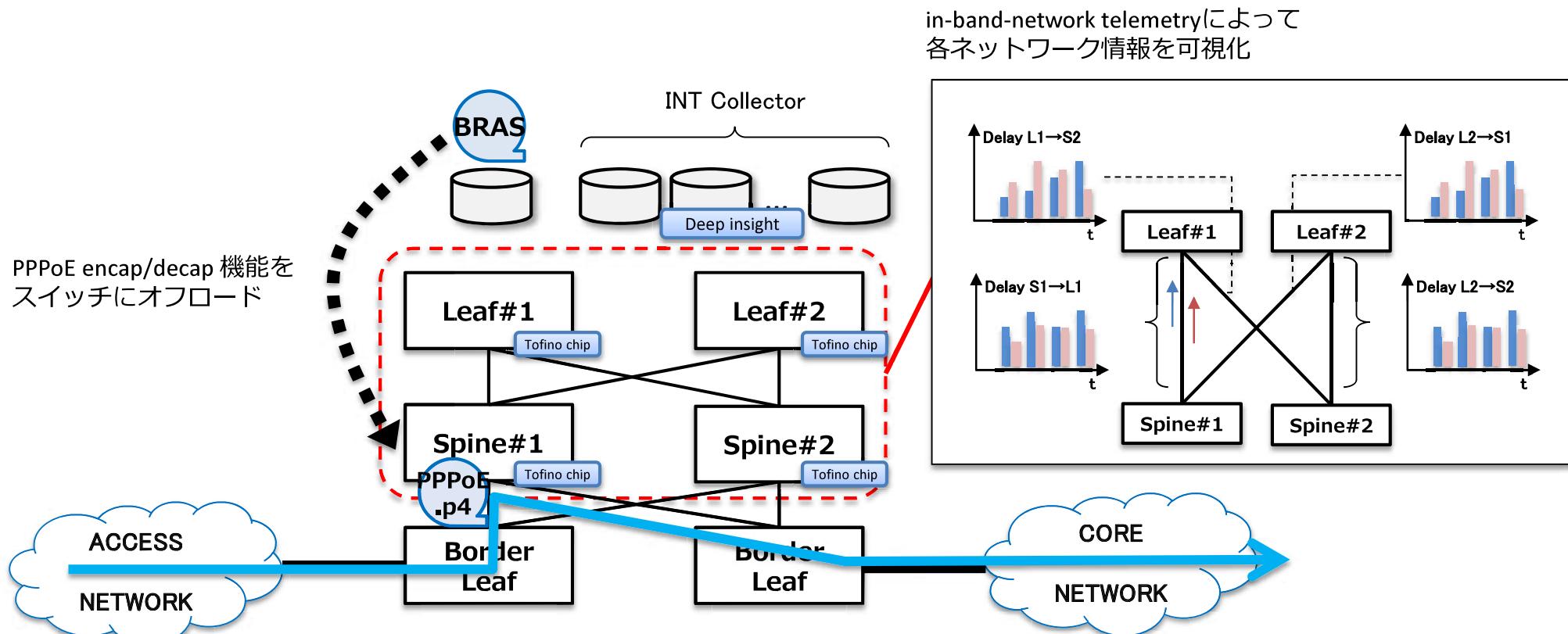


今後の取り組み

PoC @NTT NS Lab.

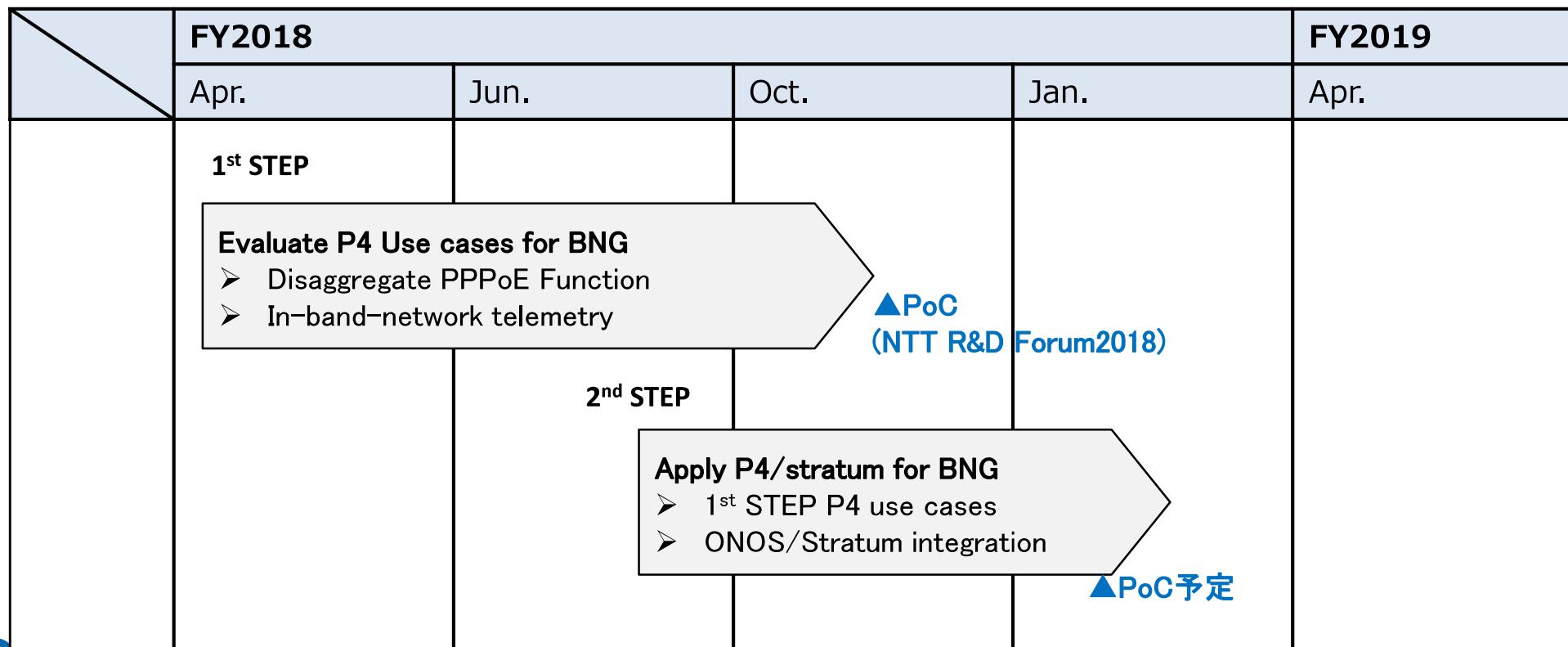
P4とプログラマブルスイッチを用いたBNGのディスアグリゲーションのPoCをNTT NS研で実施予定(11月)

- ① Disaggregate PPPoE/L2TP Function with P4 program
- ② In-band-network telemetry with P4 program



今後のスケジュール

- P4/Stratum活用ユースケースの見極めを継続。
 - ✓ 2つのユースケースについて、NTT R&D Forum (2018.11)でPoC予定
 - ✓ 2nd STEPとして、ONOS/Stratum適応について検討。2019.1頃にPoC予定





まとめ

- P4/Stratumの実装が徐々に進んでいる。
- P4/Stratumを活用のユースケースとして、2つのアプローチを検討。
 - ① VNFのオフロード、②通信品質の可視化
- P4を用いた機能実装は簡単である！