パケット処理の独自実装や高速化手法の比較と実践

Survey and Implementation examples of
custom packet processing and acceleration methods

JANOG45 ＠札幌

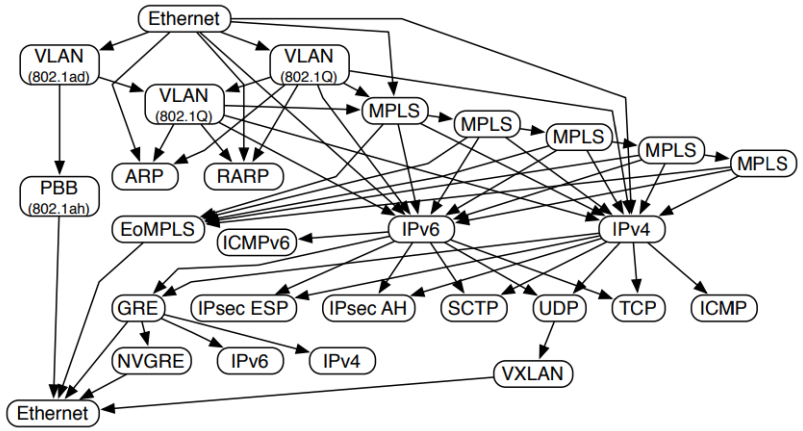
海老澤健太郎
トヨタ自動車株式会社

（補足資料）
P4を用いた実装方法解説

本セッションのお題

"P4" を用いてこれらをプログラムする方法の解説

ヘッダ フォーマットの定義　マッチフィールドの定義　アクションの定義
パースグラフの構築　テーブルタイプの定義　フィールド操作ロジック



- MAC address
- IPv4 address
- proto + TCP ports
- (any header fields)

- drop
- forward
- copy
- push / pop
- add(+) sub(-) multiple(*)
- bit shift (<<) (>>)

→ ヘッダ解析（Parse） → テーブル参照（Match） → 変更・転送（Action） →

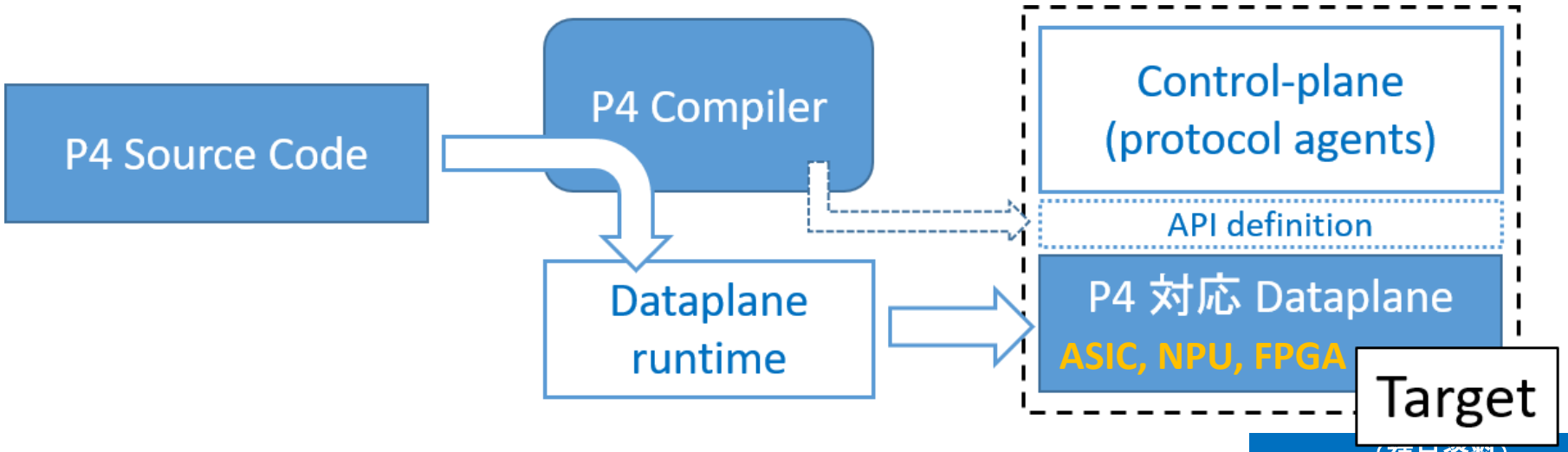
（補足資料）
P4を用いた実装方法解説

P4 … パケット処理に特化したプログラミング言語

"Programming Protocol-Independent Packet Processors"

P4 Source Code	パケット処理パイプラインの定義 パーサーやテーブル、アクション、など
P4 Compiler	P4をTarget上で実行可能な形式にコンパイル Target毎に提供される
Target (P4対応Dataplane)	P4 Dataplane runtime に従いパケットを処理 Hardware: ASIC, NPU, FPGA | Software: CPU

https://p4.org/



（補足資料）
P4を用いた実装方法解説

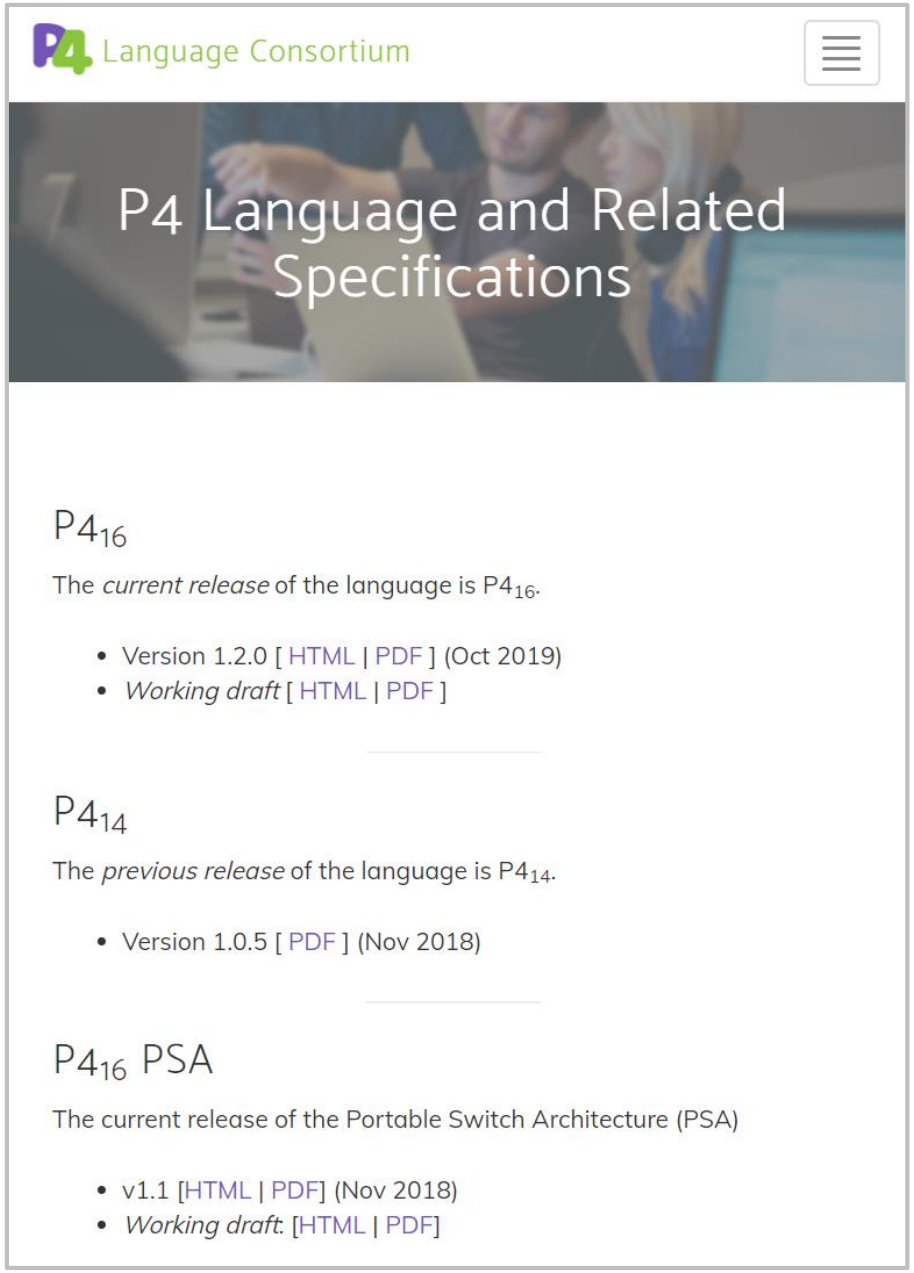
言語仕様（Spec document）

https://p4.org/specs/

P4-16 => プログラミング言語としての仕様

P4-14 => 古い仕様
（まだこちらしかサポートしていない機器もあり）

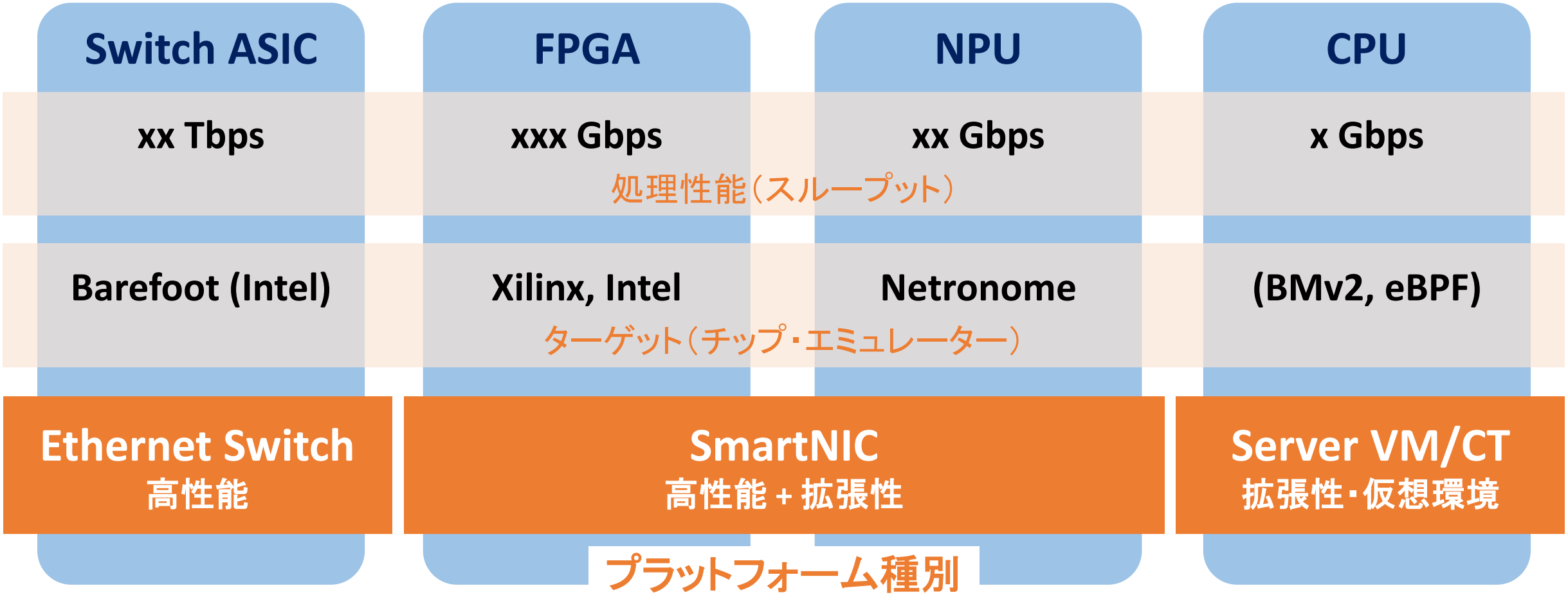
P4-16 PSA => リファレンスアーキテクチャ
（Portable Switch Architecture）



（補足資料）
P4を用いた実装方法解説

P4 実行環境（ターゲット）

P4 で記述したプログラムが動作する環境（チップ・エミュレーター）



（補足資料）
P4を用いた実装方法解説

P4 on ODM Ethernet Switch (Tofino)

様々なハードウェア構成のスイッチ

Original Device Manufacturer (ODM)

Barefoot Networks is pleased to work with leading ODM whitebox and britebox vendors to further disaggregate networking by decoupling datapath intelligence from switch silicon. Please select one of the vendors to learn more:

Edge-corE NETWORKS

Interface Masters TECHNOLOGIES Innovative Network Solutions

Inventec

STORDIS The Open Networking Expert

WNC Wistron NeWeb Corp.

https://barefootnetworks.com/partners/

（補足資料）
P4を用いた実装方法解説

Edge-corE NETWORKS

Wedge100BF-32X/65X

32/65 x QSFP28 ports

CPU Modules
Intel x86 Broadwell-DE
Pentium D-1517

Memory (RAM)
4/8/16 GB SO-DIMM DDR4

Storage
32 GB M.2 SSD

注:参考情報です。現在の仕様は
各メーカーにお問い合わせ下さい

Interface Masters
TECHNOLOGIES
Innovative Network Solutions

Tahoe 2624

24 ports of 100G/40G QSFP28
& 20 ports of 25G/10G SFP 28

Two Rear-Facing I/Os Supporting
Powerful XEON-D Offload I/O
Xilinx Virtex UltraScale FPGA I/O

Tahoe 2860

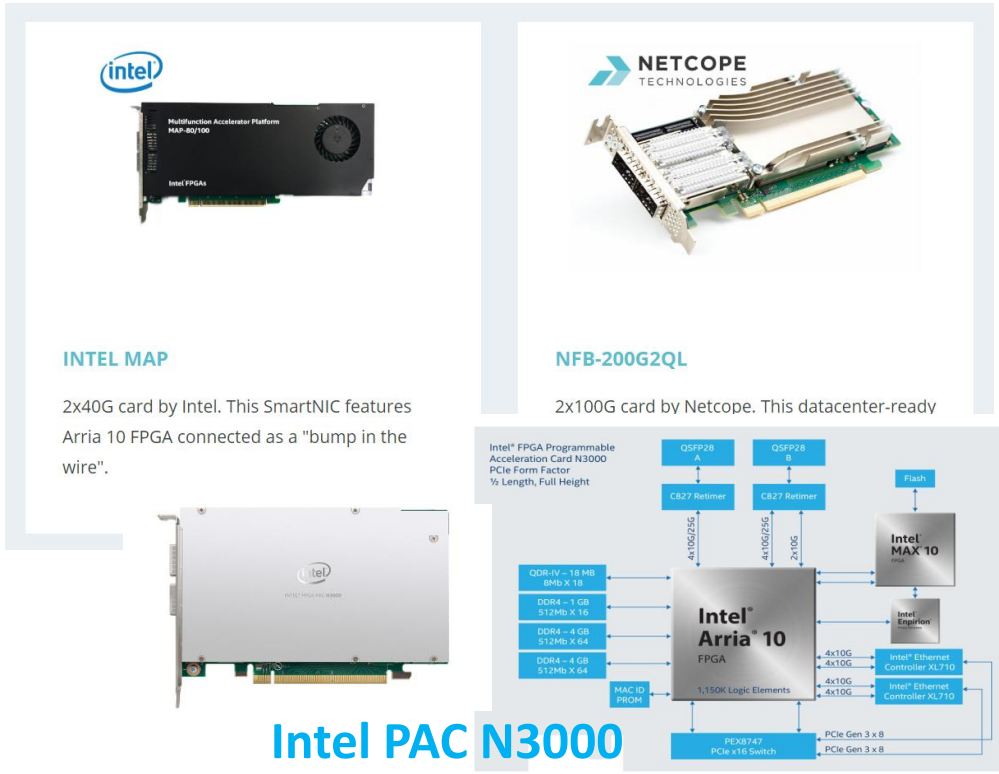
32 ports of QSFP28

Control plane processor options
x86 and Power PC
Data plane processors options
MIPS and Power PC

Edge-Core: https://www.edge-core.com/productsInfo.php?cls=1&cls2=180&cls3=181&id=335 | Interface Masters: https://interfacemasters.com/products/switch-appliances/

P4 on SmartNIC (FPGA/NPU)

NETCOPE TECHNOLOGIES

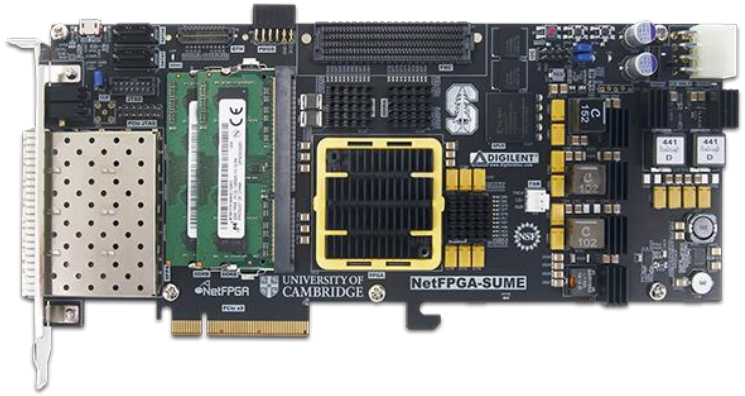


Intel PAC N3000

https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/intel-fpga-pac-n3000/overview.html

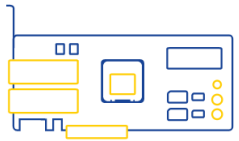
注:参考情報です。現在の仕様は
各メーカーにお問い合わせ下さい

NetFPGA-SUME Virtex-7 FPGA Development Board

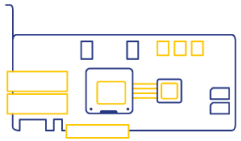


NETRONOME

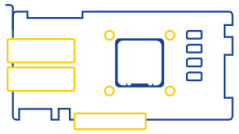
Agilio CX
for Compute Nodes



Agilio FX
for Bare Metal Servers



Agilio LX
for Service Nodes



（補足資料）
P4を用いた実装方法解説

P4 on CPU

BMv2

https://github.com/p4lang/behavioral-model

README.md

BEHAVIORAL MODEL REPOSITORY

build passing

This is the second version of the P4 software switch (aka behavioral model), nicknamed bmv2. It is meant to replace the original version, p4c-behavioral, in the long run, although we do not have feature equivalence yet. Unlike p4c-behavioral, this new version is static (i.e. we do not need to auto-generate new code and recompile every time a modification is done to the P4 program) and written in C++11. For information on why we decided to write a new version of the behavioral model, please look at the FAQ below.

P4 to eBPF compiler

https://github.com/p4lang/p4c/tree/master/backends/ebpf

README.md

eBPF Backend

The back-end accepts only P4_16 code written for the `ebpf_model.p4` filter model. It generates C code that can be afterwards compiled into eBPF (extended Berkeley Packet Filters https://en.wikipedia.org/wiki/Berkeley_Packet_Filter) using clang/llvm or bcc (https://github.com/iovisor/bcc.git).

An older version of this compiler for compiling P4_14 is available at https://github.com/iovisor/bcc/tree/master/src/cc/frontends/p4

Identifiers starting with ebpf_ are reserved in P4 programs, including for structure field names.

Stratum Tutorials: Container of BMv2 + Stratum

https://github.com/stratum/tutorial

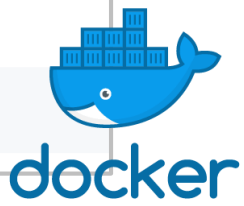
Starting a Mininet topology of Stratum switches

This tutorial uses Docker. If it is not installed, follow these instructions to install it.

There is a Docker container with Mininet and the Stratum BMv2 switch preinstalled: `opennetworking/mn-stratum`

We can use the following command to start the container:

```
docker run --privileged --rm -it -p 50001:50001 opennetworking/mn-stratum
```



（補足資料）
P4を用いた実装方法解説

P4 on CPU

BMv2

https://github.com/p4lang/behavioral-model

README.md

BEHAVIORAL MODEL REPOSITORY

build passing

This is the second version of the P4 software switch (aka behavioral model), nicknamed bmv2. It's meant to replace the original version, p4c-behavioral, in the long run, although we do not have feature equivalence yet. Unlike p4c-behavioral, this new version is static (i.e. we do not r
P4 program) and written in C++11.
please look at the FAQ below.

P4 to eBPF compiler

https://github.com/p4lang/p4c/tree/master/backends/ebpf

README.md

erates C code that can be
/wiki/Berkeley_Packet_Filter) using

ames.

Stratum T

htt

Starting a Minin

This tutorial uses Docker. If it is not installed, follow these instructions to install it.

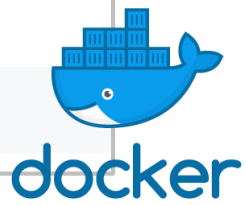
There is a Docker container with Mininet and the Stratum BMv2 switch preinstalled: `opennetworking/mn-stratum`

We can use the following command to start the container:

```
docker run --privileged --rm -it -p 50001:50001 opennetworking/mn-stratum
```

本日のターゲット

BMv2（behavioral-model）：リファレンス実装



（補足資料）
P4を用いた実装方法解説

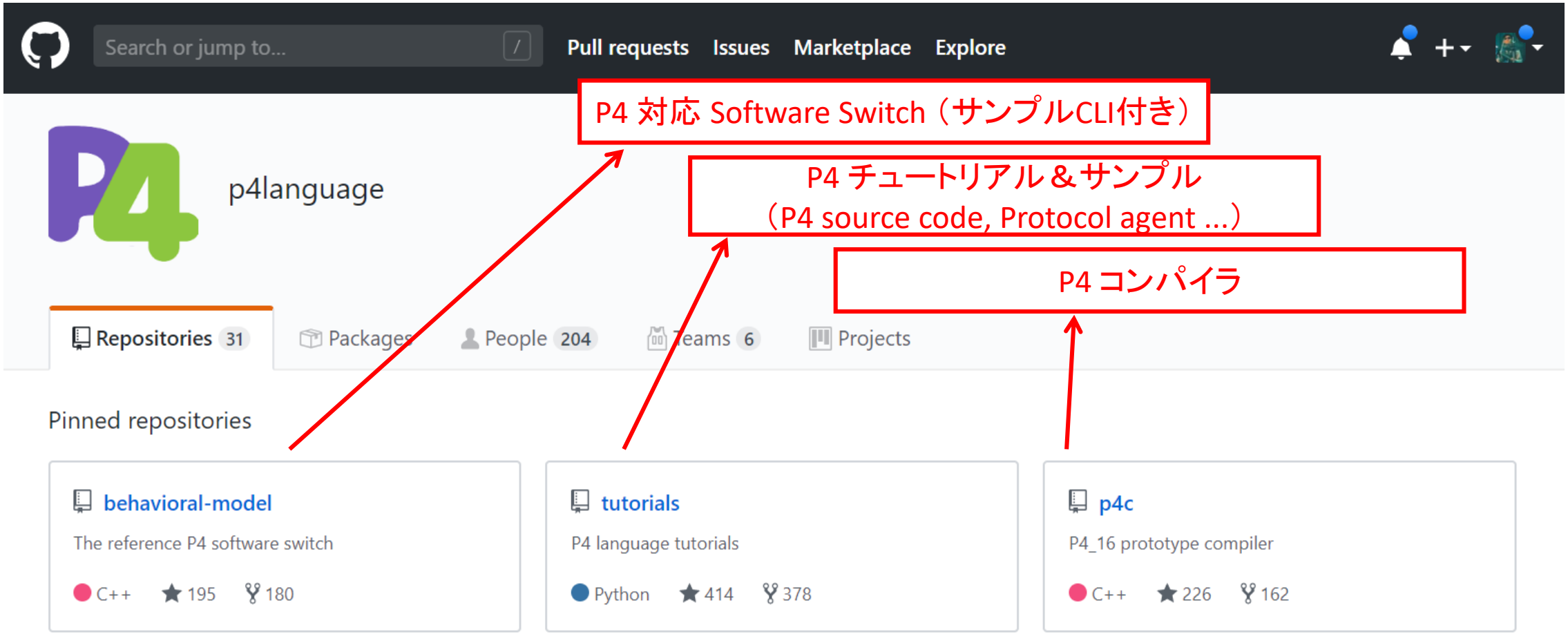
BMv2 + p4c を用いた P4 開発

ここから本題

（補足資料）
P4を用いた実装方法解説

https://github.com/p4lang

本日のターゲット：BMv2（behavioral-model）



P4 ソフトウェア実装｜ビルド・実行手順サンプル

https://www.slideshare.net/kentaroebisawa/how-to-run-p4-bmv2

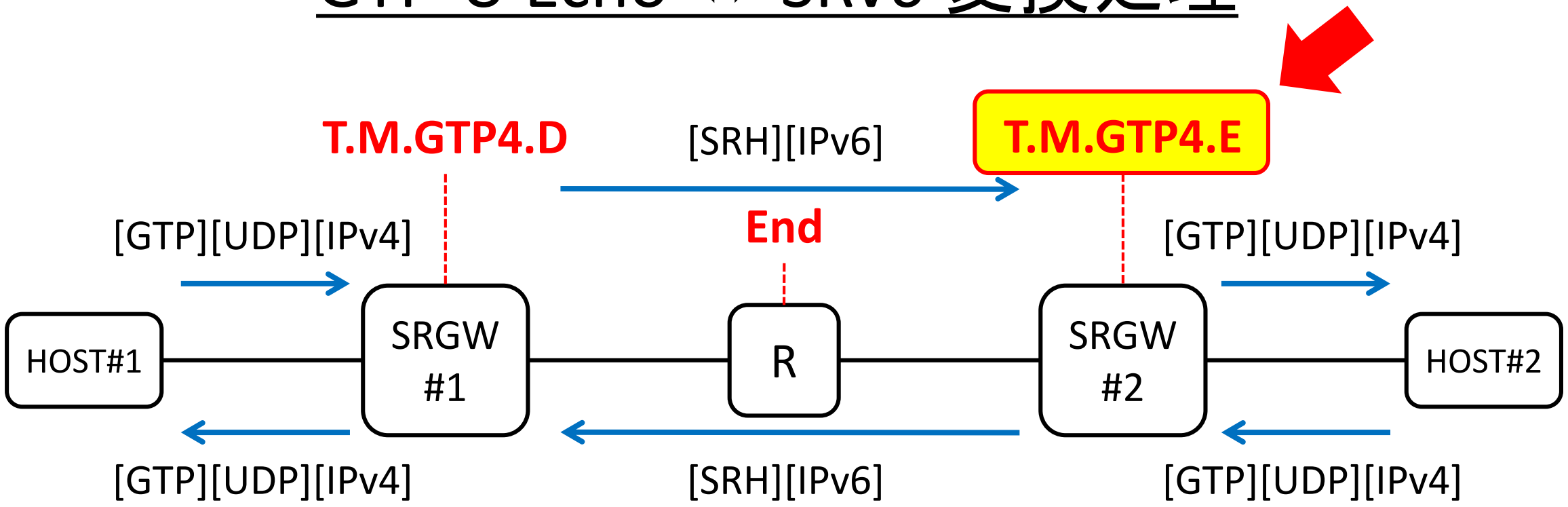
（補足資料）
P4を用いた実装方法解説

P4 Source Code 解説 using "p4srv6"

独自パケット処理の実装例

（補足資料）
P4を用いた実装方法解説

GTP-U Echo ⇔ SRv6 変換処理



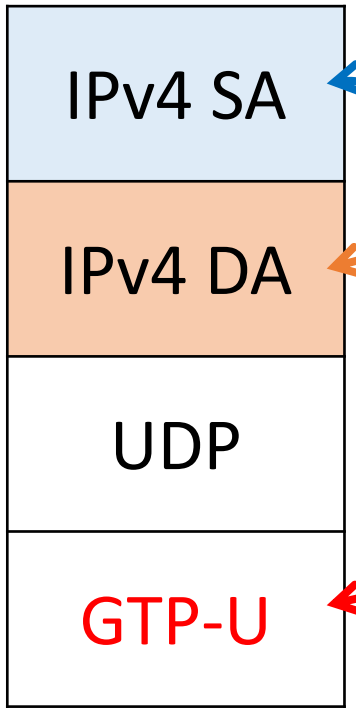
本日は T.M.GTP4.E 実装例（P4 Source Code）を解説

（補足資料）
P4を用いた実装方法解説

GTP IPv4 Addr ⇔ SRv6 Segment ID & Src Address

T.M.GTP4.D

T.M.GTP4.E



Destination Segment ID (Last Destination IPv6 Addr)

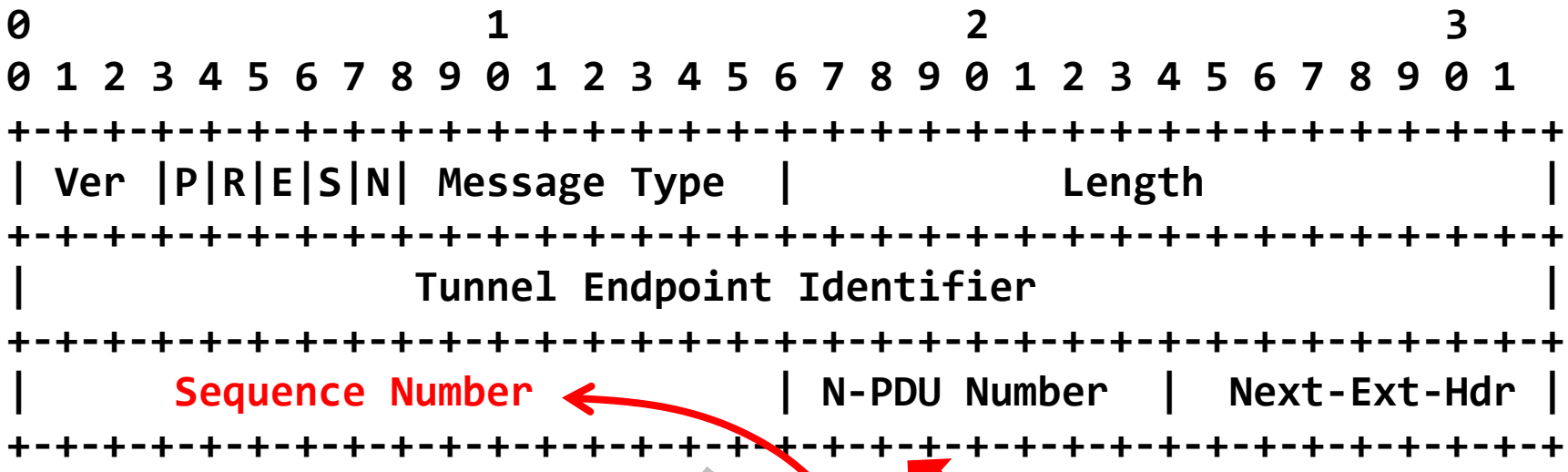
0			127
SRGW-IPv6-LOC-FUNC	IPv4 DA	Args.Mob.Session	0 Padded
128-a-b-c	a	b	c

Source IPv6 Address

0			127
Source UPF Prefix	IPv4 SA	any bit pattern(ignored)	
128-a-b	a	b	

（補足資料）
P4を用いた実装方法解説

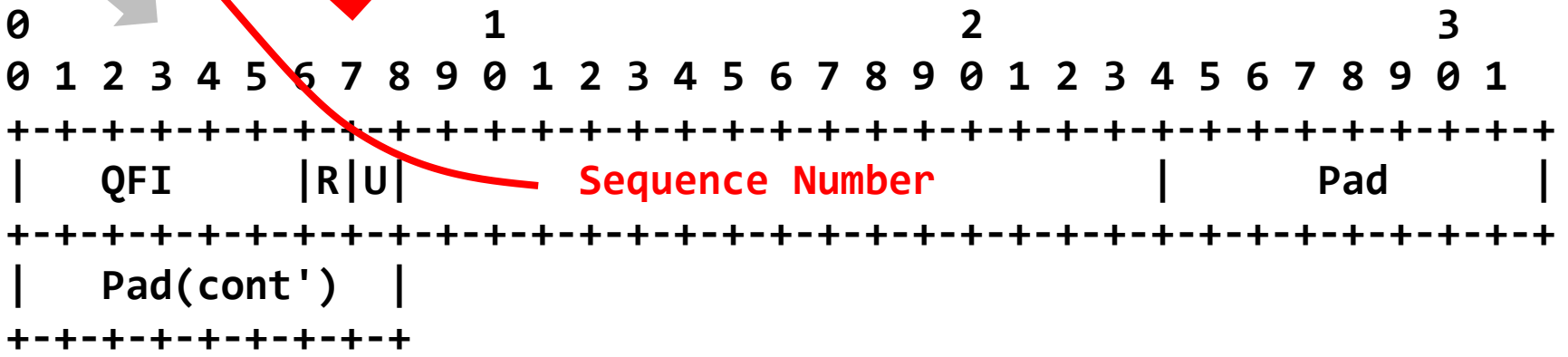
GTP-U Header ⇔ SRv6 Segment ID (Args.Mob.Session)



Echo Request/Response の
TEID==0 のため、Seq を記憶

T.M.GTP4.D

T.M.GTP4.E



Args.Mob.Session format for Echo Request, Echo Reply and Error Indication

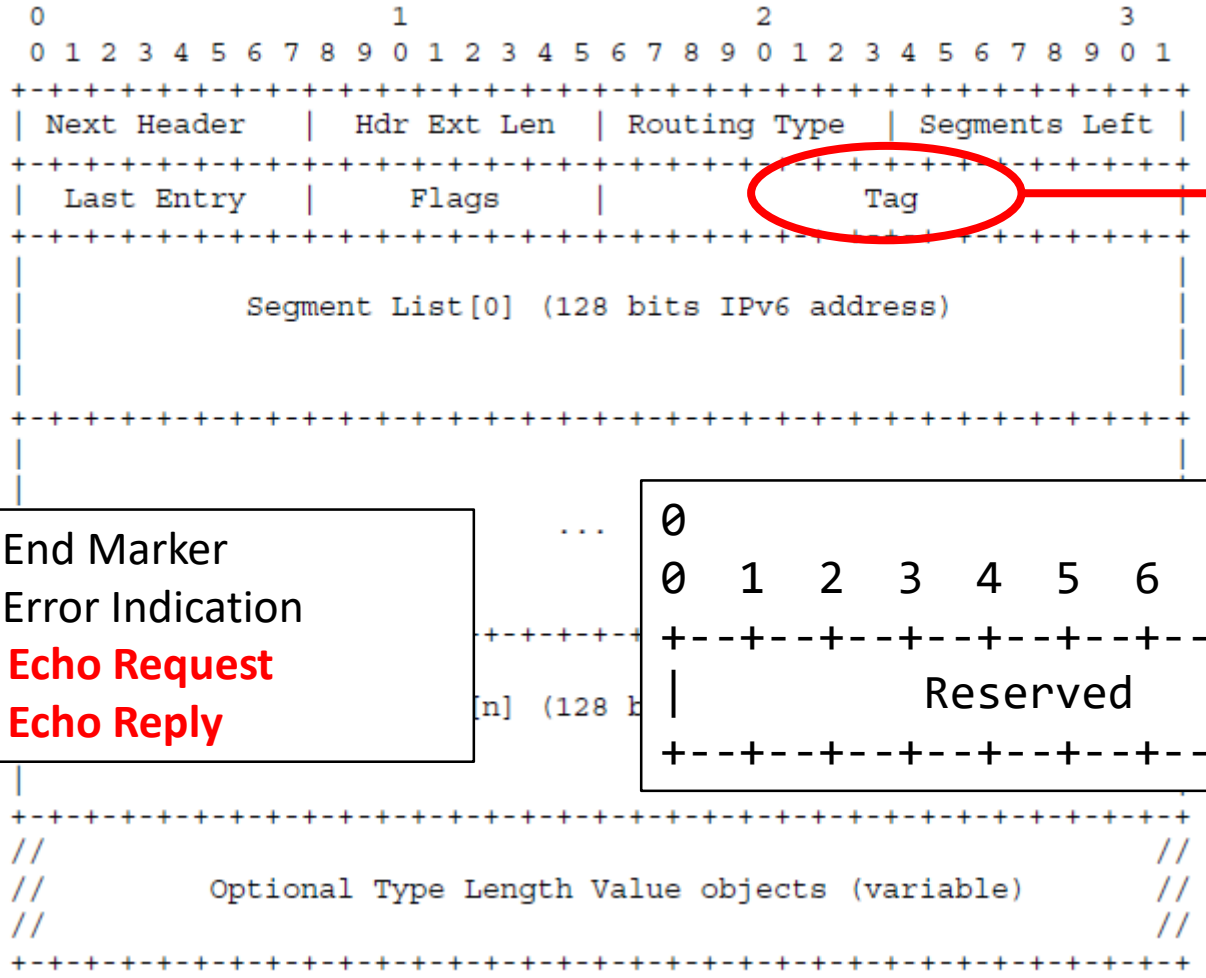
（補足資料）
P4を用いた実装方法解説

GTP-U Message Type を SRH Tag Field からデコード

IPv6 Header

IPv6 Extension Header

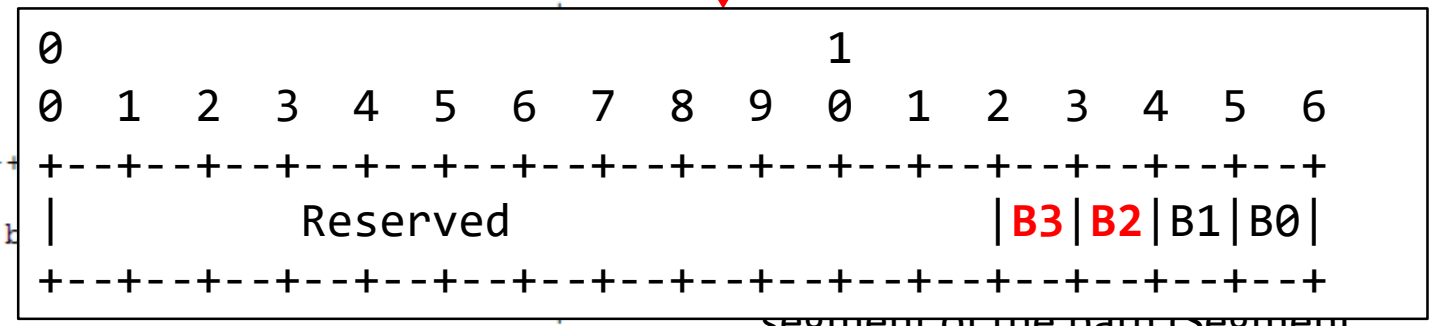
Pay



- Routing Type
  - 4 (Segment Routing)
- Segments Left
  - Index to the next segment in the Segment List
  - Decremented on Endpoint node
- Last Entry

segment of the path (Segment List [0] contains the last segment)

Bit 0 [B0]: End Marker
Bit 1 [B1]: Error Indication
Bit 2 [B2]: Echo Request
Bit 3 [B3]: Echo Reply

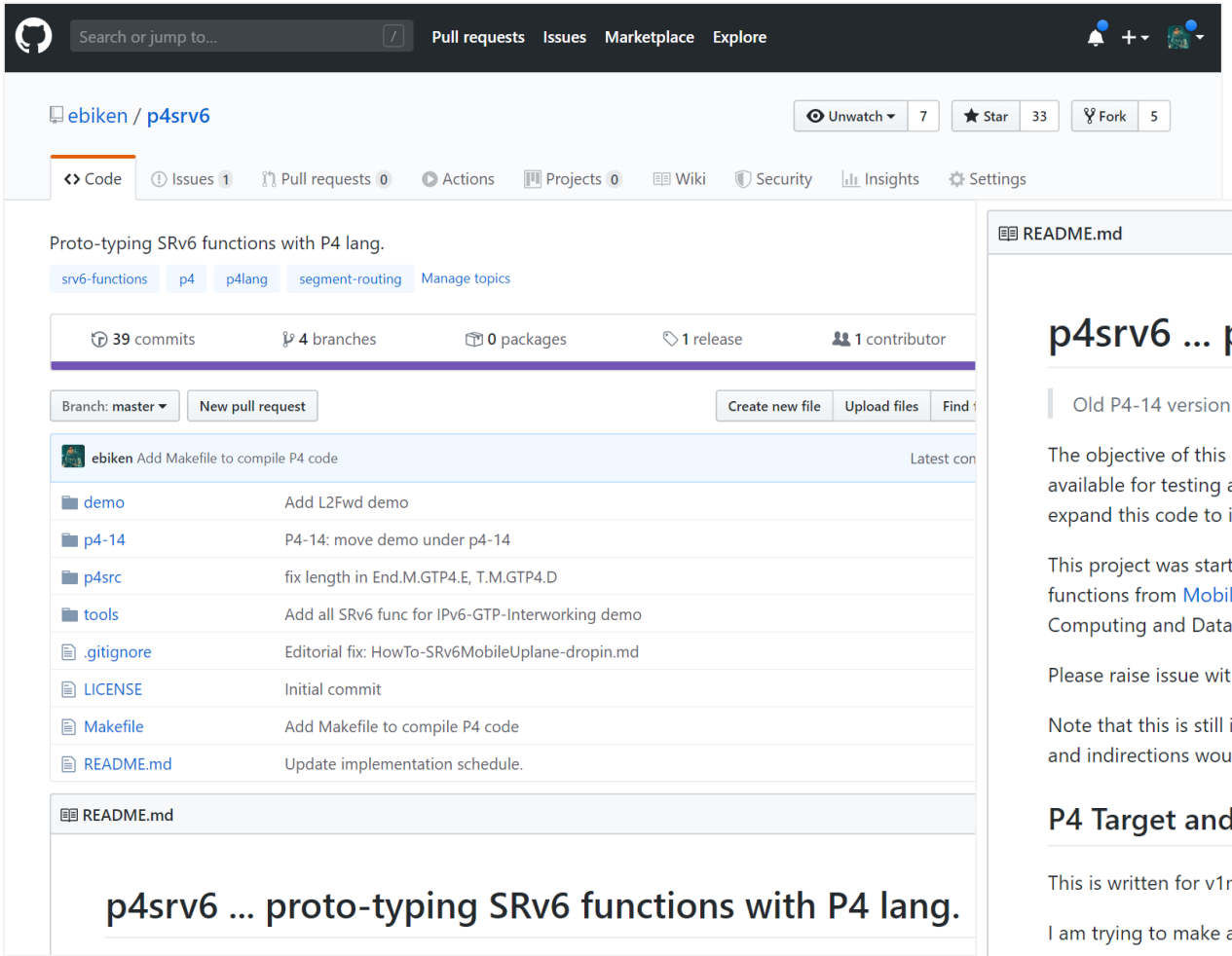


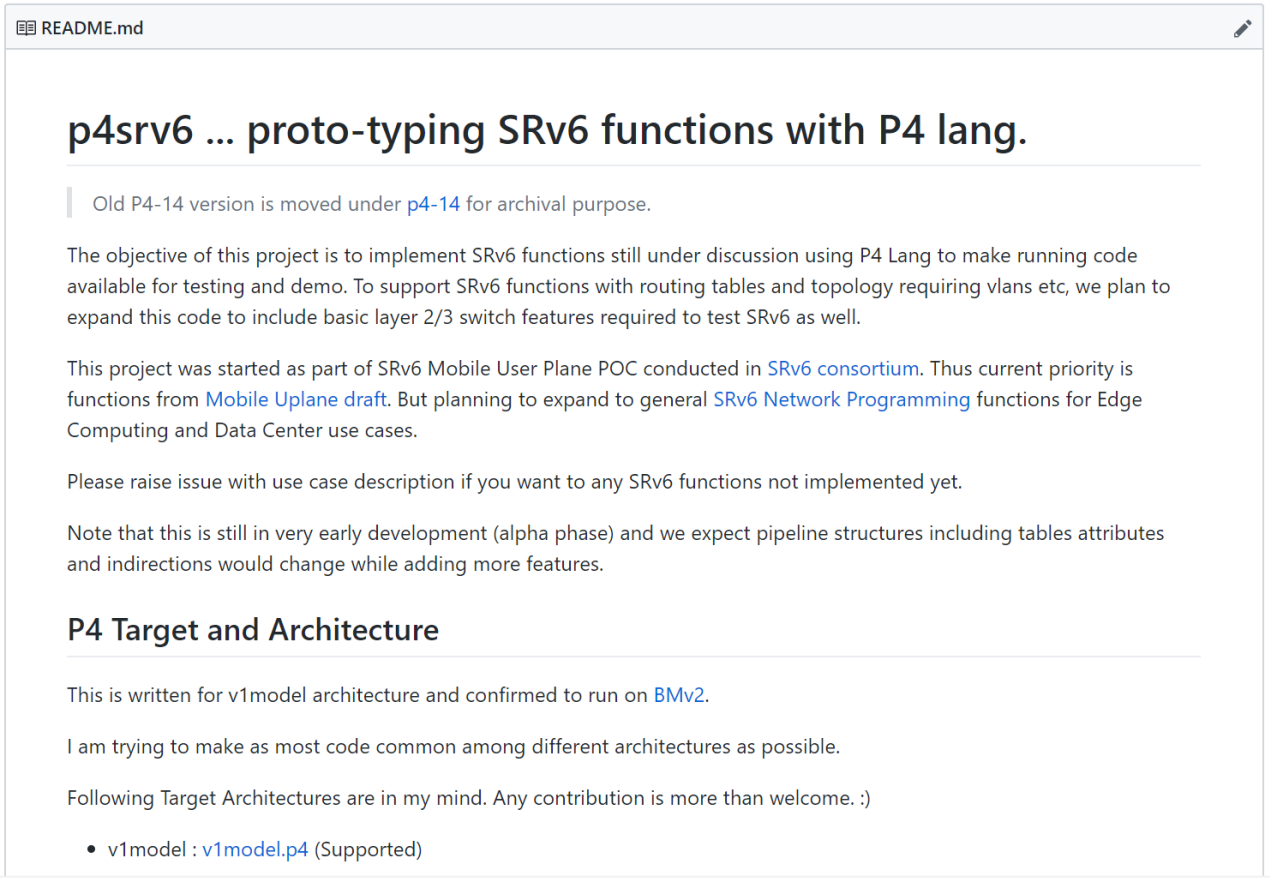
Reference: draft-ietf-6man-segment-routing-header

（補足資料）
P4を用いた実装方法解説

p4srv6 … SRv6 の P4 実装

https://github.com/ebiken/p4srv6





（補足資料）
P4を用いた実装方法解説

p4srv6 ファイル構成

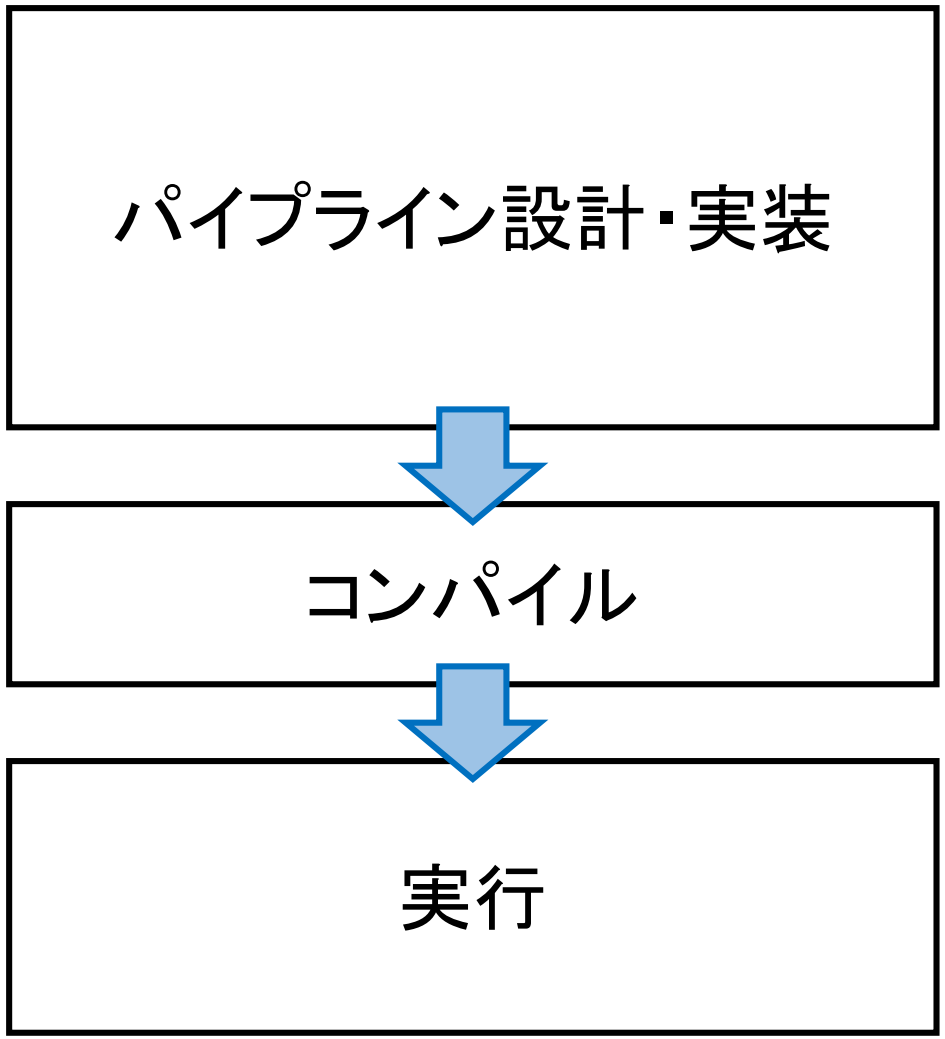
- demo => デモやサンプルコード＆ドキュメント
- p4src => P4 Source Code 本体
  - headers.p4 ... ヘッダ定義
  - parser.p4 ... パーサー
  - srv6.p4 ... SRv6 固有のコード
  - switch.p4 ... スイッチ全体
- tools => 試験環境構築スクリプト
- Makefile ... コンパイルを簡単にする make file
- README.md ... README!!

p4srv6 ファイル構成

- demo => デモやサンプルコード＆ドキュメント
- p4src => P4 Source Code 本体
  - headers.p4 ... ヘッダ定義
  - parser.p4 ... パーサー
  - srv6.p4 ... SRv6 固有のコード
  - switch.p4 ... スイッチ全体
- tools => 試験環境構築スクリプト
- Makefile ... コンパイルを簡単にする make file
- README.md ... README!!

（補足資料）
P4を用いた実装方法解説

P4 開発ステップ



パイプライン全体設計（アーキテクチャ選択）
コントロールの実装（control）
パッケージの実装（package）
ヘッダ定義
パーサー実装
アクション・マッチテーブル実装

p4c の実行： p4c -b <target> <p4 source code> -o <output>
例 : p4c -b bmv2 p4src/switch.p4 -o switch.bmv2.json

（仮想インターフェース・ネームスペース作成）
BMv2 起動
CLI 起動 ⇒ テーブル設定

（補足資料）
P4を用いた実装方法解説

P4 スイッチ・アーキテクチャ

- プログラマブルなブロック（control, parser, deparser）
  - control => Match Action Table, Checksum Computation Block etc.
  - parser ... ヘッダ解析
  - deparser ... ヘッダ出力（再構築）
- インターフェース定義
- ブロックの順番（package）
- ターゲット固有の機能（extern）
  - Buffer/Queue, Hash/Checksum Algorithm, resubmit/recirculate, clone etc.

```
control SwitchIngress(
            inout Header hdr,
            inout UserMetadata user_md,
            inout standard_metadata_t st_md) {
```

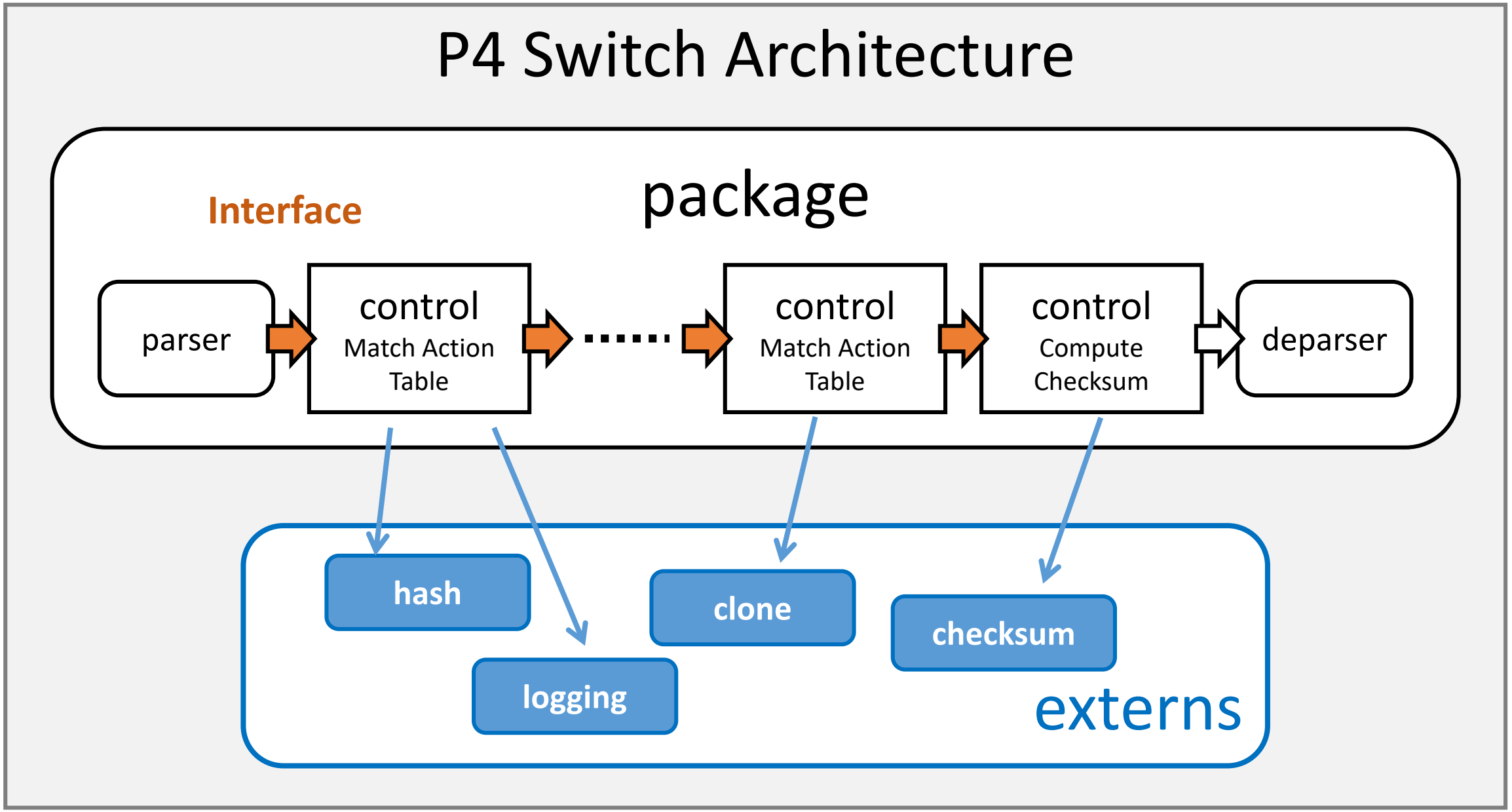
```
control SRv6(
    inout Header hdr,
    inout UserMetadata user_md,
    in PortId_t in_port,
    inout PortId_t egress_port) {
```

インターフェース

これらをまとめて “アーキテクチャ”

（補足資料）
P4を用いた実装方法解説

パイプライン全体設計（アーキテクチャ選択）



（補足資料）
P4を用いた実装方法解説

アーキテクチャ定義ファイル

https://github.com/p4lang/p4c/blob/master/p4include/

v1model.p4

```
package V1Switch<H, M>(Parser<H, M> p,
                       VerifyChecksum<H, M> vr,
                       Ingress<H, M> ig,
                       Egress<H, M> eg,
                       ComputeChecksum<H, M> ck,
                       Deparser<H> dep
                       );
```

p4srv6 では "v1model" を採用

psa.p4

```
package IngressPipeline<IH, IM, NM, CI2EM, RESUBM, RECIRCM>(
    IngressParser<IH, IM, RESUBM, RECIRCM> ip,
    Ingress<IH, IM> ig,
    IngressDeparser<IH, IM, CI2EM, RESUBM, NM> id);

package EgressPipeline<EH, EM, NM, CI2EM, CE2EM, RECIRCM>(
    EgressParser<EH, EM, NM, CI2EM, CE2EM> ep,
    Egress<EH, EM> eg,
    EgressDeparser<EH, EM, CE2EM, RECIRCM> ed);

package PSA_Switch<IH, IM, EH, EM, NM, CI2EM, CE2EM, RESUBM, RECIRCM> (
    IngressPipeline<IH, IM, NM, CI2EM, RESUBM, RECIRCM> ingress,
    PacketReplicationEngine pre,
    EgressPipeline<EH, EM, NM, CI2EM, CE2EM, RECIRCM> egress,
    BufferingQueueingEngine bqe);
```

p4srv6 Pipeline (p4srv6/p4src/switch.p4)

Architecture: v1model.p4

SwitchParser()

```
SwitchParser(
    packet_in pkt,
    out Header hdr,
    inout UserMetadata user_md,
    inout standard_metadata_t st_md)

state start {
    transition parse_ethernet;
}
```

イーサネットヘッダの解析から開始
（ターゲット固有のヘッダは無し）

SwitchVerifyChecksum()

```
SwitchVerifyChecksum(
    inout Header hdr,
    inout UserMetadata user_md)
```

< 実装無し >

SwitchIngress()

```
SwitchIngress(
    inout Header hdr,
    inout UserMetadata user_md,
    inout standard_metadata_t st_md)

table local_mac;
PortFwd() port_fwd;
SRv6() srv6;
L2Fwd(1024) l2fwd;
```

メインとなるコントロールブロック
（パケットヘッダやメタデータの操作）

v1model.p4 ... p4-14 compatible pipeline

```
package V1Switch<H, M>(
        Parser<H, M> p,
        VerifyChecksum<H, M> vr,
        Ingress<H, M> ig,
        Egress<H, M> eg,
        ComputeChecksum<H, M> ck,
        Deparser<H> dep);
```

SwitchEgress()

```
SwitchEgress(
    inout Header hdr,
    inout UserMetadata user_md,
    inout standard_metadata_t st_md)
```

< 実装無し >

SwitchComputeChecksum()

```
SwitchComputeChecksum(
    inout Header hdr,
    inout UserMetadata user_md)
```

チェックサム計算
専用の Control Block

SwitchDeparser()

```
SwitchDeparser(
    packet_out pkt,
    in Header hdr)

pkt.emit(hdr.ether);
pkt.emit(hdr.ipv6);
pkt.emit(hdr.srh);
pkt.emit(hdr.srh_sid);
pkt.emit(hdr.ipv4);
pkt.emit(hdr.icmp);
pkt.emit(hdr.tcp);
pkt.emit(hdr.udp);
pkt.emit(hdr.gtpu);
pkt.emit(hdr.inner_ether);
pkt.emit(hdr.inner_ipv6);
pkt.emit(hdr.inner_ipv4);
pkt.emit(hdr.inner_tcp);
pkt.emit(hdr.inner_udp);
```

```
>> p4srv6/p4src/switch.p4

#include <core.p4>
#include <v1model.p4>
```

https://github.com/p4lang/p4c/blob/master/p4include/core.p4
https://github.com/p4lang/p4c/blob/master/p4include/v1model.p4

（補足資料）
P4を用いた実装方法解説

コントロールの定義（構文）

```
control CONTROL(INTERFACE) {
    CONTROL_INSTANCES
    action ACTION { }
    table TABLE {
        key = { }
        actions = { ACTIONs }
    }
    apply {
        EXTERN();
        CONTROL_INSTANCES.apply()
    }
}
```

中身から解説
（詳細は後ほど）

ヘッダ定義 (headers.p4)

```
typedef bit<48> EthernetAddress;
typedef bit<32> IPv4Address;
typedef bit<128> IPv6Address;

typedef bit<16> EthernetType;
const EthernetType ETH_P_IPV4 = 16w0x0800;
const EthernetType ETH_P_ARP  = 16w0x0806;
const EthernetType ETH_P_VLAN = 16w0x8100;
const EthernetType ETH_P_IPV6 = 16w0x86dd;

// https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml
typedef bit<8> IPProtocol;
const IPProtocol IPPROTO_HOPOPT = 0; // IPv6 Hop-by-Hop Option
const IPProtocol IPPROTO_ICMP = 1;
const IPProtocol IPPROTO_IPV4 = 4;
const IPProtocol IPPROTO_TCP = 6;
const IPProtocol IPPROTO_UDP = 17;
const IPProtocol IPPROTO_IPV6 = 41;
const IPProtocol IPPROTO_ROUTE = 43; // Routing Header for IPv6
const IPProtocol IPPROTO_FRAG = 44; // Fragment Header for IPv6
const IPProtocol IPPROTO_GRE = 47;
const IPProtocol IPPROTO_ICMPv6 = 58; // ICMP for IPv6
const IPProtocol IPPROTO_NONXT = 59; // No Next Header for IPv6
```

各アドレスのビット長を定義

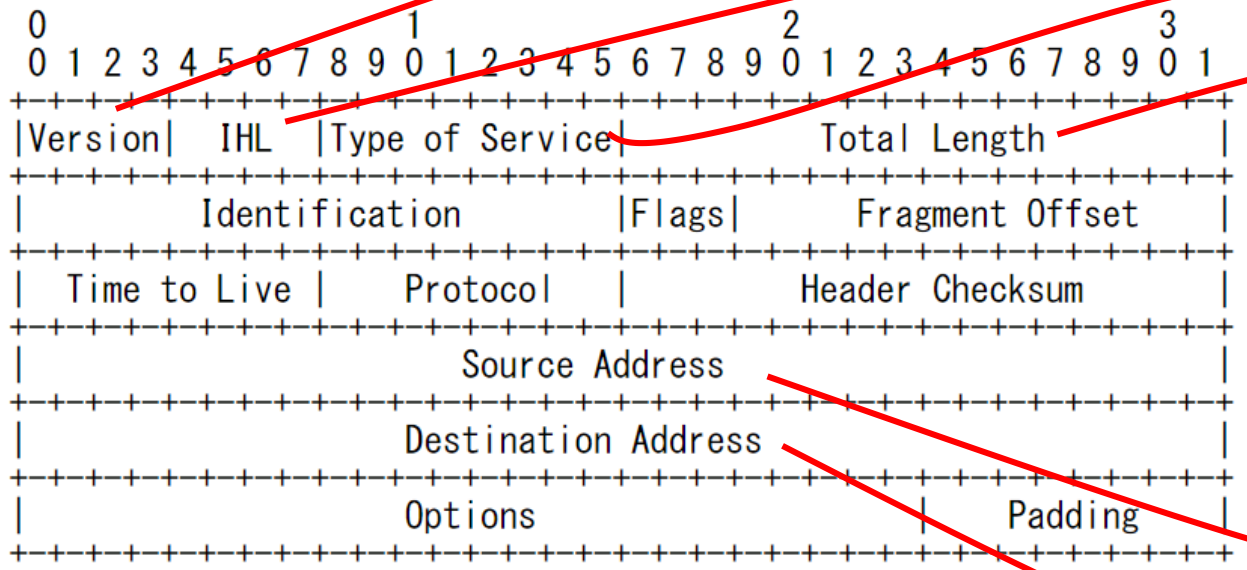
```
header Ethernet_h {
    EthernetAddress dstAddr;
    EthernetAddress srcAddr;
    EthernetType etherType;
}

header IPv4_h {
    bit<4> version;
    bit<4> ihl;
    bit<8> diffserv;
    bit<16> totalLen;
    bit<16> identification;
    bit<3> flags;
    bit<13> fragOffset;
    bit<8> ttl;
    IPProtocol protocol;
    bit<16> hdrChecksum;
    IPv4Address srcAddr;
    IPv4Address dstAddr;
}
```

ヘッダ定義 (headers.p4)

```
typedef bit<48> EthernetAddress;
```

3.1. Internet Header Format

A summary of the contents of the internet header follows:



Example Internet Datagram Header

Figure 4.

```
header IPv4_h {
    bit<4> version;
    bit<4> ihl;
    bit<8> diffserv;
    bit<16> totalLen;
    bit<16> identification;
    bit<3> flags;
    bit<13> fragOffset;
    bit<8> ttl;
    IPProtocol protocol;
    bit<16> hdrChecksum;
    IPv4Address srcAddr;
    IPv4Address dstAddr;
}
```

```
header IPv6_h {
    bit<4> version;
    bit<8> trafficClass;
    bit<20> flowLabel;
    bit<16> payloadLen;
    bit<8> nextHdr;
    bit<8> hopLimit;
    IPv6Address srcAddr;
    IPv6Address dstAddr;
}
```

```
header SRH_h {
    bit<8> nextHdr;
    bit<8> hdrExtLen;
    bit<8> routingType;
    bit<8> segmentsLeft;
    bit<8> lastEntry;
    bit<8> flags;
    bit<12> tag;
    bit<4> gtpMessageType;
}

header SRH_SegmentList_h {
    bit<128> sid;
}
```

```
// GTP User Data Messages (GTPv1)
// 3GPP TS 29.060 V15.3.0 (2018-12) "Table 1: Messages in GTP"
typedef bit<8> GTPv1Type;
const GTPv1Type GTPV1_ECHO    = 1; // Echo Request
const GTPv1Type GTPV1_ECHORES = 2; // Echo Response
const GTPv1Type GTPV1_ERROR = 26; // Error Indication
const GTPv1Type GTPV1_END  = 254; // End Marker
const GTPv1Type GTPV1_GPDU = 255; // G-PDU

// 3GPP TS 29.060 V15.3.0 (2018-12) "6 GTP Header"
header GTPU_h {
    bit<3>  version;       // Version field: always 1 for GTPv1
    bit<1>  pt;            // Protocol Type (PT): GTP(1), GTP'(0)
    bit<1>  reserved;      // always zero (0)
    bit<1>  e;             // Extension Header flag (E)
    bit<1>  s;             // Sequence number flag (S): not present(0), present(1)
    bit<1>  pn;            // N-PDU Number flag (PN)
    GTPv1Type messageType;
    bit<16> messageLen;
    bit<32> teid;          // Tunnel endpoint id
// TODO: support case with no option field (e,s,pn are all zero)
// }
// header GTPU_OPTION_h {
    bit<16> seq;           // Sequence Number
    bit<8>  npdu;          // N-PDU number
    bit<8>  nextExtHdr;  // Next Extention Header Type
}
```


（補足資料）
P4を用いた実装方法解説

パーサーの実装（p4src/parser.p4）

```
parser SwitchParser(
            packet_in pkt,
            out Header hdr,
            inout UserMetadata user_md,
            inout standard_metadata_t st_md) {

    state start {
        transition parse_ethernet;
    }
    state parse_ethernet {
        pkt.extract(hdr.ether);
        transition select(hdr.ether.etherType) {
            ETH_P_IPV4 : parse_ipv4;
            ETH_P_IPV6 : parse_ipv6;
            //ETH_P_ARP  : parse_arp;
            //ETH_P_VLAN : parse_vlan;
            default : accept;
        }
    }
```

①

②

②

```
state parse_ipv4 {
    pkt.extract(hdr.ipv4);
    transition select(hdr.ipv4.protocol) {
        IPPROTO_TCP : parse_tcp;
        IPPROTO_UDP : parse_udp;
        default : accept;
    }
}
```

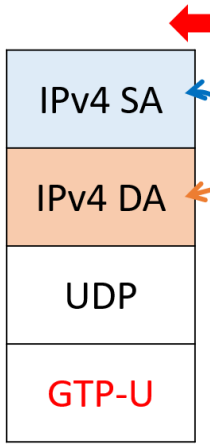
```
state parse_ipv6 {
    pkt.extract(hdr.ipv6);
    transition select(hdr.ipv6.nextHdr) {
        IPPROTO_TCP : parse_tcp;
        IPPROTO_UDP : parse_udp;
        IPPROTO_ROUTE : parse_srh;
        IPPROTO_IPV4 : parse_inner_ipv4;
        IPPROTO_IPV6 : parse_inner_ipv6;
        default : accept;
    }
}
```


（補足資料）
P4を用いた実装方法解説

アクションの実装 End.M.GTP4.E (srv6.p4)

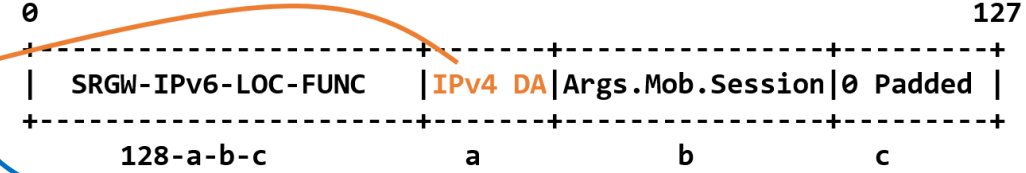
```
action end_m_gtp4_e() {
    hdr.ether.etherType = ETH_P_IPV4;
    hdr.ipv4.setValid();
    hdr.ipv4.version = 4w4;
    hdr.ipv4.ihl = 4w5;
    hdr.ipv4.diffserv = 8w0;
    hdr.ipv4.totalLen = hdr.ipv6.payloadLen - 16w8 - (bit<16>)hdr.srh.hdrExtLen*8 + 16w40;
    hdr.ipv4.identification = 16w0;
    hdr.ipv4.flags = 3w0;
    hdr.ipv4.fragOffset = 13w0;
    hdr.ipv4.ttl = hdr.ipv6.hopLimit;
    hdr.ipv4.protocol = IPPROTO_UDP;
    // IPv4 header checksum will be calculated later.
    hdr.ipv4.srcAddr = hdr.ipv6.srcAddr[63:32];
    hdr.ipv4.dstAddr = hdr.ipv6.dstAddr[95:64];
```

IPv4

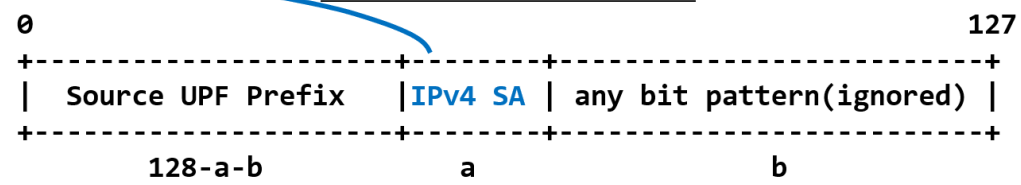


T.M.GTP4.E

Destination Segment ID (Last Destination IPv6 Addr)



Source IPv6 Address



（補足資料）
P4を用いた実装方法解説

アクションの実装 End.M.GTP4.E (srv6.p4)

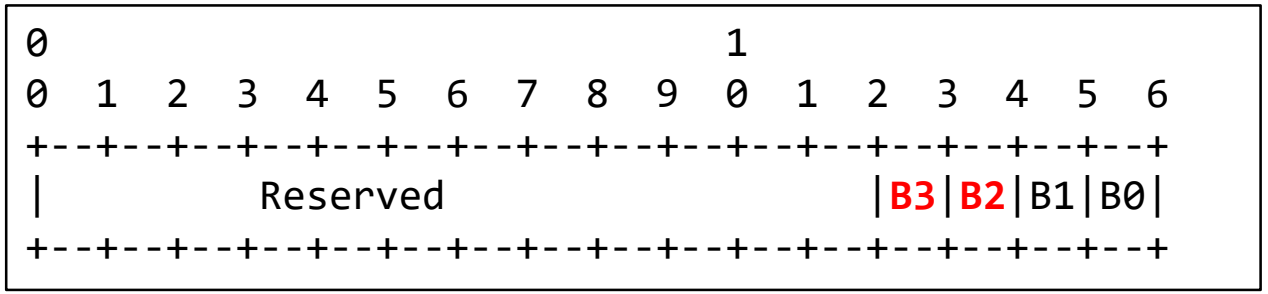
```
... end_m_gtp4_e ...
```

UDP

```
hdr.udp.setValid();
hdr.udp.srcPort = UDP_PORT_GTPC; // 16w2123
hdr.udp.dstPort = UDP_PORT_GTPC; // 16w2123
hdr.udp.length = hdr.ipv6.payloadLen + 16w20 -16w40; // Payload + UDP(8) + GTP(12)
```

GTP

```
hdr.gtpu.setValid();
hdr.gtpu.version = 3w1;
hdr.gtpu.pt = 1w1;
hdr.gtpu.reserved = 1w0;
hdr.gtpu.e = 1w0; // No Extention Header
hdr.gtpu.s = 1w1; // YES Sequence number
hdr.gtpu.pn = 1w0;
// ECHO(tag=4, gtpType=1), ECHORES(tag=8, gtpType=2)
hdr.gtpu.messageType = (bit<8>)(hdr.srh.tag / 4);
```



Bit 0 [B0]: End Marker
Bit 1 [B1]: Error Indication
Bit 2 [B2]: Echo Request
Bit 3 [B3]: Echo Reply

（補足資料）
P4を用いた実装方法解説

アクションの実装 End.M.GTP4.E (srv6.p4)

```
... end_m_gtp4_e ...
        // IPv6 Payload length - length of extention headers + GTP optional headers(4)
        hdr.gtpu.messageLen = hdr.ipv6.payloadLen - 16w8 - (bit<16>)hdr.srh.hdrExtLen*8 + 16w4;
        hdr.gtpu.teid = 32w0;
        hdr.gtpu.seq = hdr.ipv6.dstAddr[55:40];
        hdr.gtpu.npdu = 8w0;
        hdr.gtpu.nextExtHdr = 8w0;
        // remove IPv6/SRH headers
        remove_srh_header();
        hdr.ipv6.setInvalid();
}
```

GTP

削除

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Ver |P|R|E|S|N| Message Type  |           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Tunnel Endpoint Identifier                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Sequence Number          | N-PDU Number  |  Next-Ext-Hdr |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Echo Request/Response の TEID==0 のため、Seq を記憶

T.M.GTP4.D

T.M.GTP4.E

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   QFI     |R|U|          Sequence Number              |      Pad      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Pad(cont')  |
+-+-+-+-+-+-+-+-+
```

T.M.GTP4.E

IPv4 SA

IPv4 DA

UDP

GTP-U

Destination Segment ID (Last Destination IPv6 Addr)

```
0                                                              127
+----------------------+-------+---------------+---------+
|  SRGW-IPv6-LOC-FUNC  |IPv4 DA|Args.Mob.Session|0 Padded |
+----------------------+-------+---------------+---------+
      128-a-b-c           a          b           c
```

Source IPv6 Address

```
0                                                              127
+----------------------+--------+-------------------------+
|  Source UPF Prefix   |IPv4 SA | any bit pattern(ignored) |
+----------------------+--------+-------------------------+
        128-a-b           a                  b
```


（補足資料）
P4を用いた実装方法解説

マッチテーブルの実装 (srv6.p4)

```
table srv6_end { // localsid
    key = {
        hdr.ipv6.dstAddr : ternary;
    }
    actions = {
        @defaultonly NoAction;
        // SRv6 Network Program  : draft-filsfils-spring-srv6-network-programming
        end;                     // End
        // SRv6 Mobile Userplane : draft-ietf-dmm-srv6-mobile-uplane
        end_m_gtp4_e;            // End.M.GTP4.E
        // Proxy Functions : draft-xuclad-spring-sr-service-programming
        end_am;
    }
    const default_action = NoAction;
}
```

Current SID ⇒ 宛先IPv6アドレスにマッチ

選択可能なアクションのリスト

SRv6() Control の実装

```
control SRv6(
    inout Header hdr,
    inout UserMetadata user_md,
    in PortId_t in_port,
    inout PortId_t egress_port) {
 ...
    action end_m_gtp4_e() {
 ...
    table srv6_end { // localsid
 ...
    apply {
 ...
        if(srv6_end.apply().hit) {
 ...
}
```

action の定義

table の定義（ match + action)

table の実行

（補足資料）
P4を用いた実装方法解説

SRv6() Control の実装

```
control SRv6(
    inout Header hdr,
    inout UserMetadata user_md,
    in PortId_t in_port,
    inout PortId_t egress_port) {
```

table srv6_end の実行
（HITしたら action end_m_gtp4_e の実行）

parser でセットされた gtp_message_type の値に応じてGTP-U Type の値をセット

```
apply {
    if (hdr.srh.isValid()) {
        srv6_set_nextsid.apply();
    }
    if (hdr.ipv6.isValid()) {
        if(!srv6_end_iif.apply().hit) {
            if(srv6_end.apply().hit) {
                // draft-murakami-dmm-user-plane-message-encoding
                if (user_md.srv6.gtp_message_type == 1) { // TODO:
                    set_gtpu_type(GTPV1_END);
                } else if (user_md.srv6.gtp_message_type == 2) {
                    set_gtpu_type(GTPV1_ERROR);
                } else if (user_md.srv6.gtp_message_type == 4) {
                    set_gtpu_type(GTPV1_ECHO);
                } else if (user_md.srv6.gtp_message_type == 8) {
                    set_gtpu_type(GTPV1_ECHORES);
                }
            } else {
                srv6_transit_v6.apply();
            }
        }
```


（補足資料）
P4を用いた実装方法解説

コントロールの定義 (switch.p4)

```
control SwitchIngress(
            inout Header hdr,
            inout UserMetadata user_md,
            inout standard_metadata_t st_md) {
    // Instantiate controlls
    PortFwd() port_fwd;
    SRv6() srv6;
    L2Fwd(1024) l2fwd;
    action local_mac_hit() { // L3/Host処理
    }
    table local_mac {
        key = { hdr.ether.dstAddr : exact; }
        actions = {
            NoAction; local_mac_hit();
        }
        const default_action = NoAction;
    }
```

各種 control のインスタンス化

```
control CONTROL(INTERFACE) {
    CONTROL_INSTANCES
    action ACTION { }
    table TABLE {
        key = { }
        actions = { ACTIONs }
    }
    apply {
        EXTERN();
        CONTROL_INSTANCES.apply()
    }
}
```

```
... cont ...
 apply {
        mark_to_drop(st_md);
        port_fwd.apply(st_md.ingress_port, st_md.egress_spec);
        srv6.apply(hdr, user_md, st_md.ingress_port, st_md.egress_spec);
        l2fwd.apply(hdr.ether.dstAddr, user_md.ig_md, st_md.egress_spec);
    }
}
```

各種 control を実行

パッケージの実装 (switch.p4)

```
V1Switch(SwitchParser(),
         //SwitchVerifyChecksum(),
         NoSwitchVerifyChecksum(),
         SwitchIngress(),
         SwitchEgress(),
         SwitchComputeChecksum(),
         SwitchDeparser()) main;
```

p4srv6 動作サンプル

GTP-U <=> SRv6 変換

（補足資料）
P4を用いた実装方法解説

GTP-U <=> SRv6 変換

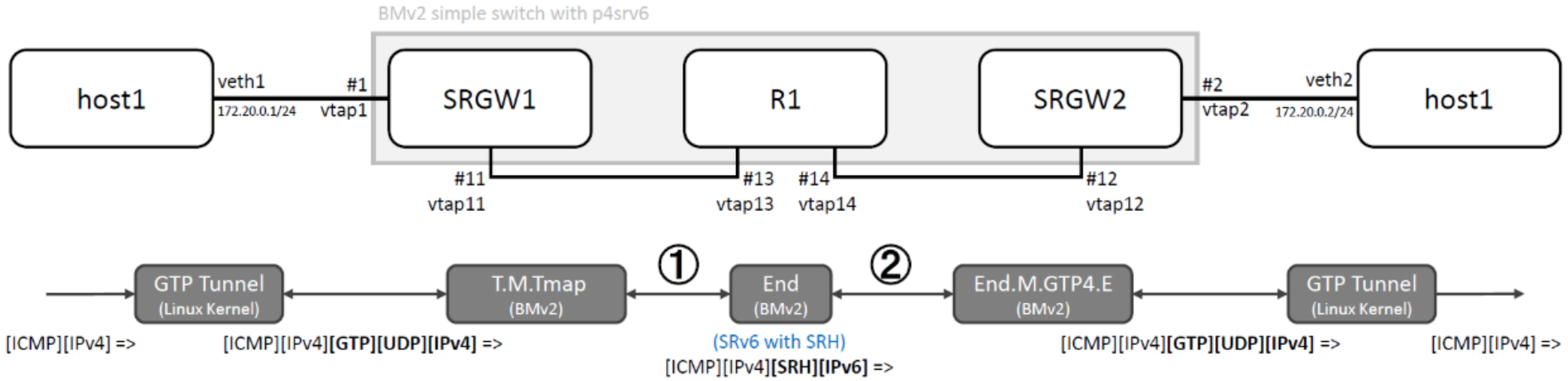
demo1 : How To run SRv6 Mobile Uplane POC (drop in replacement of GTP)

https://github.com/ebiken/p4srv6/blob/ietf106/demo/srv6/demo1-SRv6MobileUplane-dropin.md

diagram

For simplisity, PortFwd table is used to forward packets instead of L2/L3 forwarding tables.

GTP to SRv6 with T.M.Tmap & End.M.GTP4.E (with SRH)



（補足資料）
P4を用いた実装方法解説

BMv2 と p4c のインストール

Install p4c and behavior-model (BMv2)

You can use convenient script `install-p4dev-p4runtime.sh` written by Andy Fingerhut to install p4c and bmv2.

> See original page README-install-troubleshooting.md for the most updated informaiton.

```
$ git clone https://github.com/jafingerhut/p4-guide.git
$ cd p4-guide/bin/
~/p4-guide/bin$ ./install-p4dev-p4runtime.sh
~/p4-guide/bin$ cat p4setup.bash >> ~/.bashrc
```

コンパイル

Clone and Build p4srv6

```
$ git clone https://github.com/ebiken/p4srv6.git
$ cd p4srv6
~/p4srv6$ git checkout v0.01
~/p4srv6$ p4c --target bmv2 --arch v1model p4src/switch.p4
```

仮想インターフェース・ネームスペース作成

```
// host1
run ip netns add host1


# Create veth and vtap
run ip link add veth1 type veth peer name vtap1


run ip link set veth1 netns host1


# Set IP address
run ip netns exec host1 ip addr add 172.20.0.1/24 dev veth1
run ip netns exec host1 ip -6 addr add fd01::1/64 dev veth1
```

```
~/p4srv6/demo/srv6$ sudo ./ns-hosts-srv6-demo1.sh -c
```

```
Usage: ./ns-hosts-srv6-demo1.sh -{c|d} (c: create, d:delete)

netns/veth:
    host1, veth1:172.20.0.1/24, fd01::1/64
    host2, veth2:172.20.0.2/24, fd01::2/64
vtap:
    vtap1, vtap2, vtap11, vtap12, vtap13, vtap14

Create 2 netns with tap interface visible to default ns as vtap1, vtap2.

 ns:host1   ns:host2
 +-------+  +-------+  veth1:172.20.0.1/24
 | veth1 |  | veth2 |
 +---+---+  +---+---+  veth2:172.20.0.2/24
     |          |
   vtap1      vtap2

vtap11,12,13,14 will be also created to link between switch ports.
```

BMv2 & CLI 起動 ⇒ テーブル設定

```
~/p4srv6$ sudo simple_switch switch.json -i 1@vtap1 -i 2@vtap2 -i 11@vtap11 -i 12@vtap12 -i 13@vtap13 -i 14@vtap14 --nanolog ¥
ipc:///tmp/bm-0-log.ipc --log-console -L debug --notifications-addr ¥
ipc:///tmp/bmv2-0-notifications.ipc

$ runtime_CLI.py
table_add portfwd set_egress_port 1 => 11
table_add portfwd set_egress_port 11 => 1
table_add portfwd set_egress_port 12 => 13
table_add portfwd set_egress_port 13 => 12
table_add portfwd set_egress_port 14 => 2
table_add portfwd set_egress_port 2 => 14

$ runtime_CLI.py
table_add srv6_transit_udp t_m_tmap_sid1 2152 => 0xfc345678 0xfd00000000000000000000000000001 0xfd0101000000000000000000000001
table_add srv6_end end 0xfd0101000000000000000000000001&&&0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF => 100
table_add srv6_end end_m_gtp4_e 0xfc34567800000000000000000000000&&&0xFFFFFFFF0000000000000000000000 => 100
```

日本P4ユーザー会の紹介

（補足資料）
P4を用いた実装方法解説

https://p4users.org/

質問・相談は日本P4ユーザー会Slackへ

ホーム · 最新情報 · 日本 P4 ユーザ会 運営委員 · 日本 P4 ユーザ会 会則

日本 P4 ユーザ会

日本 P4 ユーザ会について

日本 P4 ユーザ会は P4 Lang (https://p4.org/) について日本語で語るグループです。

P4 関連のセミナー情報、カンファレンス情報及び技術情報を本ページで共有します。

インタラクティブなディスカッションは p4users-jp.slack.com でどうぞ。【Slack のリンク】

日本 P4 ユーザ会 2019 開催のお知らせ

第2条（目的）

1「日本 P4 ユーザ会」は、 P4 Lang (https://p4.org/) について日本語で語るグループであり、P4 言語について、オープンなディスカッションを出来るインフラを提供し、技術情報の共有、交換を活性化することで、日本の技術者、および P4 利用者に貢献することを目的としたグループである。

第3条（会の活動）

1　前条の目的達成のために、以下の活動を行なう。

(1)メーリングリストもしくは Slack による会員相互の情報交換

(2)ミーティング開催による会員相互の情報交換

(3)P4 設計技術の研究、開発

(4)P4 設計技術に関する技術文書の蓄積

(5)P4 設計技術に関する技術文書の翻訳

(6)その他本会の目的を達成するために必要な活動

第2章 会員

第4条（入会）

1 本会の参加員は、第2条の目的に賛同し、第3条の事業遂行に協力する意思を有する個人、法人、団体とする。

2 本会所定の（メーリングリストもしくは Slack） への参加により会員となる。

（補足資料）
P4を用いた実装方法解説

https://connpass.com/event/155799/

アジェンダ

※アジェンダは基本的に下記の流れになります

13時30分～15時00分

- データプレンプログラミングとは
- P4言語の基礎

15時00分～15時20分

- 休憩

15時20分～16時20分

- 動作環境の説明、事前準備時間

16時20分～17時20分

- Lab1 ハンズオン

17時20分～17時30分

- 本日のまとめ

！！今年も実施予定！！

興味ある人は「参加したい！！」と Slack にコメントお願いします！

（補足資料）
P4を用いた実装方法解説