JANOG46

次世代SDNスイッチStratumと ONOSコントローラー使ったパケット制御

松本 達郎

tatsuro.matsumoto.ty@apresiasystems.co.jp

APRESIA SYSTEMS

自己紹介

- 名前
  - 松本 達郎 (まつもと たつろう)
- 所属
  - APRESIA Systems 株式会社
    次世代技術部 アプリケーション開発グループ
- 業務内容
  - ネットワーク管理システムの開発
  - 次世代技術の調査

目次

- 本日の発表内容
  - Stratumの概要
  - ONOSの概要
  - StratumとONOSを使った海外事例紹介
  - 簡単なサンプルアプリケーション（簡易NAT）を作成
    - データプレーン（Stratum + P4）実装
    - コントロールプレーン（ONOS）実装
    - デモ
  - まとめ

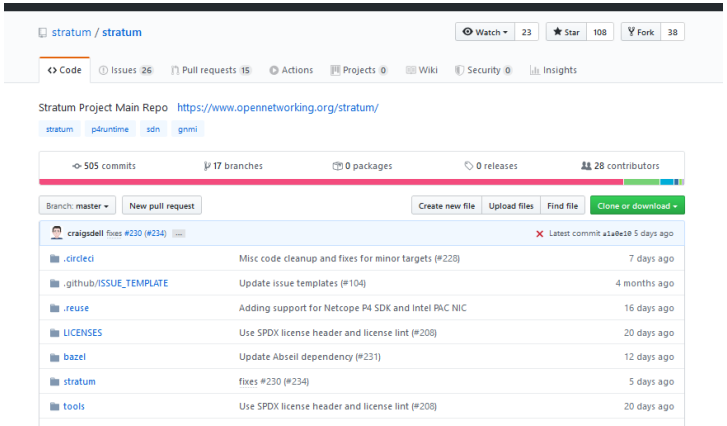
目次

- 本日の発表内容
  - Stratumの概要
  - ONOSの概要
  - StratumとONOSを使った海外事例紹介
  - 簡単なサンプルアプリケーション（簡易NAT）を作成
    - データプレーン（Stratum + P4）実装
    - コントロールプレーン（ONOS）実装
    - デモ
  - まとめ

Stratumとは

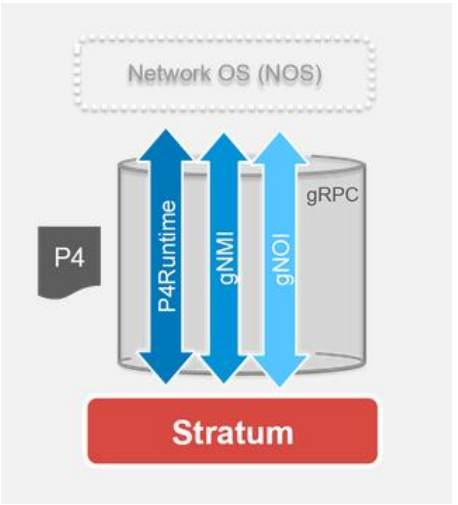
- ホワイトボックススイッチ用のオープンソースなスイッチOS
  - Apache License, Version 2.0
- Google社の内部で利用していたコードの一部をオープンソース化
  - ONFの1プロジェクトとして開発が進められている
- gRPCベースの3つの新しいインターフェース
  - P4Runtime（データプレーンパイプラインの設定/制御）
  - gNMI（デバイスの設定、Telemetry）
  - gNOI（デバイスオペレーション）
- プログラマブルデバイス以外にもFixed Pipeline Modelもサポート
  - Broadcom社のTomahawk, Trident2
- コントロールプレーン実装のないミニマルな実装
  - コントロールプレーンはコントローラで実装が必要

2019年9月にオープンソース化！



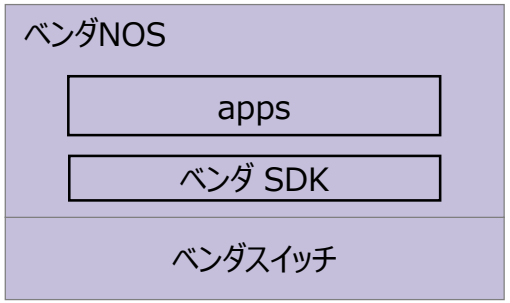
https://github.com/stratum/stratum

gRPCベースの新しいインターフェース！



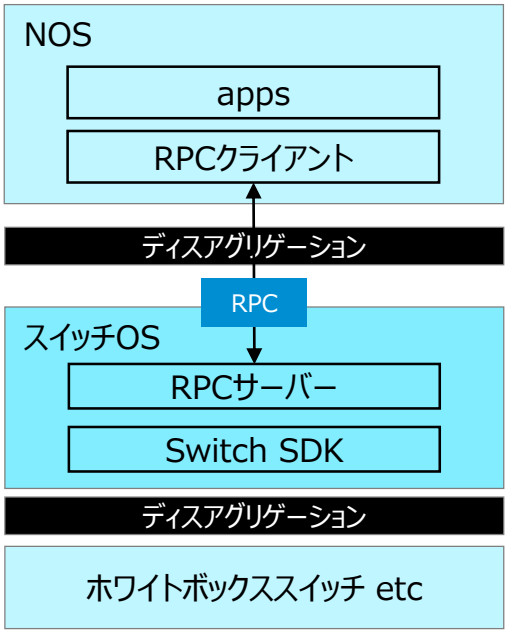
https://www.opennetworking.org/stratum/

スイッチの様々なかたち



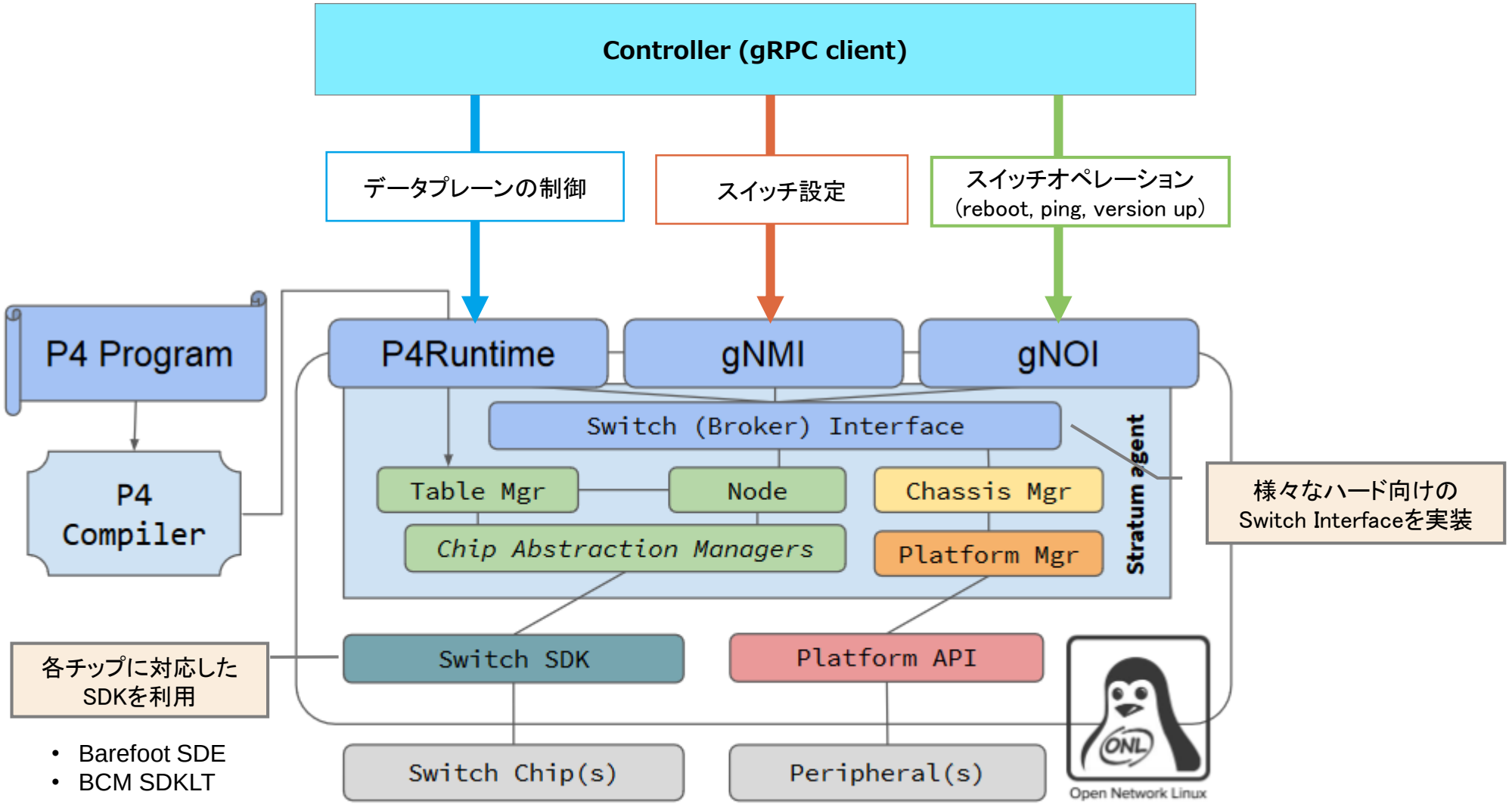
- SW/HWをベンダが提供

- NOSとHWを別のベンダが提供
- OCPにてHWの仕様を公開（ホワイトボックス）
- OSSのNOS （SONiC）



- データプレーンとコントロールプレーンの分離
- コントローラによる集中管理
- OSSのスイッチOS（Stratum）
- OSSのコントローラ（ONOS）

Stratum のアーキテクチャ



- Barefoot SDE
- BCM SDKLT

https://www.opennetworking.org/wp-content/uploads/2019/09/2.00pm-Brian-OConnor-Stratum.pdf

Stratumがサポートするハードウェア

	DELL	Delta	Edgecore networks	Inventec	QCT	STORDIS
Barefoot Tofino		AG9064v1	Wedge 100BF 32x Wedge 100BF 65x	D5254		BF6064X BF2556X
Broadcom Tomahawk	Z9100		AS7712-32X Cassini	D7032	T7032-IX1	
Broadcom Trident II			AS6712			

P4非対応チップも対応

https://github.com/stratum/stratum

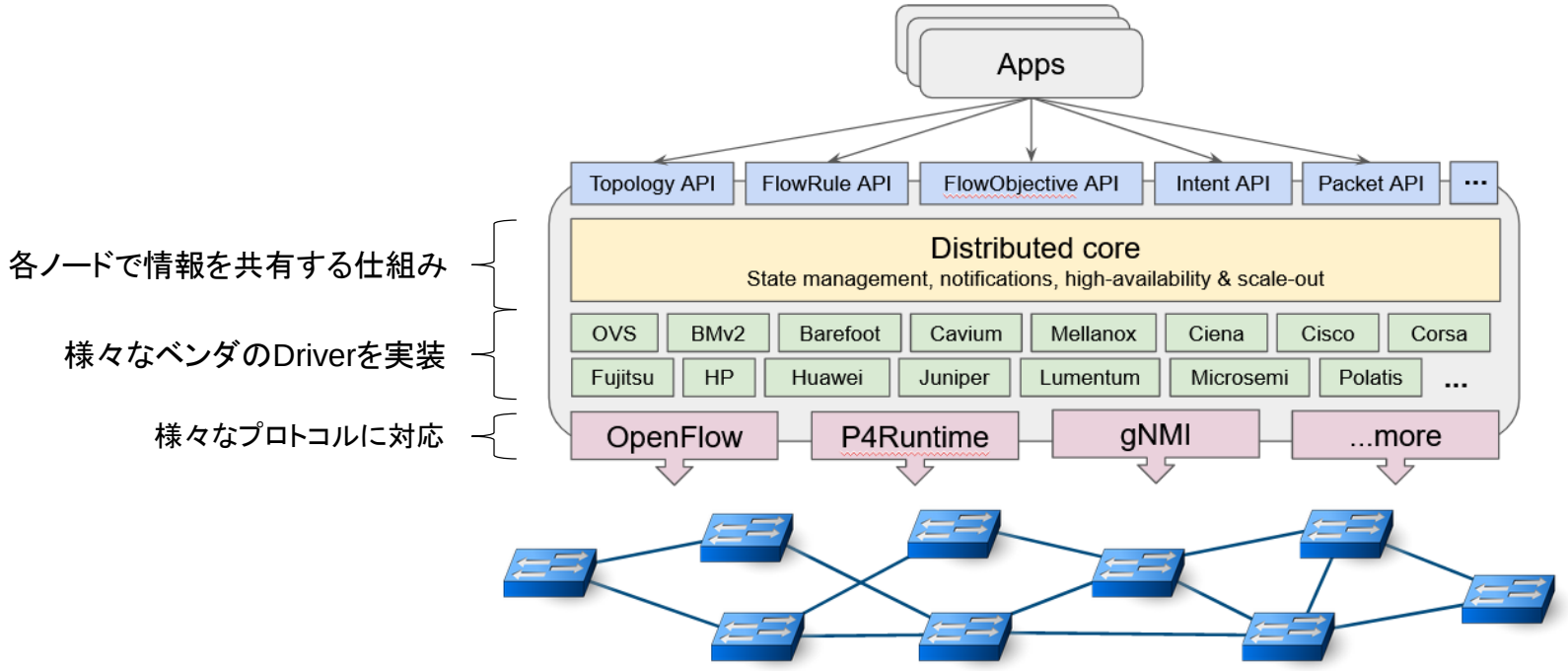
http://www.opennetworking.org/wp-content/uploads/2020/04/Stratum-Webinar-March-2020-Combined-Slide-Deck.pdf

目次

- 本日の発表内容
  - Stratumの概要
  - ONOSの概要
  - StratumとONOSを使った海外事例紹介
  - 簡単なサンプルアプリケーション（簡易NAT）を作成
    - データプレーン（Stratum + P4）実装
    - コントロールプレーン（ONOS）実装
    - デモ
  - まとめ

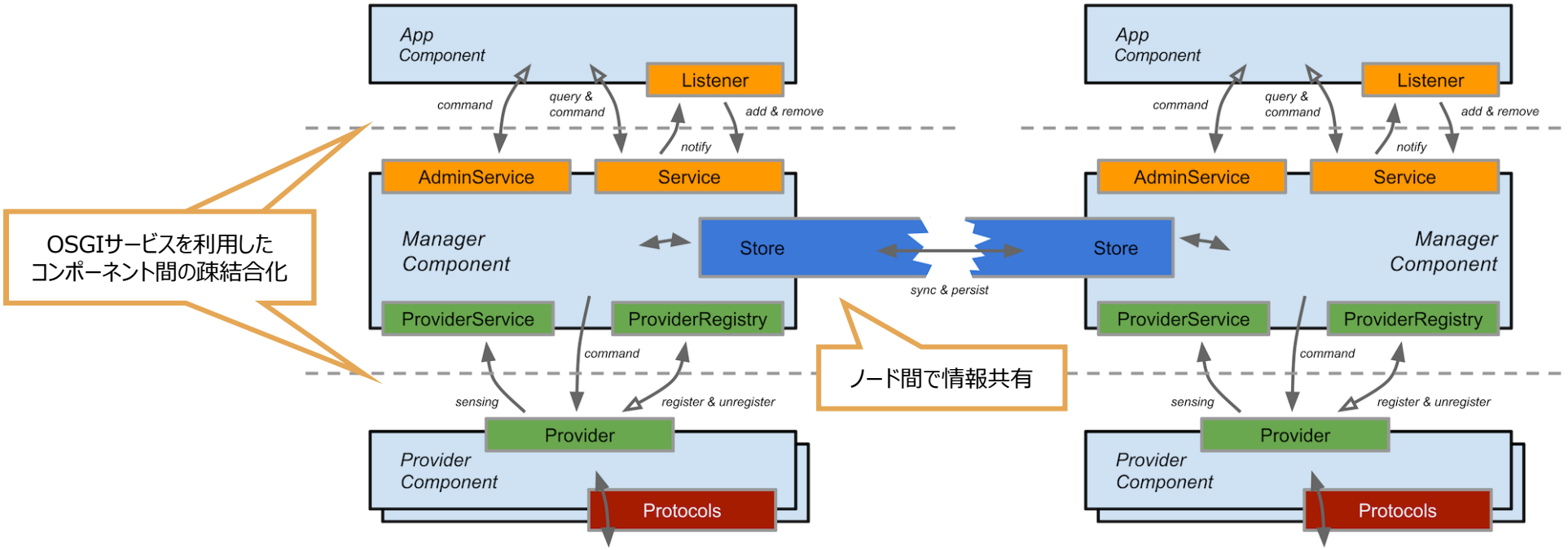
ONOSとは

- Open Network Operating System (ONOS)
- ONFが主導で開発を進めているOSSのSDNコントローラ
  - Apache License, Version 2.0
- 分散アーキテクチャによりスケールアウトや冗長化が可能
- OSGIのプラグイン技術を利用することで動的に機能追加が可能
  - OSGI: 機能の動的追加や実行を管理することが出来るJavaのフレームワーク



http://bit.ly/ngsdn-tutorial-slides

ONOSのコンポーネント間の関係

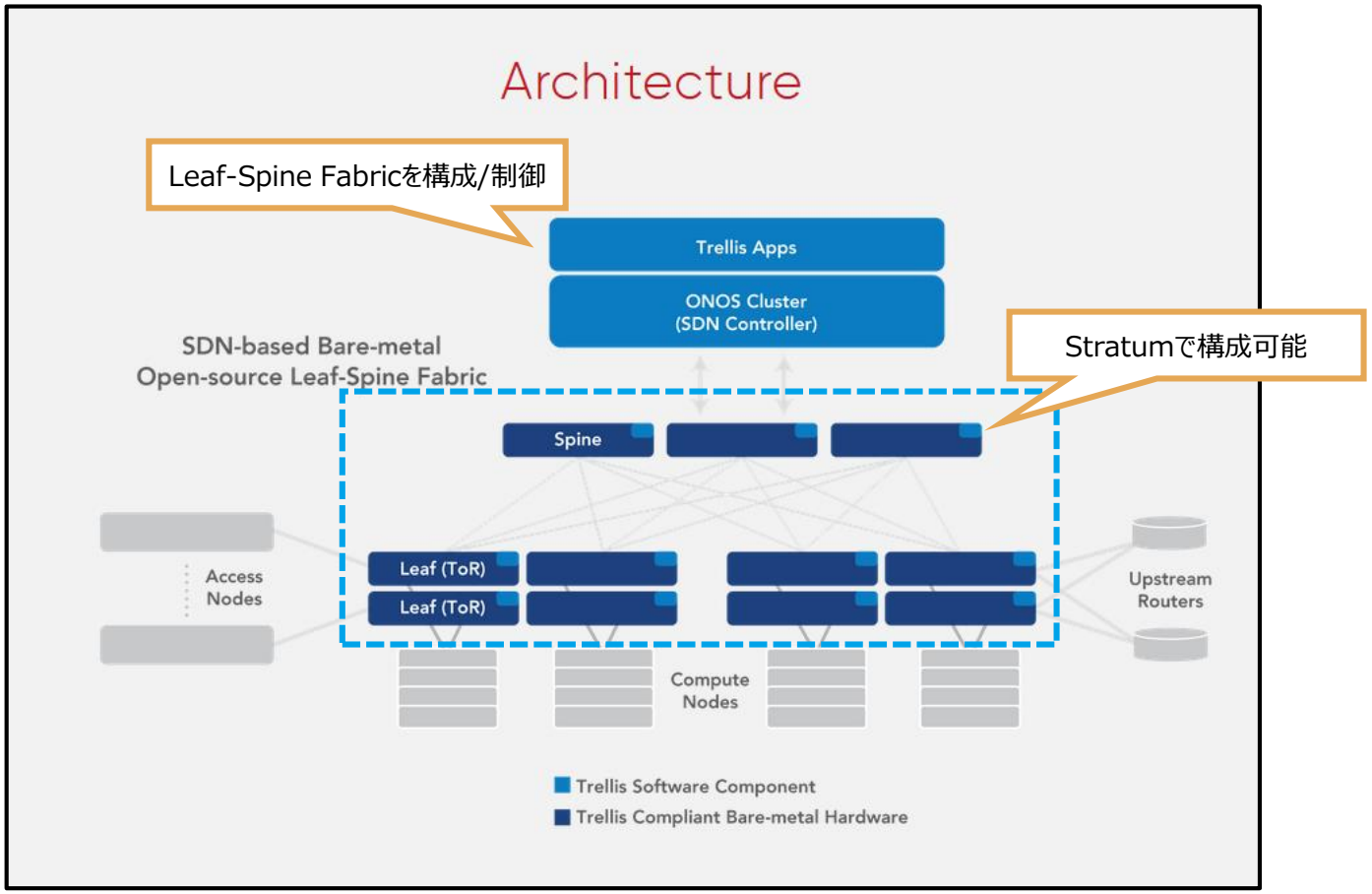


https://wiki.onosproject.org/display/ONOS/System+Components

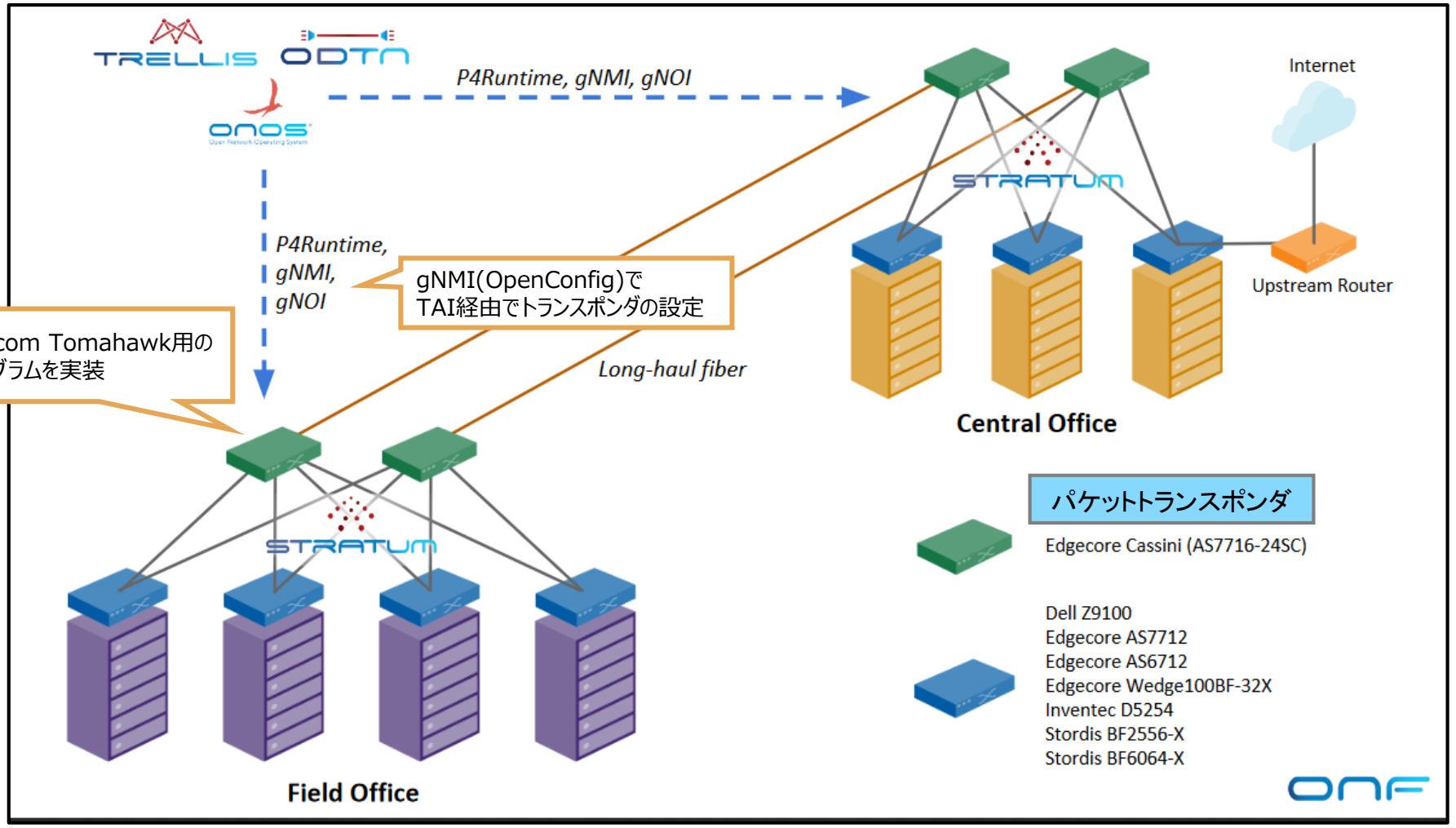
目次

- 本日の発表内容
  - Stratumの概要
  - ONOSの概要
  - StratumとONOSを使った海外事例紹介
  - 簡単なサンプルアプリケーション（簡易NAT）を作成
    - データプレーン（Stratum + P4）実装
    - コントロールプレーン（ONOS）実装
    - デモ
  - まとめ

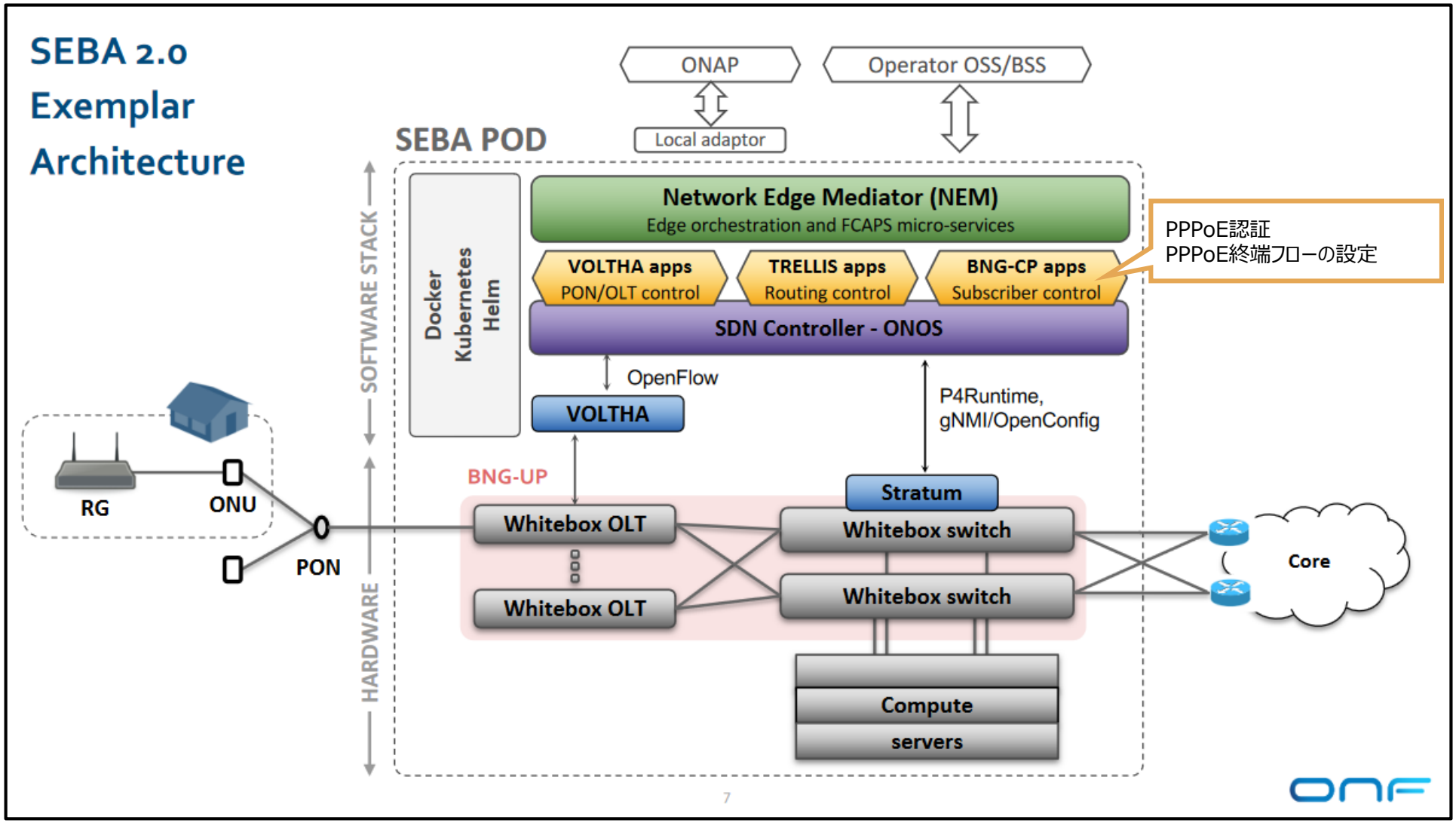
- Leaf-Spine Fabricを構成するためのコントロールプレーン実装
- データプレーンとしてStratumを使用可能



https://www.opennetworking.org/trellis/



http://www.opennetworking.org/wp-content/uploads/2020/04/Stratum-Webinar-March-2020-Combined-Slide-Deck.pdf



http://www.opennetworking.org/wp-content/uploads/2020/04/Stratum-Webinar-March-2020-Combined-Slide-Deck.pdf

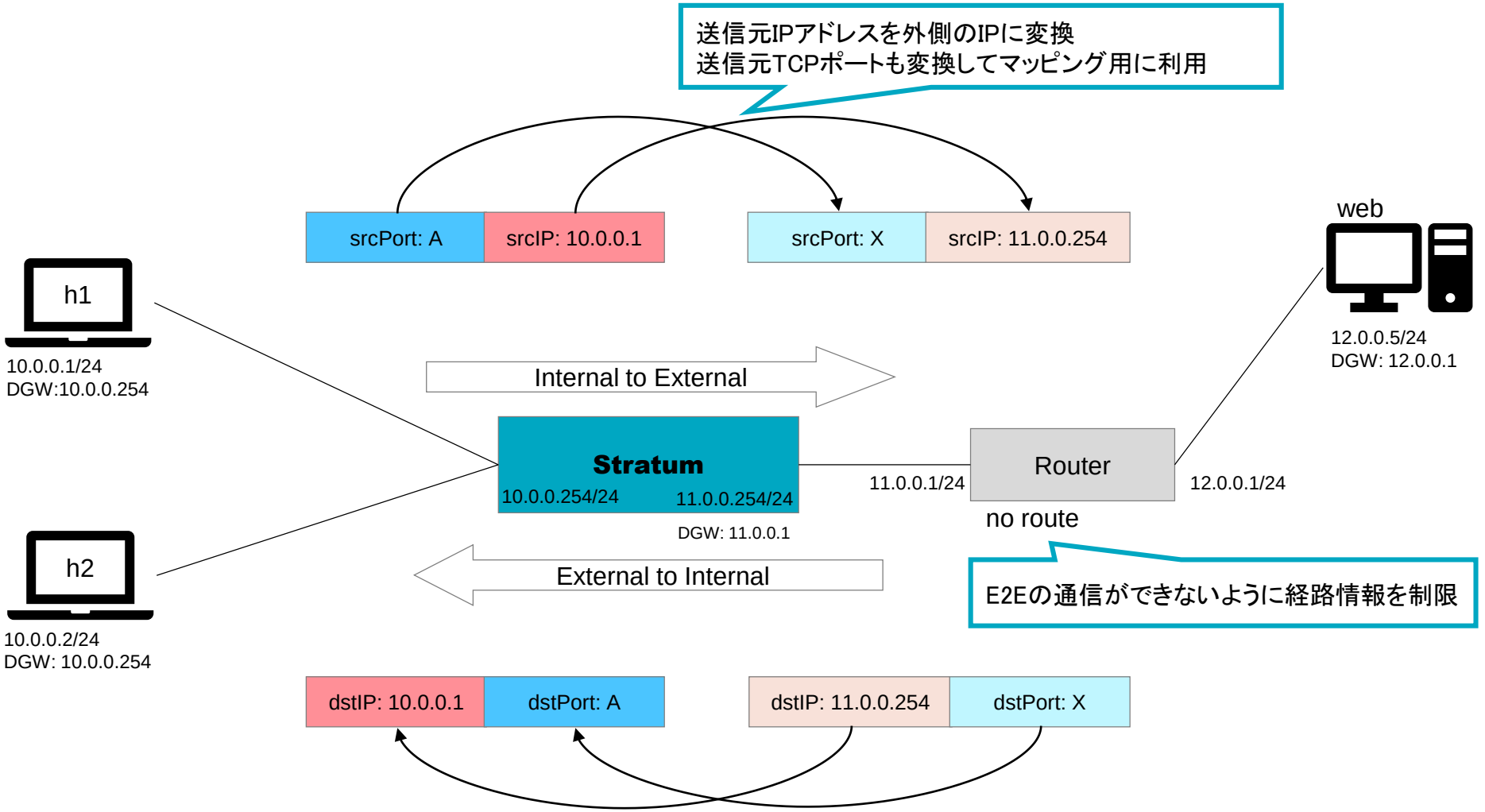
目次

- 本日の発表内容
  - Stratumの概要
  - ONOSの概要
  - StratumとONOSを使った海外事例紹介
  - 簡単なサンプルアプリケーション（簡易NAT）を作成
    - データプレーン（Stratum + P4）実装
    - コントロールプレーン（ONOS）実装
    - デモ
  - まとめ

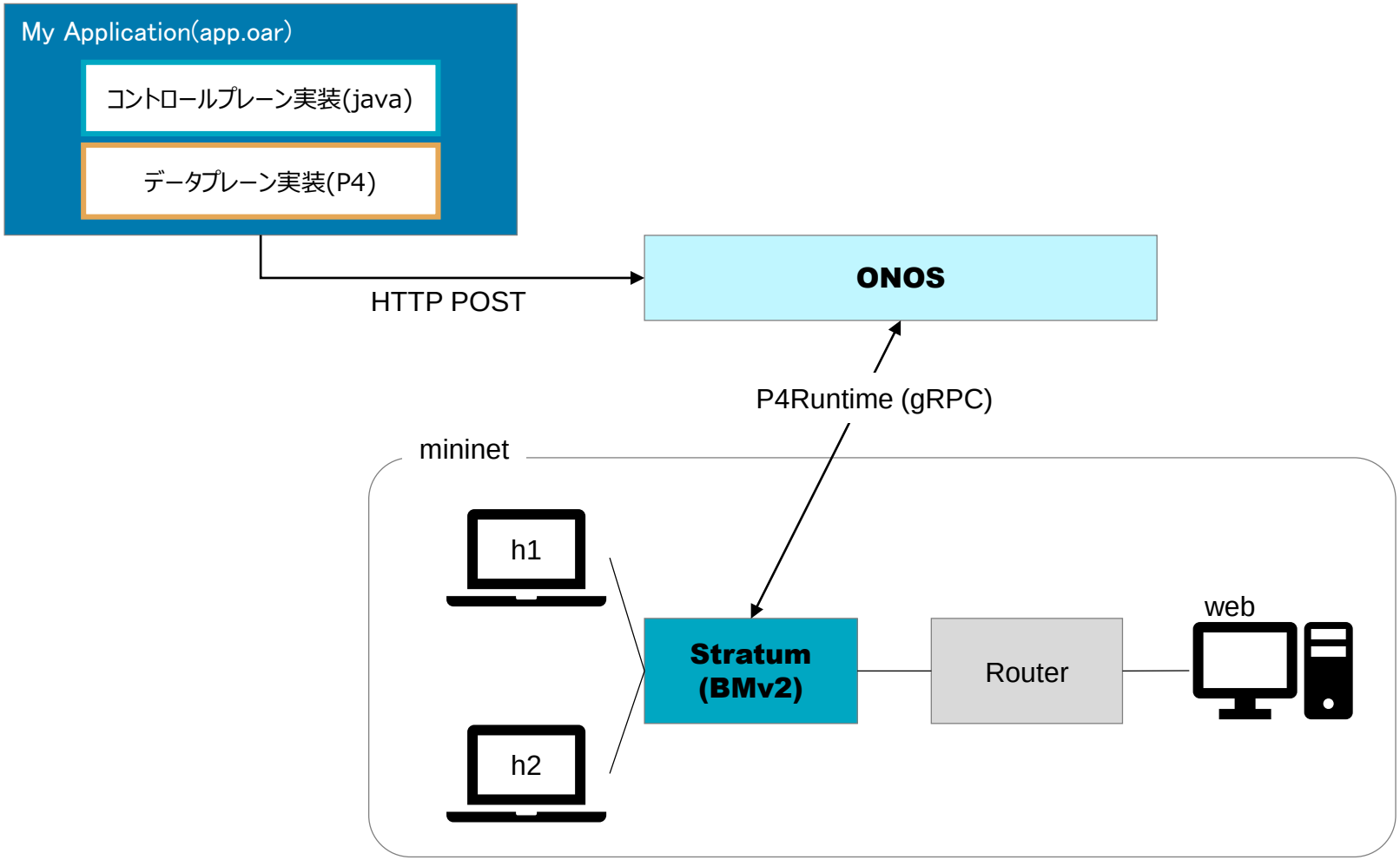
今回作成したソースコードなどの詳細は
後日テクニカルブログで公開予定！
https://www.apresiatac.jp/blog/

今回実現したい処理

- 簡易NATを実現させる
  - TCP通信のみ送信元IP/ポート変換を実施



試作全体構成


My Application(app.oar)
コントロールプレーン実装(java)
データプレーン実装(P4)
HTTP POST
ONOS
P4Runtime (gRPC)
mininet
h1
h2
Stratum
(BMv2)
Router
web

- ONOSとmininet環境はDocker (compose)で構築

docker-compose.yml

```
version: "3"

services:
  mininet:
    image: opennetworking/mn-stratum
    hostname: mininet
    container_name: mininet
    privileged: true
    tty: true
    stdin_open: true
    volumes:
      - ./tmp:/tmp
      - ./mininet:/mininet
    ports:
      - "50001:50001"
    entrypoint: "/mininet/demo.py"
  onos:
    image: onosproject/onos:2.2.2
    hostname: onos
    container_name: onos
    ports:
      - "8181:8181" # HTTP
      - "8101:8101" # SSH (CLI)
      - "5005:5005" # DEBUG
    volumes:
      - ./tmp/onos:/root/onos/apache-karaf-4.2.6/data/tmp
    environment:
      - ONOS_APPS=gui2,drivers.bmv2,hostprovider
      - JAVA_DEBUG_PORT=0.0.0.0:5005
    command:
      - server
      - debug
    links:
      - mininet
```

mininet

ONOS本体

mininet用のStratumスイッチ内蔵Dockerイメージ

Demo用のトポロジーを起動

インストールするアプリケーションはbmv2用のドライバ及びhostprovider

demo.pyの一部

- トポロジー

```
class TutorialTopo(Topo):

    def __init__(self, *args, **kwargs):
        Topo.__init__(self, *args, **kwargs)

        # gRPC port 50001
        s1 = self.addSwitch('switch1', cls=StratumBmv2Switch, cpuport=255)

        h1 = self.addHost('h1', ip='10.0.0.1/24', mac="00:00:00:00:00:10", defaultRoute="via 10.0.0.254")
        h2 = self.addHost('h2', ip='10.0.0.2/24', mac="00:00:00:00:00:20", defaultRoute="via 10.0.0.254")
        rt = self.addNode('rt', ip='11.0.0.1/24', mac="00:00:00:00:00:30", cls=LinuxRouter) # external router
        web = self.addHost('web', ip='12.0.0.5/24', mac="00:00:00:00:00:40", defaultRoute="via 12.0.0.1", cls=WebServer) #external host

        self.addLink(h1, s1)  # port 1
        self.addLink(h2, s1)  # port 2
        self.addLink(s1, rt,  params2={'ip': '11.0.0.1/24'})  # port 3
        self.addLink(web, rt, params2={'ip': '12.0.0.1/24'})
```

mininet用のStratumBmv2Switch

- その他

```
class LinuxRouter( Node ):
    "A Node with IP forwarding enabled."

    def config( self, **params ):
        super( LinuxRouter, self).config( **params )
        self.cmd( 'sysctl net.ipv4.ip_forward=1' )
        self.cmd( 'sysctl net.ipv4.conf.all.rp_filter=0' )
        self.cmd( 'sysctl net.ipv4.conf.default.rp_filter=0' )
        for intfName in self.intfNames():
          self.cmd( 'sysctl net.ipv4.conf.' + intfName  + '.rp_filter=0' )

    def terminate( self ):
# snip...
```

```
class WebServer(Host):

    def config(self, **kwargs):
        super(WebServer, self).config(**kwargs)
        self.cmd('python -m SimpleHTTPServer 80 &')

    def terminate(self):
        self.cmd('kill %python')
        super(WebServer, self).terminate()
```

目次

- 本日の発表内容
  - Stratumの概要
  - ONOSの概要
  - StratumとONOSを使った海外事例紹介
  - 簡単なサンプルアプリケーション（簡易NAT）を作成
    - データプレーン（Stratum + P4）実装
    - コントロールプレーン（ONOS）実装
    - デモ
  - まとめ

- P4: Programming Protocol-independent Packet Processors
  - パケットのパイプライン処理（パーサー、マッチアクションテーブル、デパーサー）を記述するためのプログラミング言語

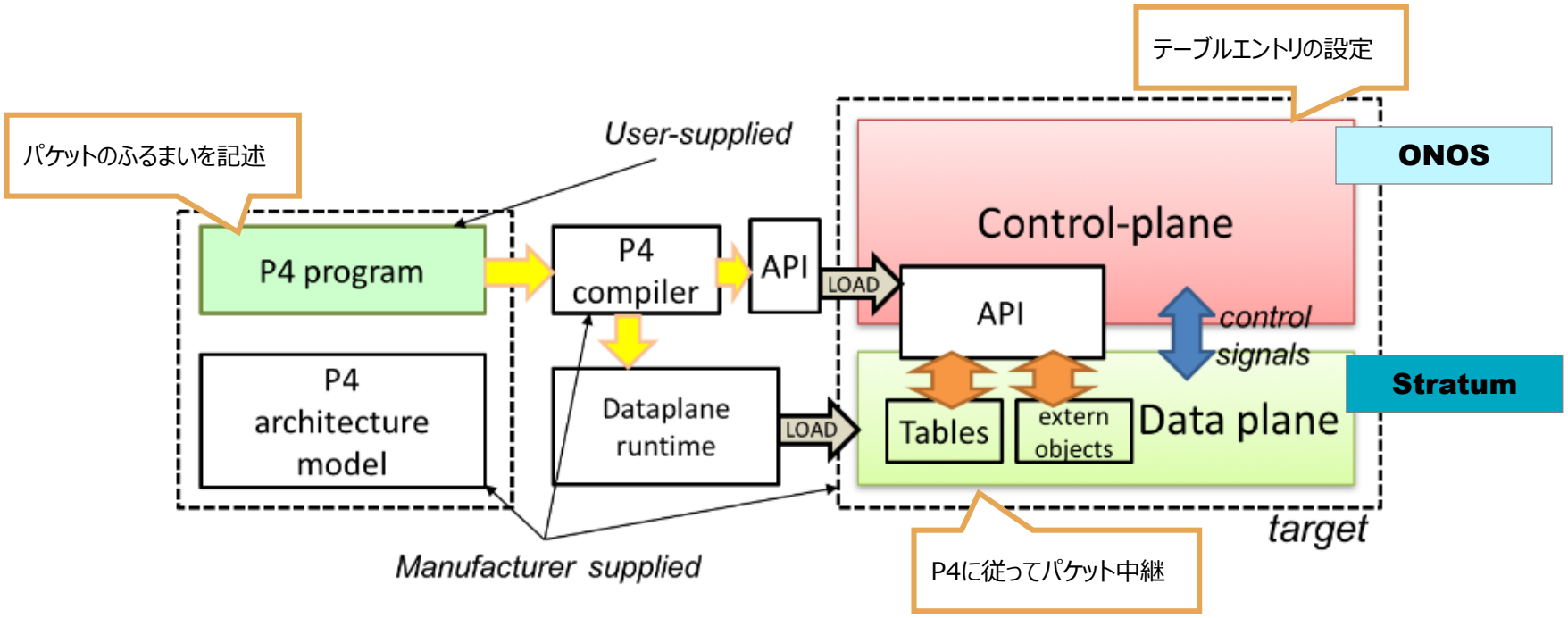
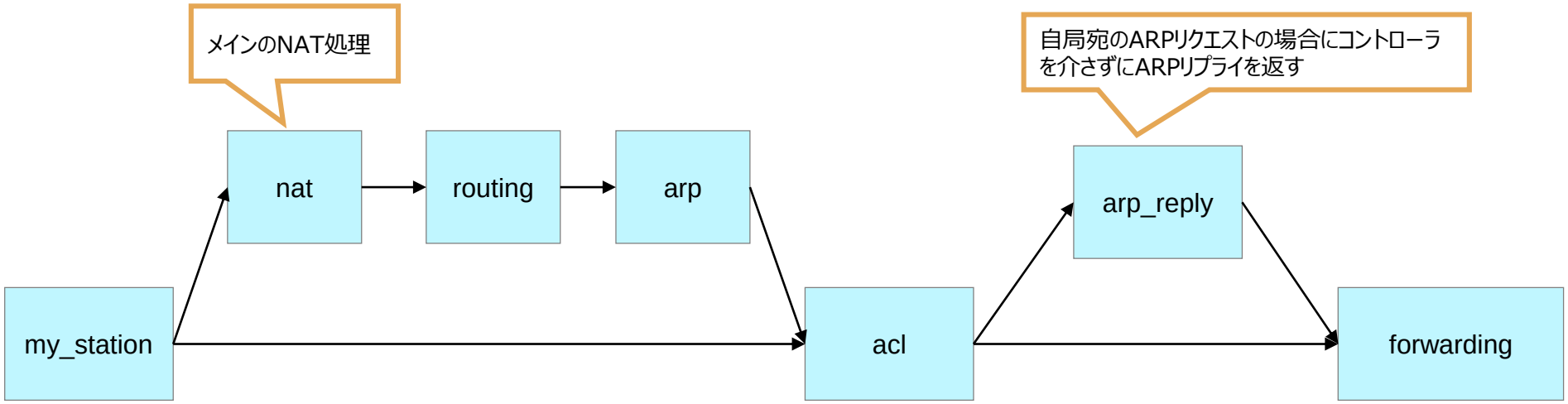


Figure 2. Programming a target with P4.

https://p4.org/p4-spec/docs/P4-16-v1.2.1.pdf

P4で実装が必要なマッチアクションテーブル

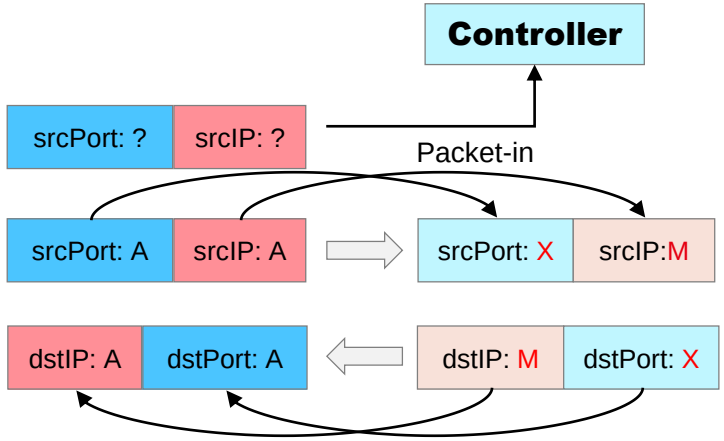
- **my_station:** 自局**MAC**宛の通信かどうかを判別するテーブル（アクションなし）
- **nat:** 送信元IPを外側のIPに、送信元TCPポートを任意の値に変換（その逆も）
- **routing:** 送信先IPアドレスに対するネクストホップの決定
- **arp:** ネクストホップのMACアドレスを送信先MACアドレスに変換
- **acl:** 制御パケットをコントローラーに送る処理
- **arp_reply:** ARP応答の処理
- **forwarding:** 宛先MACアドレスに対する出力ポートの決定



nat_table

- key
  - hdr.ipv4.src_addr　: ternary
  - hdr.ipv4.dst_addr　: ternary
  - hdr.tcp.src_port　: ternary
  - hdr.tcp.dst_port　: ternary
- Action
  - nat_miss_int_to_ext
    - 新規TCP通信を受信した場合にPacket-inするためのアクション
  - nat_hit_int_to_ext
    - 内→外への通信のために送信元IP/ポートを変換するアクション
  - nat_hit_ext_to_int
    - 外→内への通信のために送信先IP/ポートを元の送り主のものへ戻すアクション

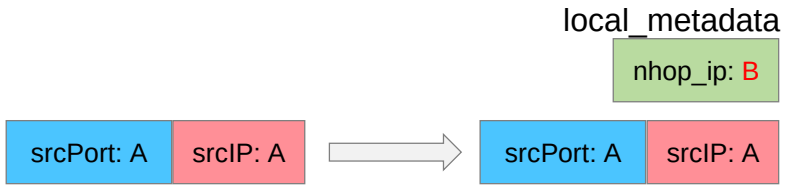
```
action nat_miss_int_to_ext() {
    // エントリーがない場合はCPUへコピー
    clone3(CloneType.I2E, CPU_CLONE_SESSION_ID, { standard_metadata });
}

action nat_hit_int_to_ext(ip_address_t srcAddr, l4_port_t srcPort) {
    hdr.ipv4.src_addr = srcAddr;
    hdr.tcp.src_port = srcPort;
}

action nat_hit_ext_to_int(ip_address_t dstAddr, l4_port_t dstPort) {
    hdr.ipv4.dst_addr = dstAddr;
    hdr.tcp.dst_port = dstPort;
}
```



※赤文字はAction引数から取得

routing_table

- key
  - hdr.ipv4.dst_addr: lpm
- Action
  - set_nhop
    - Next hopのIPアドレスを決定するアクション
    - 実際の送信先IPアドレスではないため、metadata領域に保持する
    - TTLの減算もここで実施

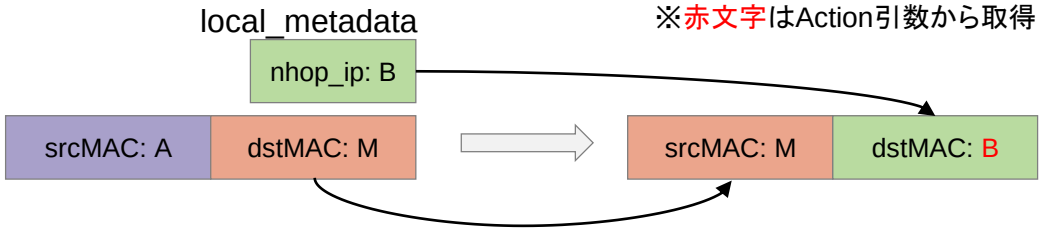


※赤文字はAction引数から取得

```
action set_nhop(ip_address_t nhop_ip) {
    local_metadata.nhop_ip = nhop_ip;
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}
```

arp_table

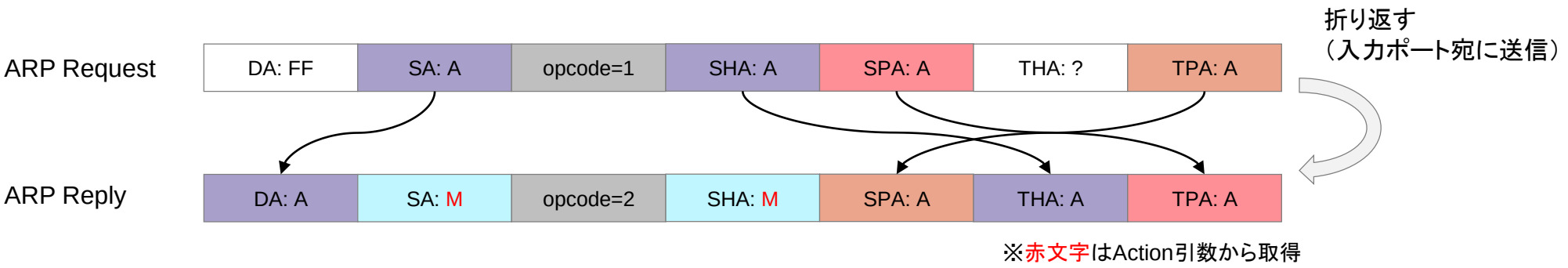
- key
  - local_metadata.nhop_ip: exact
- Action
  - change_mac
    - 送信元MACアドレスを自局のMACアドレスへ、
      送信先MACアドレスをNext hopのIPを持つ装置のMACアドレスへ変換




```
action change_mac(mac_t dmac) {
    hdr.ethernet.src_addr = hdr.ethernet.dst_addr;
    hdr.ethernet.dst_addr = dmac;
}
```

arp_reply_table

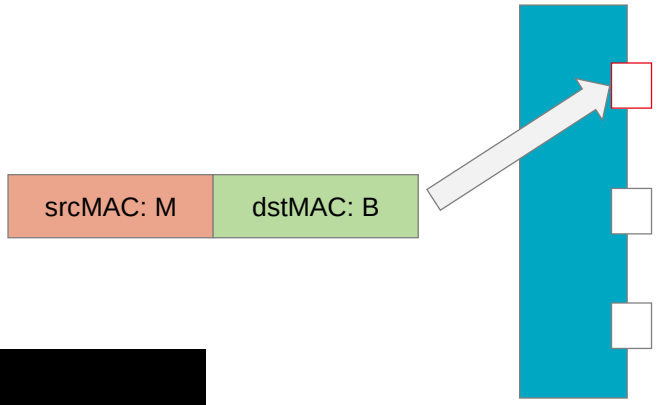
- key
  - hdr.arp.target_protocol_address: exact;
- Action
  - arp_req_to_rep
    - 自局宛のARPリクエストに対してリプライを送信するアクション




```
action arp_req_to_rep(mac_t target_mac) {
    hdr.ethernet.dst_addr = hdr.ethernet.src_addr;
    hdr.ethernet.src_addr = target_mac;
    hdr.arp.opecode = 2;
    ip_address_t target_ip_tmp = hdr.arp.target_protocol_address;
    hdr.arp.target_hardware_address = hdr.arp.sender_hardware_address;
    hdr.arp.target_protocol_address = hdr.arp.sender_protocol_address;
    hdr.arp.sender_hardware_address = target_mac;
    hdr.arp.sender_protocol_address = target_ip_tmp;
    standard_metadata.egress_spec = standard_metadata.ingress_port;
}
```

l2_exact_table

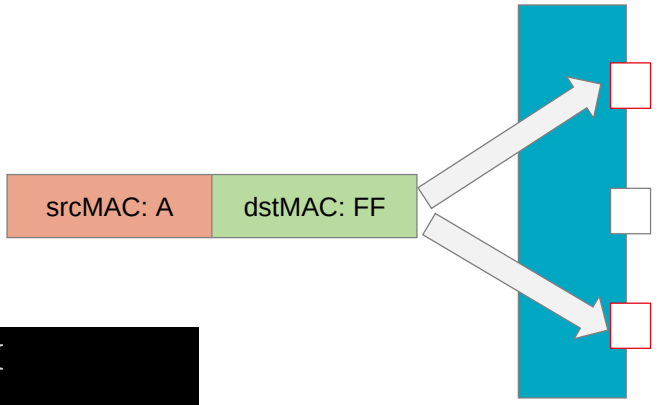
- key
  - hdr.ethernet.dst_addr: exact;
- Action
  - set_egress_port
    - 送信先MACアドレスに対して出力ポートを決定するアクション




```
action set_egress_port(port_t port_num) {
    standard_metadata.egress_spec = port_num;
}
```

l2_multicast_table

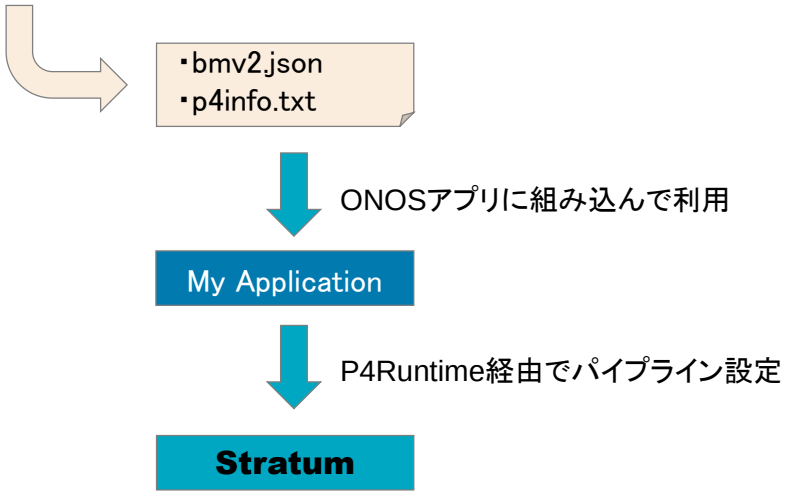
- key
  - hdr.ethernet.dst_addr: ternary;
  - standard_metadata.ingress_port: exact;
- Action
  - set_multicast_group
    - マルチキャストグループを付与するアクション




```
action set_multicast_group(mcast_group_id_t gid) {
    standard_metadata.mcast_grp = gid;
    local_metadata.is_multicast = true;
}
```

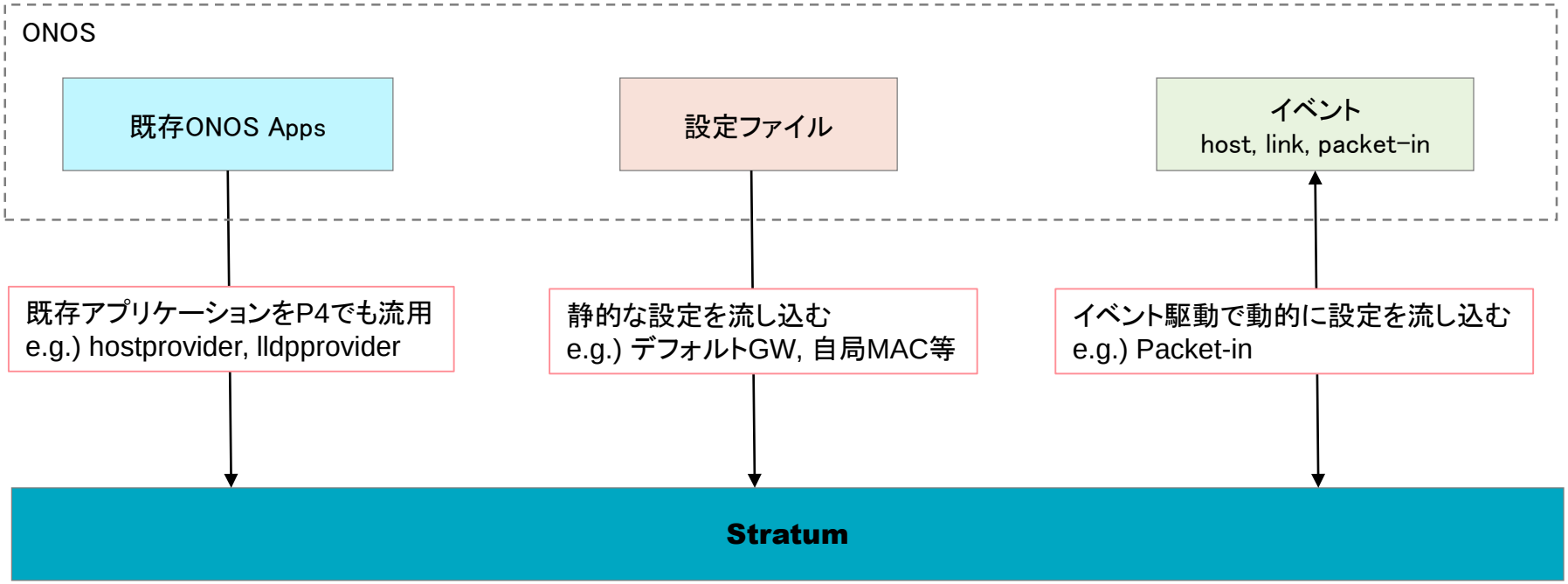
- P4コンパイラのDockerイメージ(opennetworking/p4c)を利用してコンパイル

```
$ docker run --rm -v $PWD:/workdir -w /workdir opennetworking/p4c:stable ¥
      p4c-bm2-ss --arch v1model -o p4src/build/bmv2.json ¥
      --p4runtime-files p4src/build/p4info.txt --Wdisable=unsupported ¥
      p4src/demo.p4
```



- 本日の発表内容
  - Stratumの概要
  - ONOSの概要
  - StratumとONOSを使った海外事例紹介
  - 簡単なサンプルアプリケーション（簡易NAT）を作成
    - データプレーン（Stratum + P4）実装
    - コントロールプレーン（ONOS）実装
    - デモ
  - まとめ

ONOSからP4制御するパターン



既存ONOSアプリケーションを流用する方法

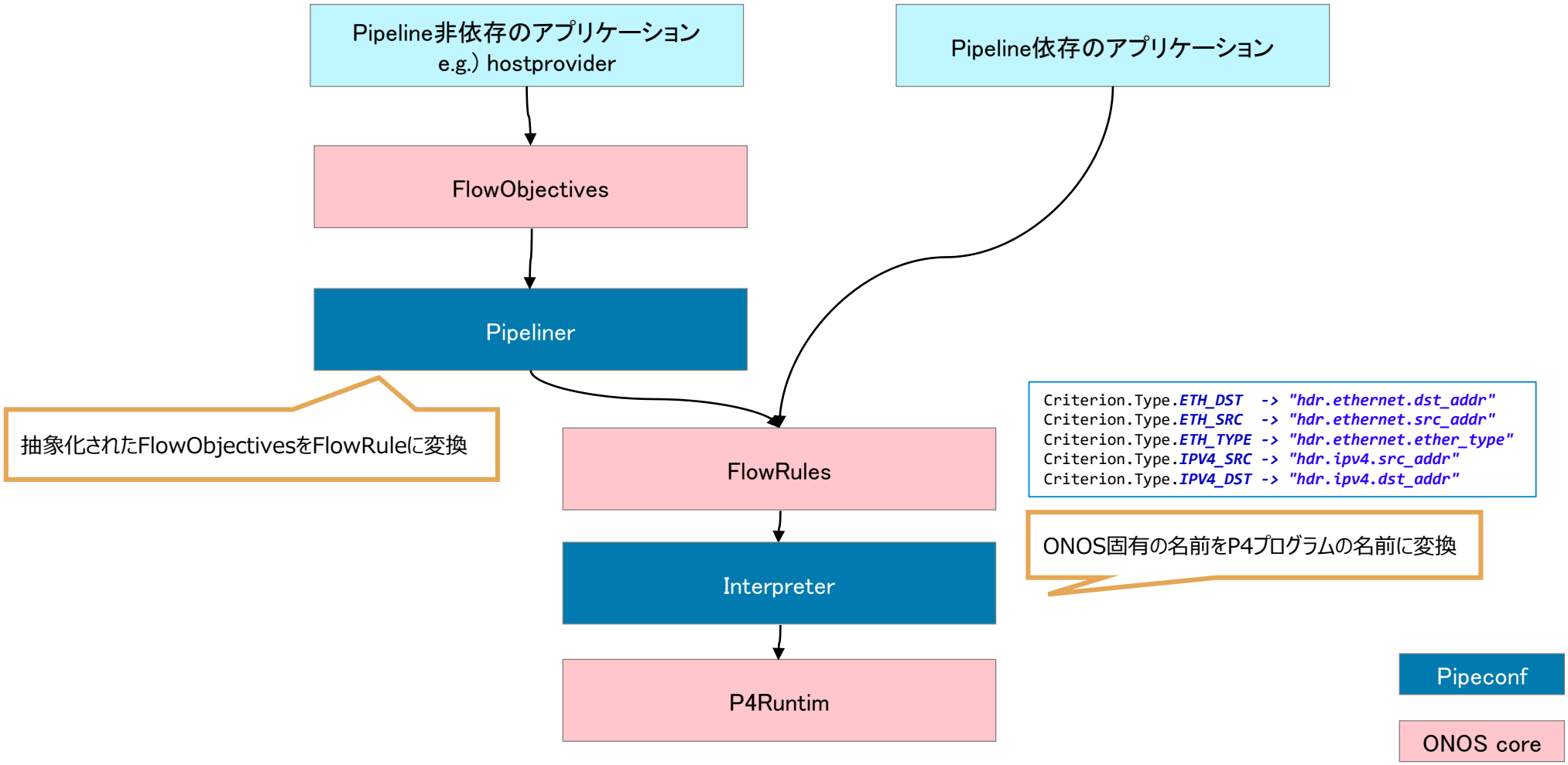
- 課題
  - OpenFlowを意識して作成された既存ONOSアプリケーションをP4対応させるには？
  - 既存アプリケーションのコード変更は避けたい

- 解決
  - ONOSのFlowRuleからP4のテーブルに変換するルールを挿入できるようにする
  - パイプラインの情報を1つのONOSアプリケーション(Pipeconf.oar)としてインストールできるようにする

pipeconf.oar

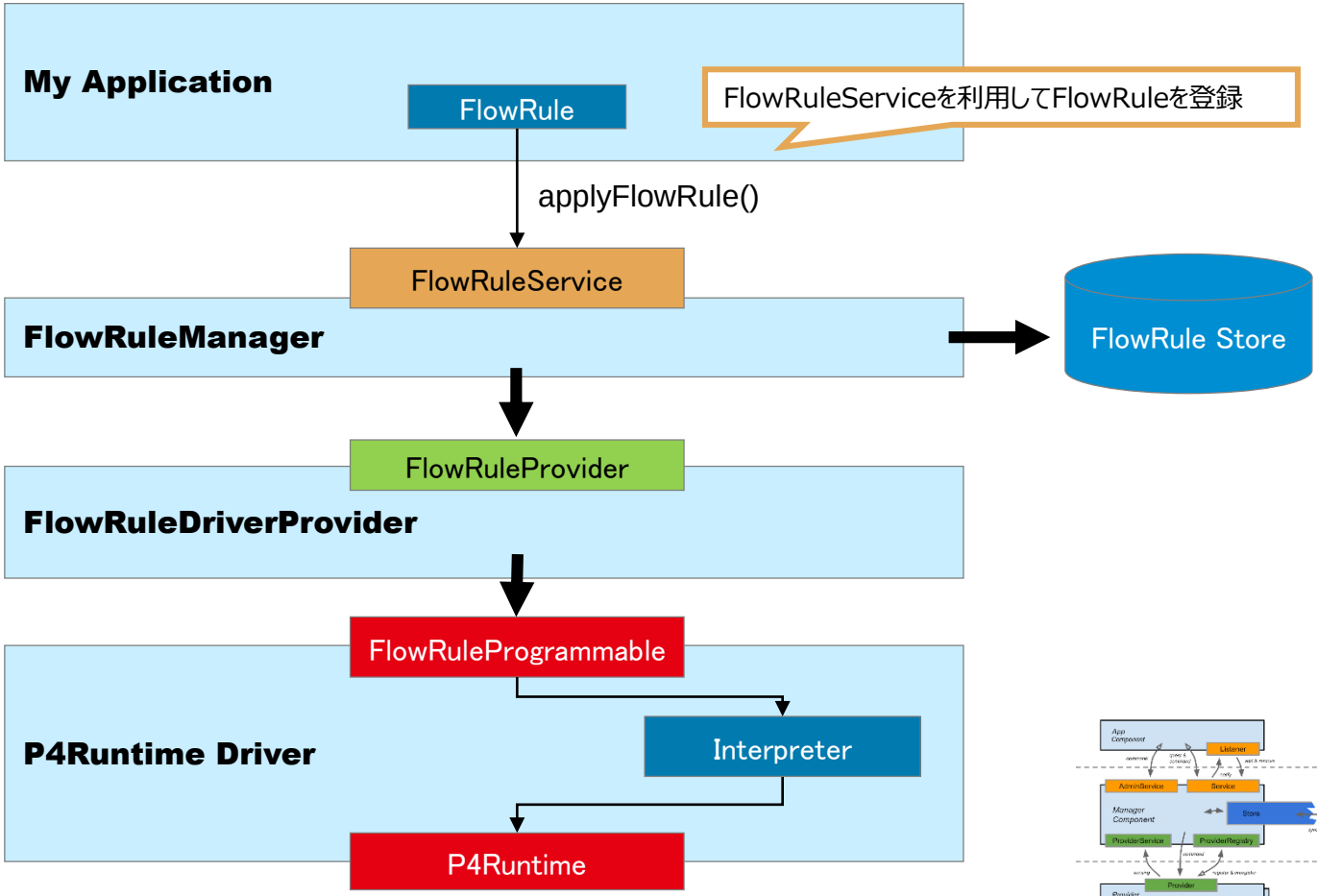
- Pipelineモデル定義
  - P4 プログラムをコンパイルして生成される (p4info.txt)
- P4スイッチ用のバイナリ
  - P4 プログラムをコンパイルして生成される (e.g. bmv2.json, Tofino.bin)
- ONOS FlowRule -> P4 tableへの変換ルール（Java program）

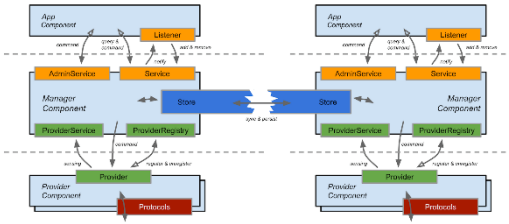
既存ONOSアプリケーションを流用する方法



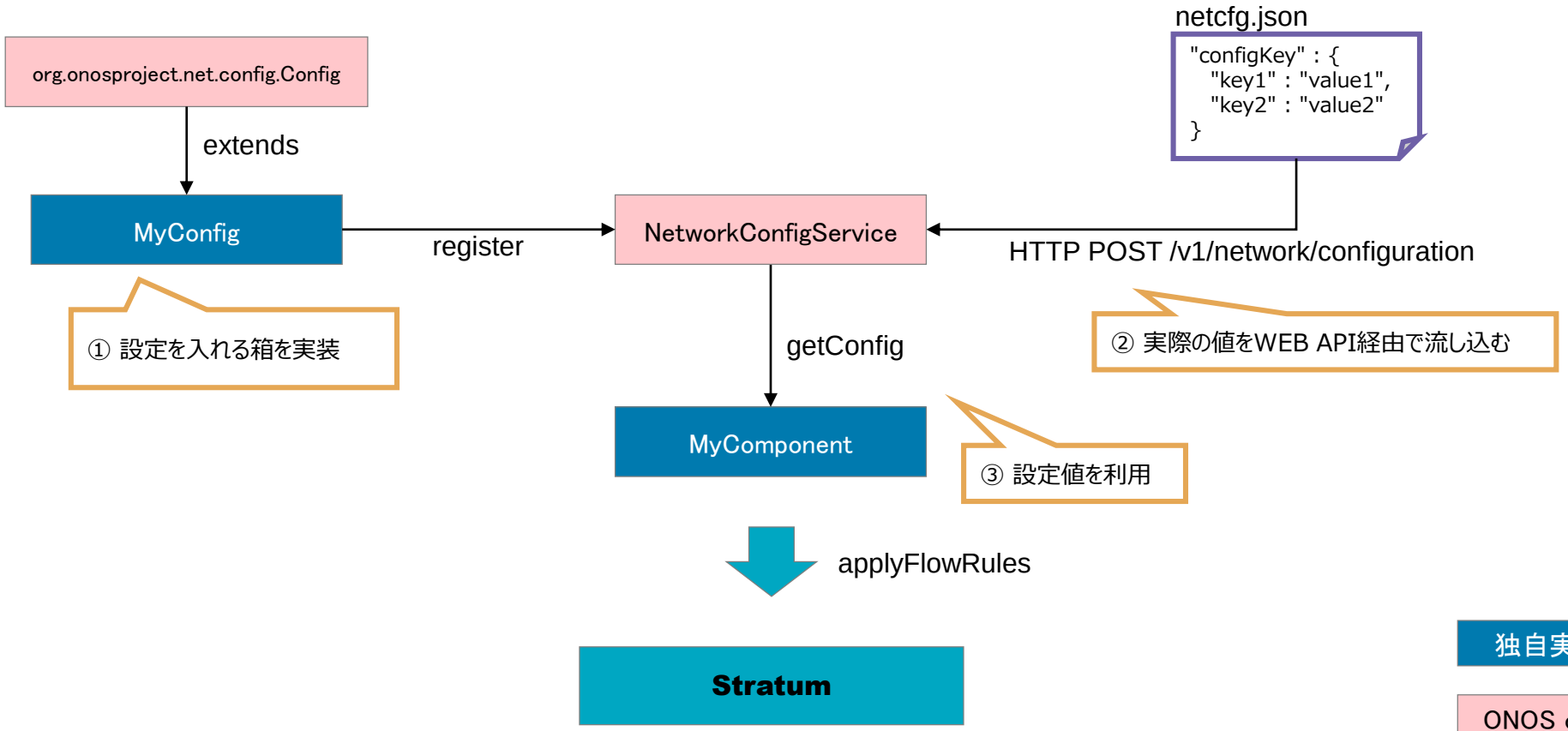
PipeconfをP4プログラムに合わせて実装することで既存アプリケーションのコードを変更せずに流用可能

FlowRule登録時の処理の流れ





設定ファイルから静的な設定を流し込む方法



設定ファイルから静的な設定を流し込む方法

設定を入れる箱(オブジェクト)を実装

```
public class NatConfig extends Config<DeviceId> {
    public static final String CONFIG_KEY = "natConfig";
    private static final String SRC_RANGE = "srcRange";
    private static final String GLOBAL_IP = "globalIp";

    public Ip4Prefix srcRange() {
        String srcRange = get(SRC_RANGE, null);
        return srcRange != null ? Ip4Prefix.valueOf(srcRange) : null;
    }
//...snip...
```

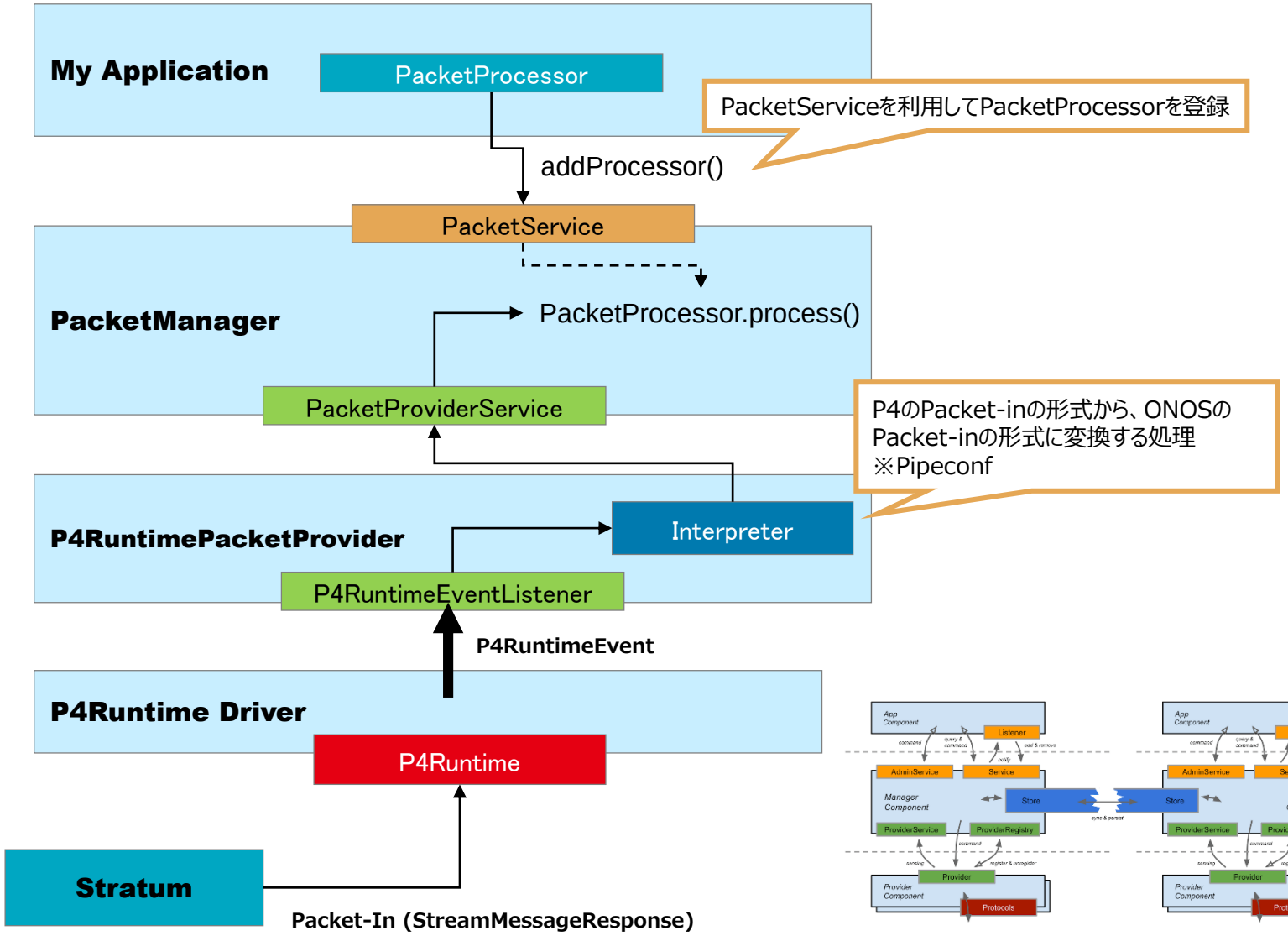
configRegistryを介して登録

```
 @Activate
    protected void activate() {
       configRegistry.registerConfigFactory(natConfigFactory);
//...snip...
```

NetworkConfigServiceから呼び出す

```
NatConfig config = networkConfigService.getConfig(deviceId, NatConfig.class);
```

イベント（Packet-In）契機で設定する方法



PacketProcessorの設定方法

PacketProcessorの実装

```
 private class PacketInProcessor implements PacketProcessor {
        @Override
        public void process(PacketContext context) {

            InboundPacket pkt = context.inPacket();
            Ethernet ethPkt = pkt.parsed();
// ...snip...
```

定義およびOSGIからPacketServiceの実装を呼び出し

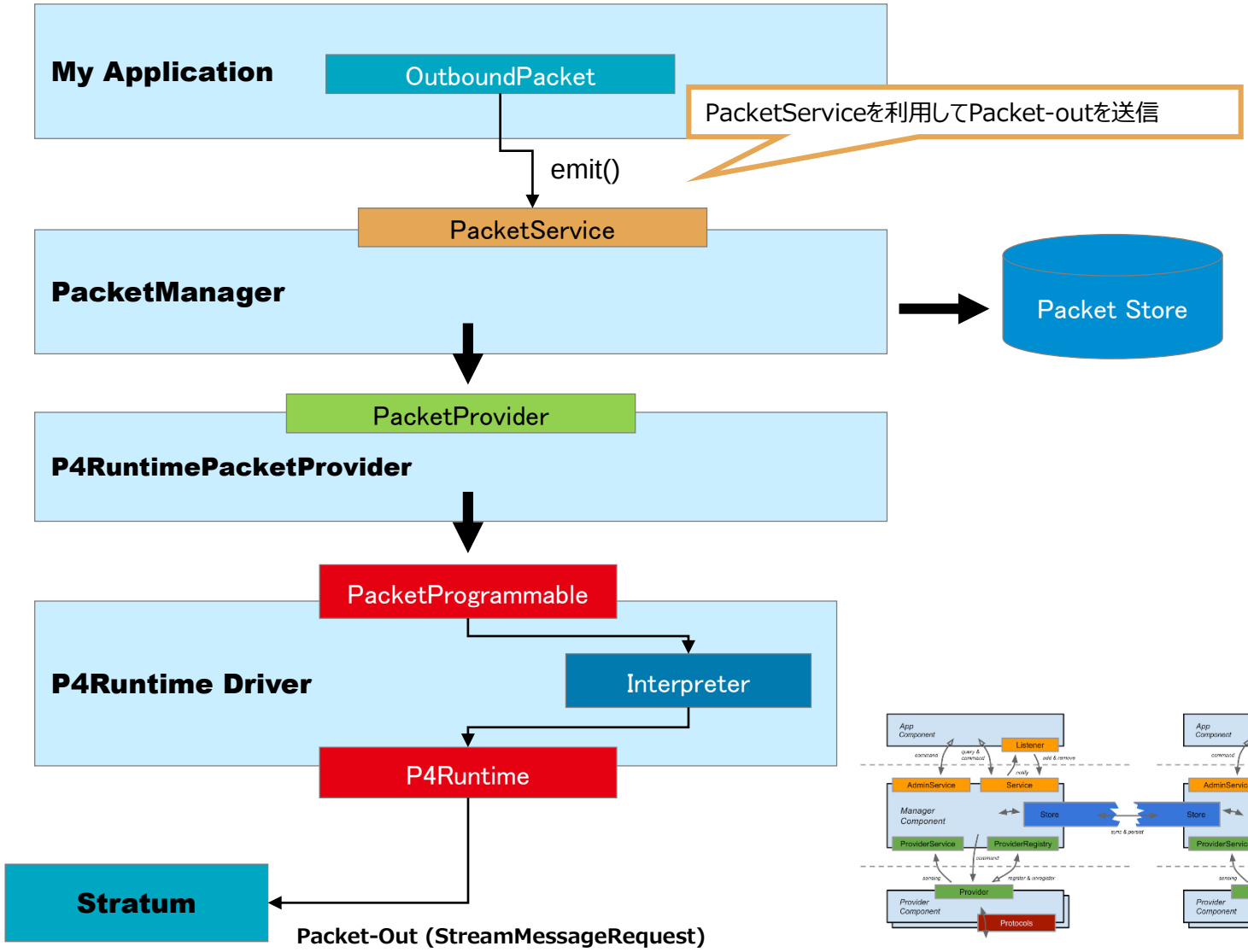
```
private final PacketInProcessor packetInProcessor = new PacketInProcessor();


@Reference(cardinality = ReferenceCardinality.MANDATORY)
private PacketService packetService;
//...snip...
```

@Referenceアノテーションで OSGIサービスに登録してある実装を呼び出し可能

初期化時にprocessorを登録

```
 @Activate
    protected void activate() {
        packetService.addProcessor(packetInProcessor, PacketProcessor.director(6));
// ...snip...
```

【参考】Packet-Out時の内部動作

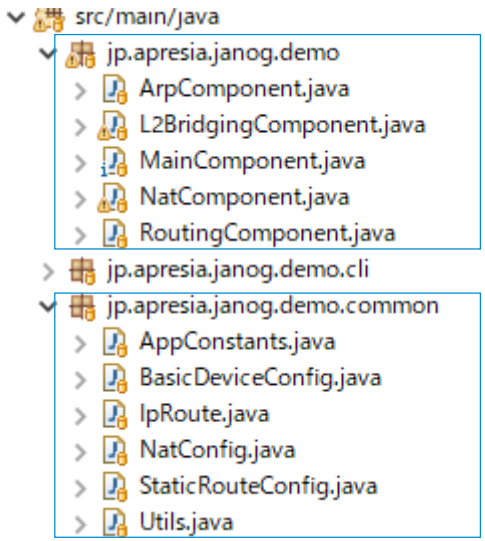


ONOSアプリケーションの雛形

- Mavenのカタログで空のプロジェクトを生成可能だが・・・

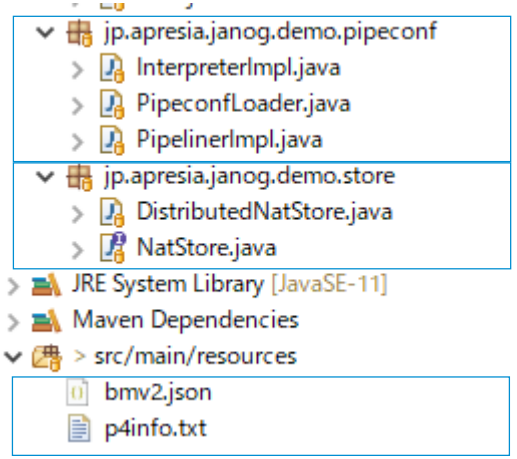
```
> mvn archetype:generate -DarchetypeCatalog=local,remote -DarchetypeGroupId=org.onosproject -DarchetypeArtifactId=onos-bundle-archetype  -DarchetypeVersion=2.2.2
```

- とりあえず試したい場合はデモ用プロジェクトを改造するのが早い
  - https://github.com/opennetworkinglab/ngsdn-tutorial
- 今回作成したプロジェクトのツリー



メインのコンポーネント群

静的な設定用のモデル
及びユーティリティ



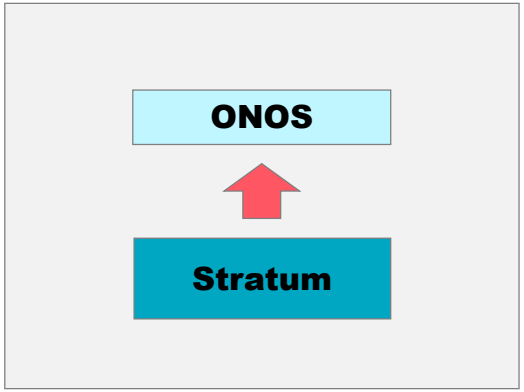
Pipeconf

Store

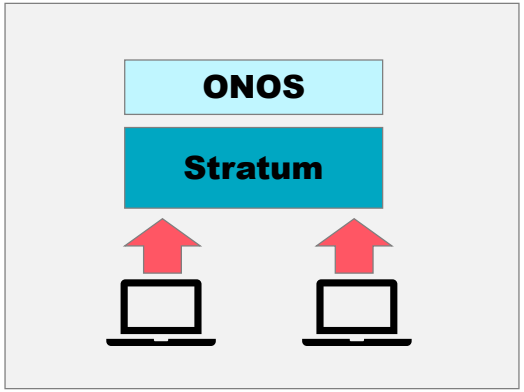
コンパイル済のP4プログラム

代表的なFlow設定の契機

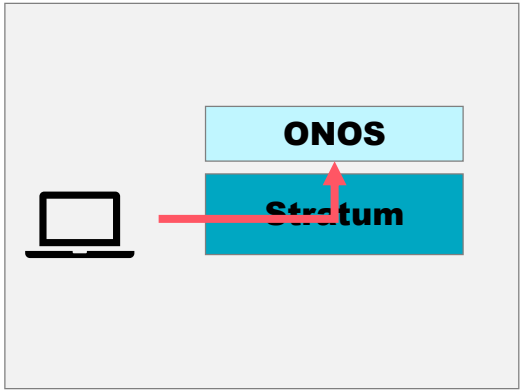
①ONOSにデバイス登録したとき



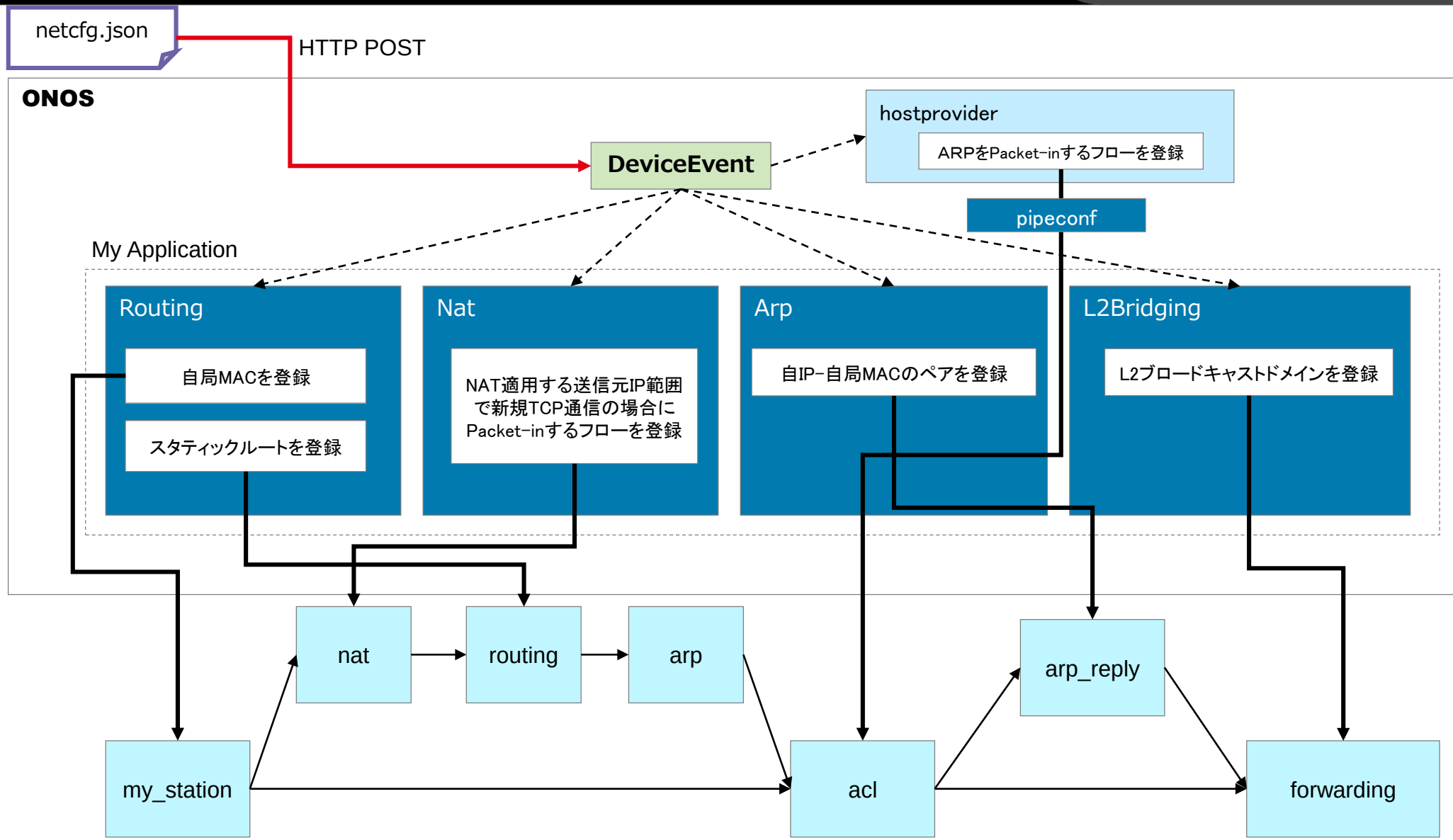
②ONOSにホスト登録したとき



③Packet-Inイベント発生時



コントローラーでの処理（デバイス登録時）



NAT範囲のFlow設定の例

```
    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private DeviceService deviceService;


    @Activate
    protected void activate() {
       deviceService.addListener(deviceListener);
       //...snip...
    }
```

```
class InternalDeviceListener implements DeviceListener {
        @Override
        public void event(DeviceEvent event) {
            mainComponent.getExecutorService().execute(() -> {
                DeviceId deviceId = event.subject().id();
                log.info("{} event! device id={}", event.type(), deviceId);
                setUpNatSrc(deviceId);
            });
        }
    }
```

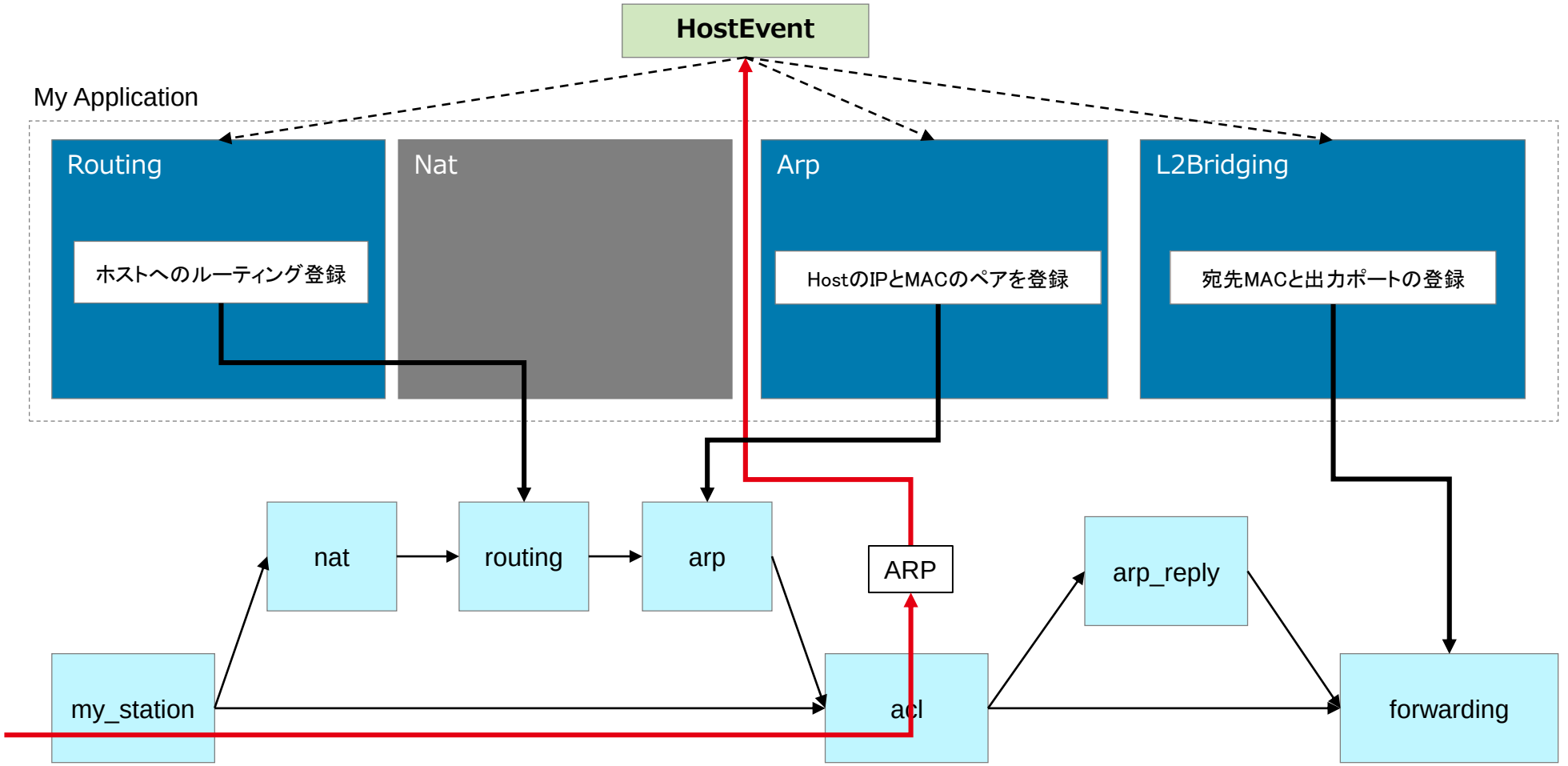
DeviceEvent契機にsetUpNatSrc()が呼び出される

```
private void setUpNatSrc(DeviceId deviceId) {
   log.info("Adding nat src range");

   final String tableId = "ingress.nat_control.nat";

   final Ip4Prefix srcRange = getSrcIpRange(deviceId);

   final PiCriterion match = PiCriterion.builder()
                             .matchTernary(PiMatchFieldId.of("hdr.ipv4.src_addr"), srcRange.address().toOctets(),
                                           Ip4Address.makeMaskPrefix(srcRange.prefixLength()).toOctets())
                             .build();
   final PiTableAction action = PiAction.builder()
                             .withId(PiActionId.of("ingress.nat_control.nat_miss_int_to_ext"))
                             .build();

   final FlowRule natSrcRule = Utils.buildFlowRule(deviceId, appId, tableId, match, action, 1);
   flowRuleService.applyFlowRules(natSrcRule);
}
```

設定ファイルから値を読みだす

flowRuleServiceのapplyFlowRulesでFlow登録

コントローラーでの処理（ホスト登録時）



ホストへのルーティングFlow設定の例

```
    @Reference(cardinality = ReferenceCardinality.MANDATORY)
    private HostService hostService;

    @Activate
    protected void activate() {
        hostService.addListener(hostListener);
    //...snip...
    }
```

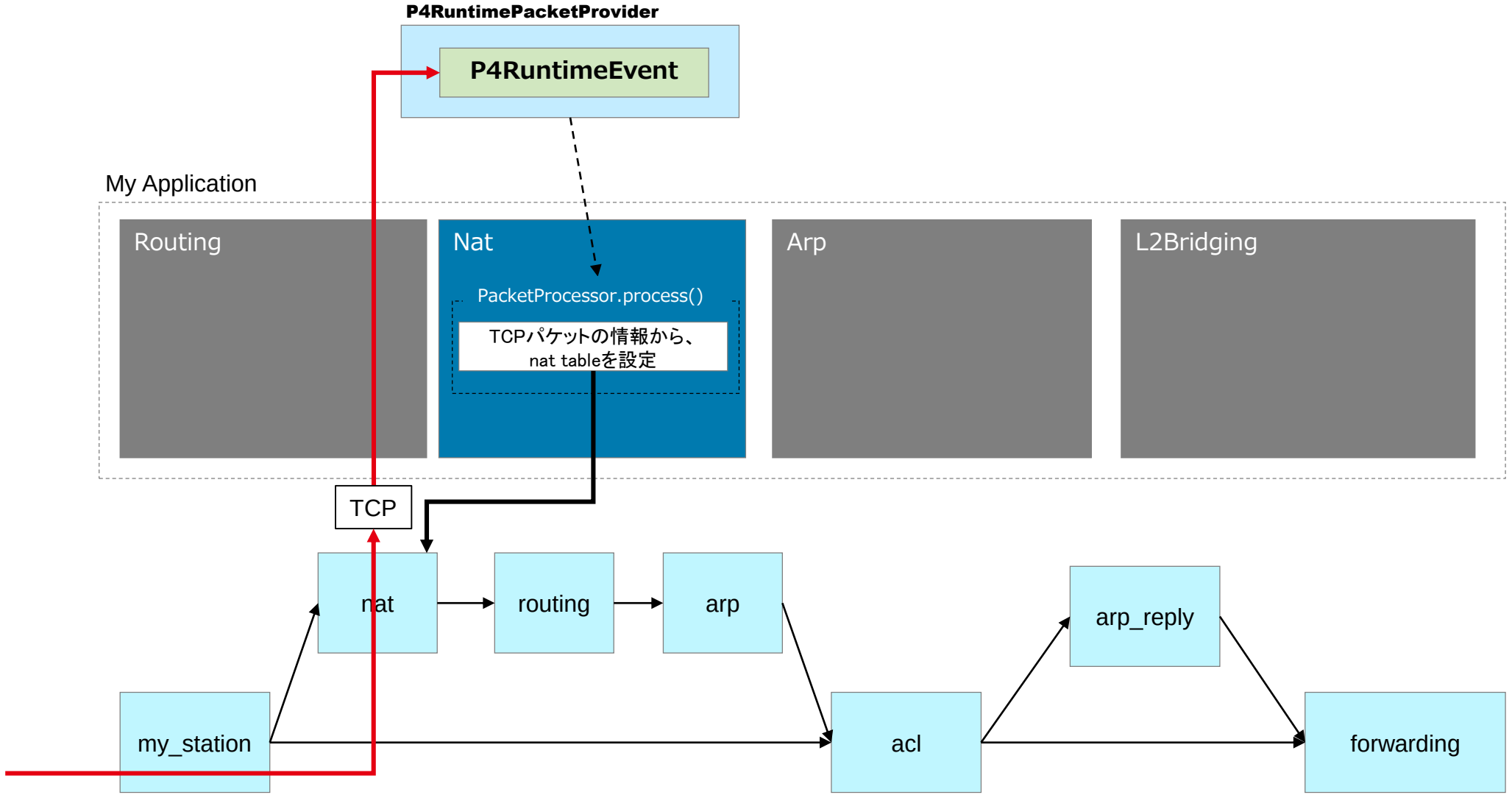
```
class InternalHostListener implements HostListener {

        @Override
        public void event(HostEvent event) {

            final Host host = event.subject();
            final DeviceId deviceId = host.location().deviceId();
            final PortNumber port = host.location().port();

            mainComponent.getExecutorService().execute(() -> {
                log.info("{} event! host={}, deviceId={}, port={}",
                         event.type(), host.id(), deviceId, port);
                setUpDirectConnectedHostRoute(deviceId, host);
            });
        }
    }
```

HostEvent契機に
setUpDirectConnectedHostRoute()が呼び出される

```
private void setUpDirectConnectedHostRoute(DeviceId deviceId, Host host) {
    final String tableId = "ingress.nat_control.routing";
    for(IpAddress hostIp: host.ipAddresses()) {
        final PiCriterion match = PiCriterion.builder()
                                  .matchLpm(PiMatchFieldId.of("hdr.ipv4.dst_addr"), hostIp.toOctets(), 32)
                                  .build();

        final PiTableAction action = PiAction.builder()
                                     .withId(PiActionId.of("ingress.nat_control.set_nhop"))
                                     .withParameter(new PiActionParam(
                                                    PiActionParamId.of("nhop_ip"),
                                                    hostIp.toOctets()))
                                     .build();

        final FlowRule hostRouteRule = Utils.buildFlowRule(deviceId, appId, tableId, match, action);
        flowRuleService.applyFlowRules(hostRouteRule);
    }
}
```

HostEventのEvent情報から必要な情報を取り出して処理

コントローラーでの処理（新規TCP通信）



NATのFlow設定の例

```
private class PacketInProcessor implements PacketProcessor {
        @Override
        public void process(PacketContext context) {

            InboundPacket pkt = context.inPacket();
            Ethernet ethPkt = pkt.parsed();

            if(ethPkt.getEtherType() != Ethernet.TYPE_IPV4) {
                return;
            }
            IPv4 ipv4Packet = (IPv4) ethPkt.getPayload();
            if (ipv4Packet.getProtocol() != IPv4.PROTOCOL_TCP) {
                return;
            }
            TCP tcpPacket = (TCP) ipv4Packet.getPayload();
            DeviceId deviceId = pkt.receivedFrom().deviceId();
            Integer num = natStore.newPort();

            final String tableId = "ingress.nat_control.nat";

            PiCriterion matchIntToExt = PiCriterion.builder()
                .matchTernary(PiMatchFieldId.of("hdr.ipv4.src_addr"),
                              IpAddress.valueOf(ipv4Packet.getSourceAddress()).toOctets(), IpAddress.valueOf("255.255.255.255").toOctets())
                .matchTernary(PiMatchFieldId.of("hdr.tcp.src_port"), tcpPacket.getSourcePort(), Integer.parseInt("FFFF", 16))
                .build();

            PiAction actionIntToExt = PiAction.builder()
                .withId(PiActionId.of("ingress.nat_control.nat_hit_int_to_ext"))
                .withParameter(new PiActionParam(PiActionParamId.of("srcAddr"), getGlobalIp(deviceId).toOctets()))
                .withParameter(new PiActionParam(PiActionParamId.of("srcPort"), num)).build();

            FlowRule ruleIntToExt = Utils.buildFlowRuleWithIdleTimeout(deviceId, appId, tableId, matchIntToExt, actionIntToExt, AppConstants.IDLE_TIMEOUT);

            PiCriterion matchExtToInt = PiCriterion.builder()
                .matchTernary(PiMatchFieldId.of("hdr.ipv4.dst_addr"), getGlobalIp(deviceId).toOctets(), IpAddress.valueOf("255.255.255.255").toOctets())
                .matchTernary(PiMatchFieldId.of("hdr.tcp.dst_port"), num, Integer.parseInt("FFFF", 16)).build();
            PiAction actionExtToInt = PiAction.builder()
                .withId(PiActionId.of("ingress.nat_control.nat_hit_ext_to_int"))
                .withParameter(new PiActionParam(PiActionParamId.of("dstAddr"), IpAddress.valueOf(ipv4Packet.getSourceAddress()).toOctets()))
                .withParameter(new PiActionParam(PiActionParamId.of("dstPort"), tcpPacket.getSourcePort())).build();

            FlowRule ruleExtToInt = Utils.buildFlowRuleWithIdleTimeout(deviceId, appId, tableId, matchExtToInt, actionExtToInt, AppConstants.IDLE_TIMEOUT);

            flowRuleService.applyFlowRules(ruleIntToExt, ruleExtToInt);
        }
}
```

ONOS Coreのパーサーライブラリを利用

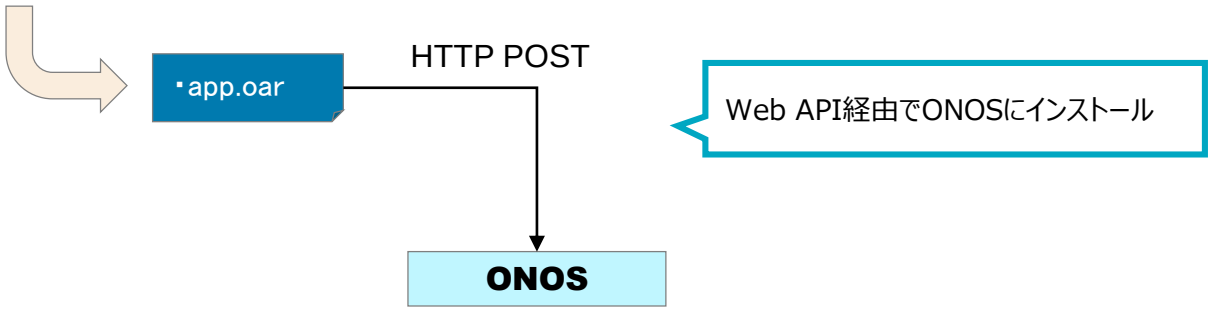
内→外の変換用Flow

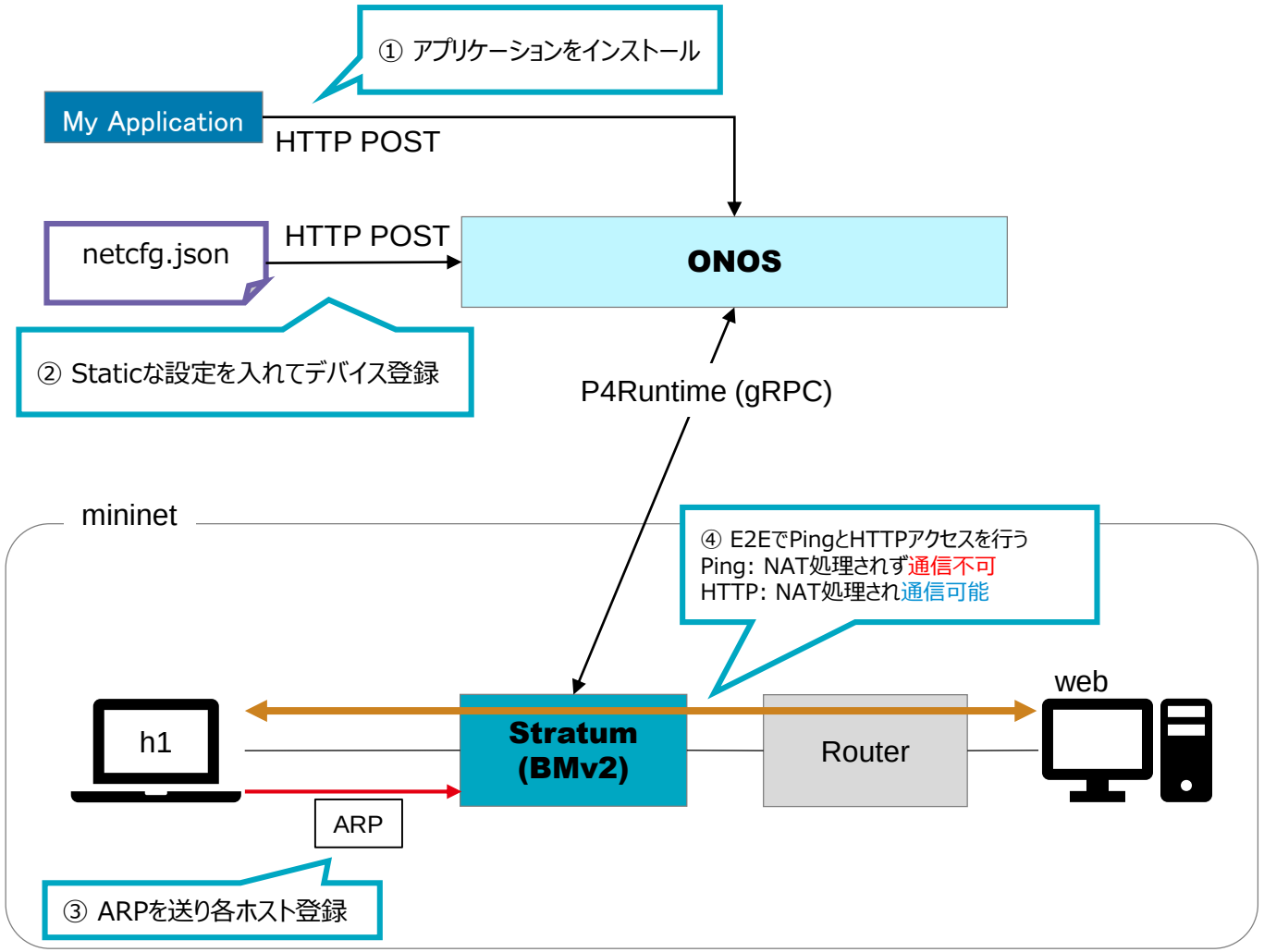
外→内の変換用Flow

ONOSコードのコンパイル

- mavenでコンパイル

```
$ mvn clean package
```




① アプリケーションをインストール
My Application
HTTP POST
netcfg.json
HTTP POST
ONOS
② Staticな設定を入れてデバイス登録
P4Runtime (gRPC)
mininet
④ E2EでPingとHTTPアクセスを行う
Ping: NAT処理されず通信不可
HTTP: NAT処理され通信可能
web
h1
Stratum
(BMv2)
Router
ARP
③ ARPを送り各ホスト登録

まとめ

StratumとONOSを触ってみた所感

- 規模感
  - P4: 777行、Java: 2281行
- データプレーン実装 (Stratum)
  - データプレーンプログラミングは楽しい
    - 思い通りのパケット制御ができる
  - 基本的なことからすべて実装が必要
    - MAC学習、ARP解決・・・・
      - 学習用途には最適？
    - 基本的な動作はONOS fabric.p4などの参照実装を利用可能
- コントロールプレーン実装 (ONOS)
  - コントロールプレーンの実装は大変
    - 実運用を意識した場合のエラーパターンの対応等
  - ONOSに用意されている仕組みを利用することで比較的楽に開発ができる
    - P4Runtimeクライアント実装、高可用性、イベントハンドリング等
- パケットインでの処理は性能問題が懸念
  - 今回のサンプルの場合、初回通信で登録処理が入るため再送処理のないUDP通信はつらい
    - Registerを使ってコントローラを介さずにNAT処理（付録参照）

議論したい内容

- 実運用で用いる場合に問題・課題になる点等、率直な意見をお願いします！
- その他質問等なんでもOKです

付録

APRESIA®

Rev.1 CONFIDENTIAL

開示先：

付録A. Registerを使った実装例

- Registerを使ってコントローラを介さずにNAT処理を実現
- 方法
  - 内→外方向
    - 5-tuppleのhashをインデックスとして、TCP通信における送信元アドレス、ポートをregisterに登録
    - 送信元アドレスをグローバルIPへ、送信元ポートをhashにする
  - 外→内方向
    - 送信先がグローバルIPであれば、registerから、送信先ポートをインデックスとして値を読み出し、送信先アドレスとポートに変換する

registerの定義

```
register<bit<32>>(65535) sip;
register<bit<16>>(65535) sport;

action compute_five_tupple_hash() {
    hash(local_metadata.hash,
         HashAlgorithm.crc32,
         (bit<32>)50000,
         {
             hdr.ipv4.src_addr,
             hdr.ipv4.dst_addr,
             hdr.ipv4.protocol,
             hdr.tcp.src_port,
             hdr.tcp.dst_port
         },
         (bit<32>)15535);
}

action write_register(bit<32> index) {
    sip.write(index, hdr.ipv4.src_addr);
    sport.write(index, hdr.tcp.src_port);
}

action read_register(bit<32> index) {
    sip.read(hdr.ipv4.dst_addr, index);
    sport.read(hdr.tcp.dst_port, index);
}
```

5-tuppleのハッシュ計算

registerに登録

registerから情報取得

内→外はregister登録と送信元アドレス書き換え

```
action nat_int_to_ext(ip_address_t srcAddr) {
    write_register(local_metadata.hash);
    hdr.ipv4.src_addr = srcAddr;
    hdr.tcp.src_port = (bit<16>)local_metadata.hash;
}

action nat_ext_to_int() {
    bit<32> index = (bit<32>)hdr.tcp.dst_port;
    read_register(index);
}
```

外→内はdst_portをindexとしてregisterから情報取得（書換）