OpenFlowコントローラ開発支援ツールの提案
～ロジックおよびフロー識別ラベルの導入～

Yutaka Yasuda, Kyoto Sangyo University

はじめに

- OpenFlow

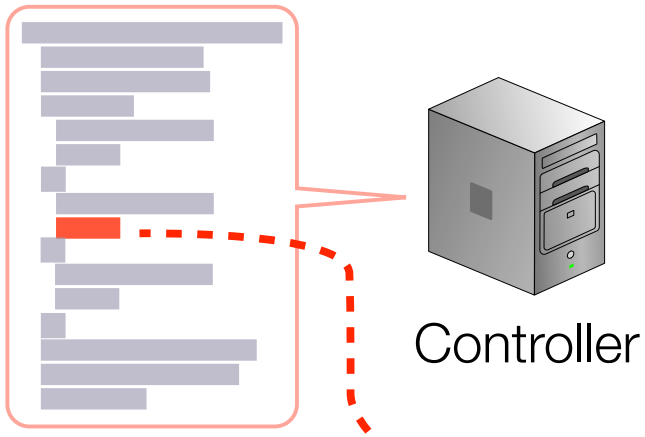
  プログラムによる各スイッチの転送先制御

- コントローラ・プログラミングにおける問題

  自分が書いたコードが実際にどのような転送先制御として
  反映されたのか直接的に確認する方法がない

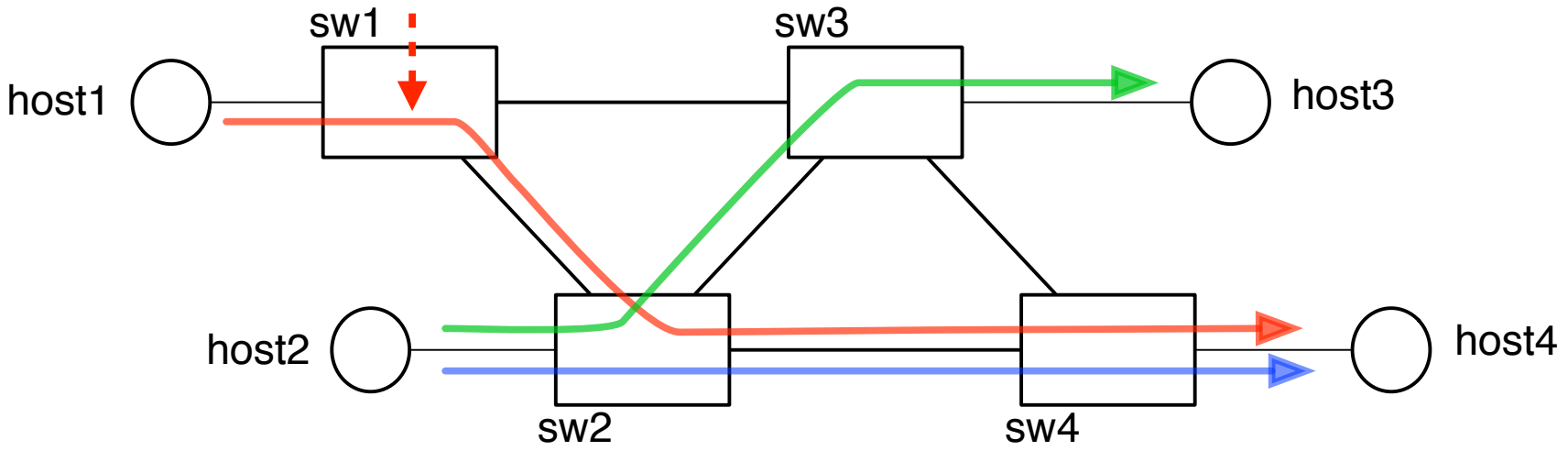
“このコードは、どのフローに対して、どう作用したのか？”
“このフローは、どのコードによって生成されたのか？”

ゴール：開発支援ツールの提案



Controller

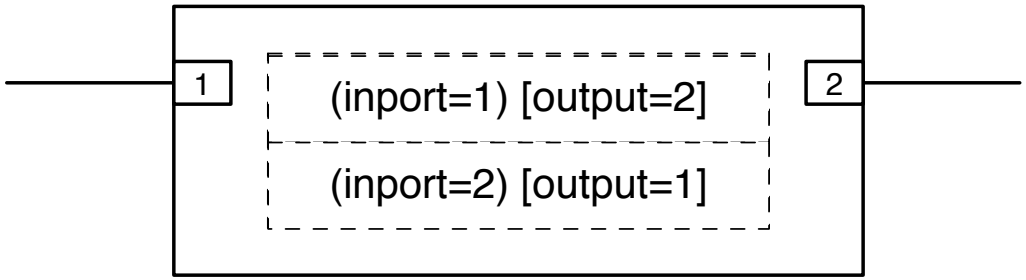
- コードとフロー制御情報を対応づける
- 影響を受けたフローを追跡可能にする



step 1:

ロジックとエントリを結びつける

例：2ポートスイッチとそのためのコード



この場の記法：

(match fields) [actions]

NOX like pseudo code:

```
def packet_in_callback(dpid, inport, reason, len, bufid, packet):
    flow = extract_flow(packet)
    flow[core.IN_PORT] = inport
    if inport == 1 :
        outport = 2
    else:
        outport = 1
    actions = [[openflow.OFPAT_OUTPUT, [0, outport]]]
    inst.install_datapath_flow(dpid, flow,..., actions, ...)
    return CONTINUE
```

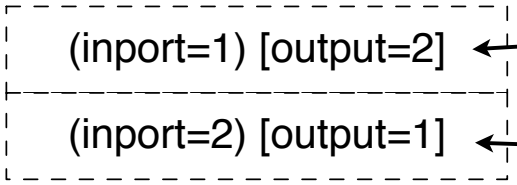
callback

make match fields

make actions

set flow entry

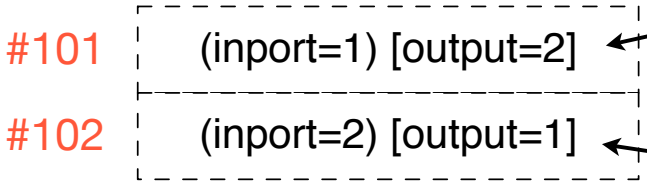
ロジック識別ラベルの設定




```
def packet_in_callback(dpid, inport, reason, len, bufid, p
    flow = extract_flow(packet)
    flow[core.IN_PORT] = inport
    if inport == 1 :
      outport = 2
    else:
      outport = 1
    actions = [[openflow.OFPAT_OUTPUT, [0, outport]]]
    inst.install_datapath_flow(dpid, flow,..., actions, ...)
    return CONTINUE
```

フローエントリとそれに対応するコードに注目

ロジック識別ラベルの設定




```
def packet_in_callback(dpid, inport, reason, len, bufid,
    flow = extract_flow(packet)
    flow[core.IN_PORT] = inport
    if inport == 1 :
        inst.markLogicPoint( 101 )
        outport = 2
    else:
        inst.markLogicPoint( 102 )
        outport = 1
    actions = [[openflow.OFPAT_OUTPUT, [0, outport]]]
    inst.install_datapath_flow(dpid, flow,..., actions, ...)
    return CONTINUE
```

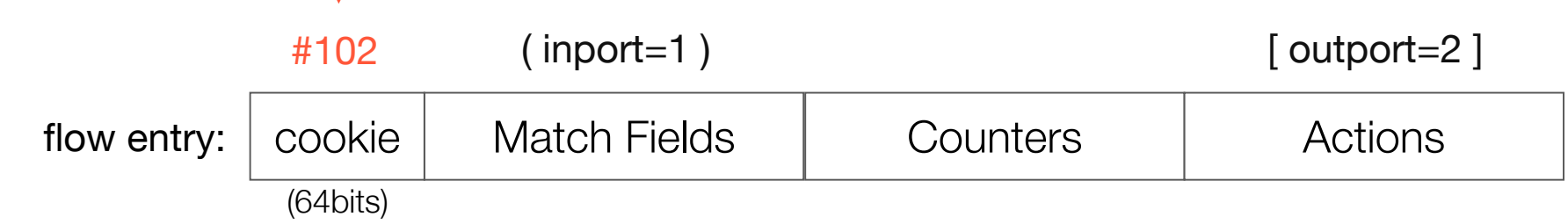
このような関数を追加してフローエントリに識別ラベルを設定する

ロジック識別ラベルの設定

```
def packet_in_callback(dpid, inport, reason, len, bufid,
    flow = extract_flow(packet)
    flow[core.IN_PORT] = inport
    if inport == 1 :
        inst.markLogicPoint( 101 )
        outport = 2
    else:
        inst.markLogicPoint( 102 )
        outport = 1
    actions = [[openflow.OFPAT_OUTPUT, [0, outport]]]
    inst.install_datapath_flow(dpid, flow,..., actions, ...)
    return CONTINUE
```

#101 (inport=1) [output=2]

#102 (inport=2) [output=1]

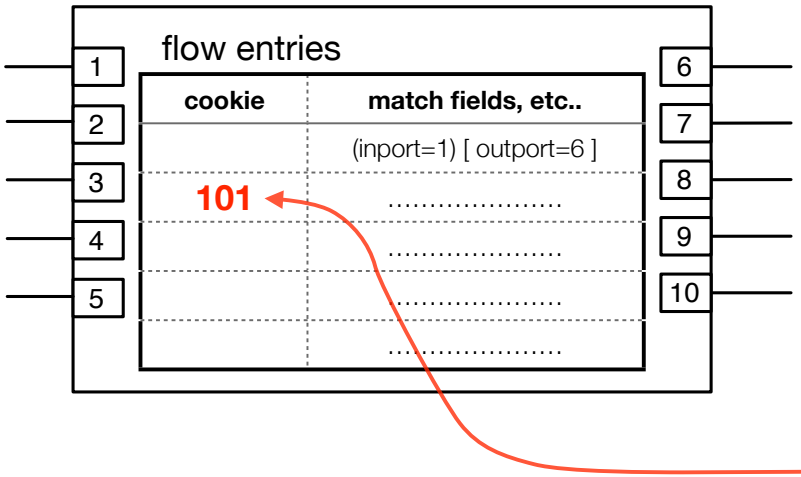


ラベルは Cookie フィールドに書き込む

結果：フローエントリとコードの相互参照

switch

controller program




```
def getOutport(dpid, vport, packet):
    if not OPORTS.has_key(dpid):
        log.err('Unknown dpid=%x on getoutput' % (dpid),system='paniersw')
        return False
    if not OPORTS[dpid].has_key(vport):
        log.err('Invalid (dpid,vport)=(%x,%d) on getOutport' % (dpid, vport),system='paniersw')
        return False
    ops=OPORTS[dpid][vport]
    for (rule, outport, cookie) in ops:
        if rule == RULE80 and testTCPport(80, packet):
            inst.setLogicMark( 101 )
            return (outport, cookie)
        elif rule == RULE443 and testTCPport(443, packet):
            inst.setLogicMark( 102 )
            return (outport, cookie )
        elif rule == RULE25 and testTCPport(25, packet):
            return (outport, cookie)
        elif rule == ANY:
            return (outport, cookie)
    return False
```

フローからコードへの参照：

「このフローエントリにはこのコードが関わった」

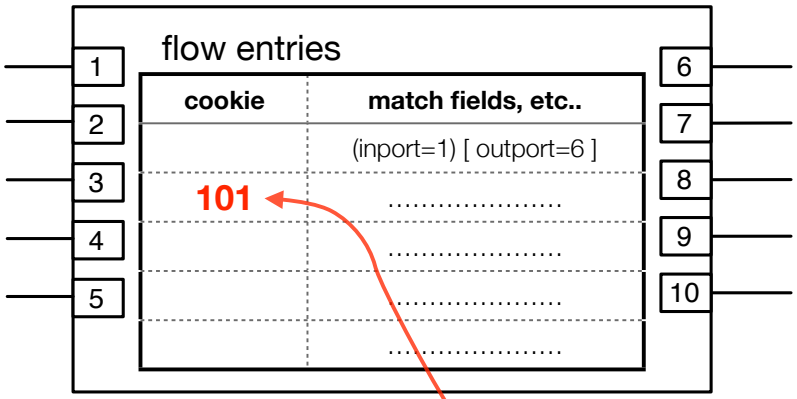
コードからフローへの参照：

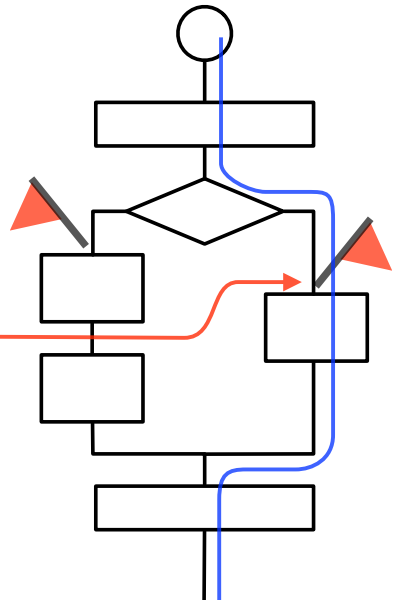
「このコードが影響を与えたフローエントリはどれか」

一つだけの分岐：問題なし

switch

controller program

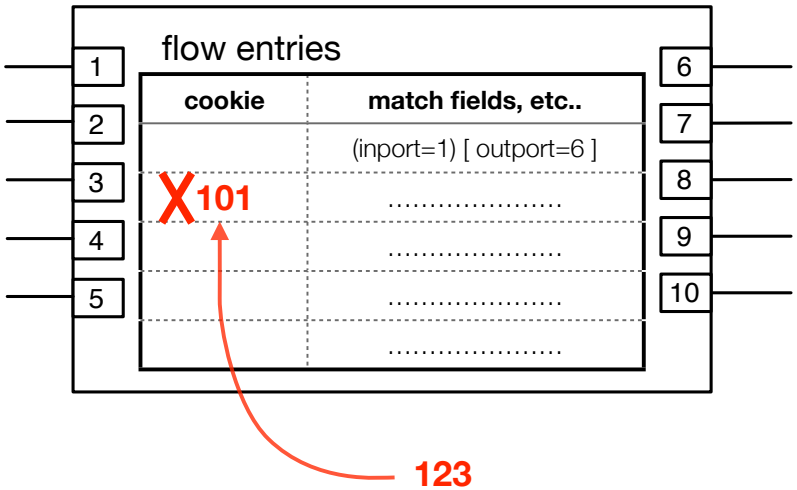




```
def getOutport(dpid, vport, packet):
    if not OPORTS.has_key(dpid):
        log.err('Unknown dpid=%x on getoutput' %
        return False
    if not OPORTS[dpid].has_key(vport):
        log.err('Invalid (dpid,vport)=(%x,%d) on getO
        return False
    ops=OPORTS[dpid][vport]
    for (rule, outport, cookie) in ops:
        if rule == RULE80 and testTCPport(80, pack
            inst.setLogicMark( 101 )
            return (outport, cookie)
        elif rule == RULE443 and testTCPport(443, p
            inst.setLogicMark( 102 )
            return (outport, cookie )
        elif rule == RULE25 and testTCPport(25, pac
            return (outport, cookie)
        elif rule == ANY:
            return (outport, cookie)
    return False
```

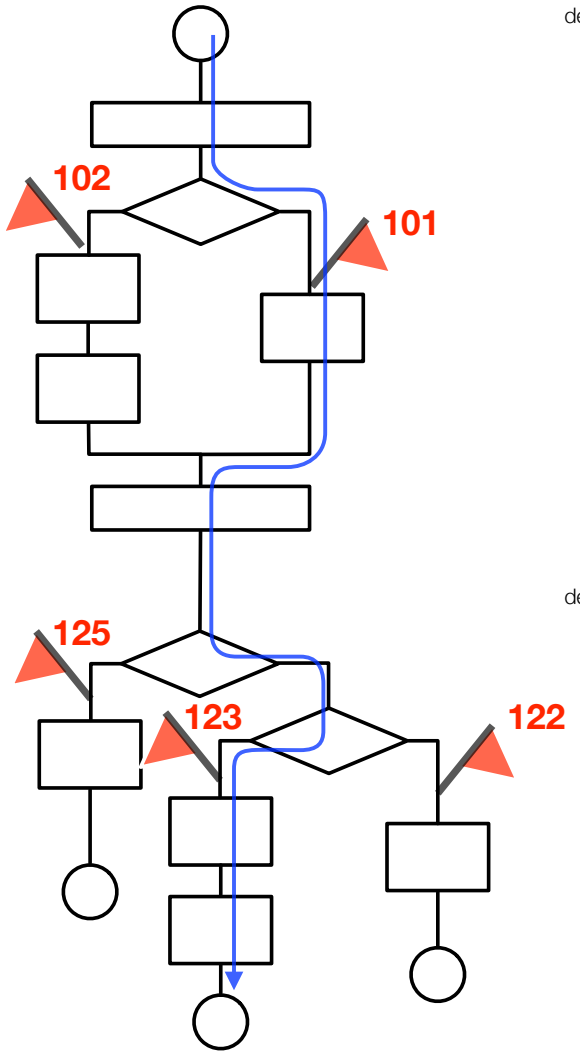
コードとの対象は事実上、分岐との対象と見なせる
（分岐や関数呼び出しなどの重要な通過点に識別ラベル
をつけるため）

複数段の分岐：以前の記録が必要



上書き

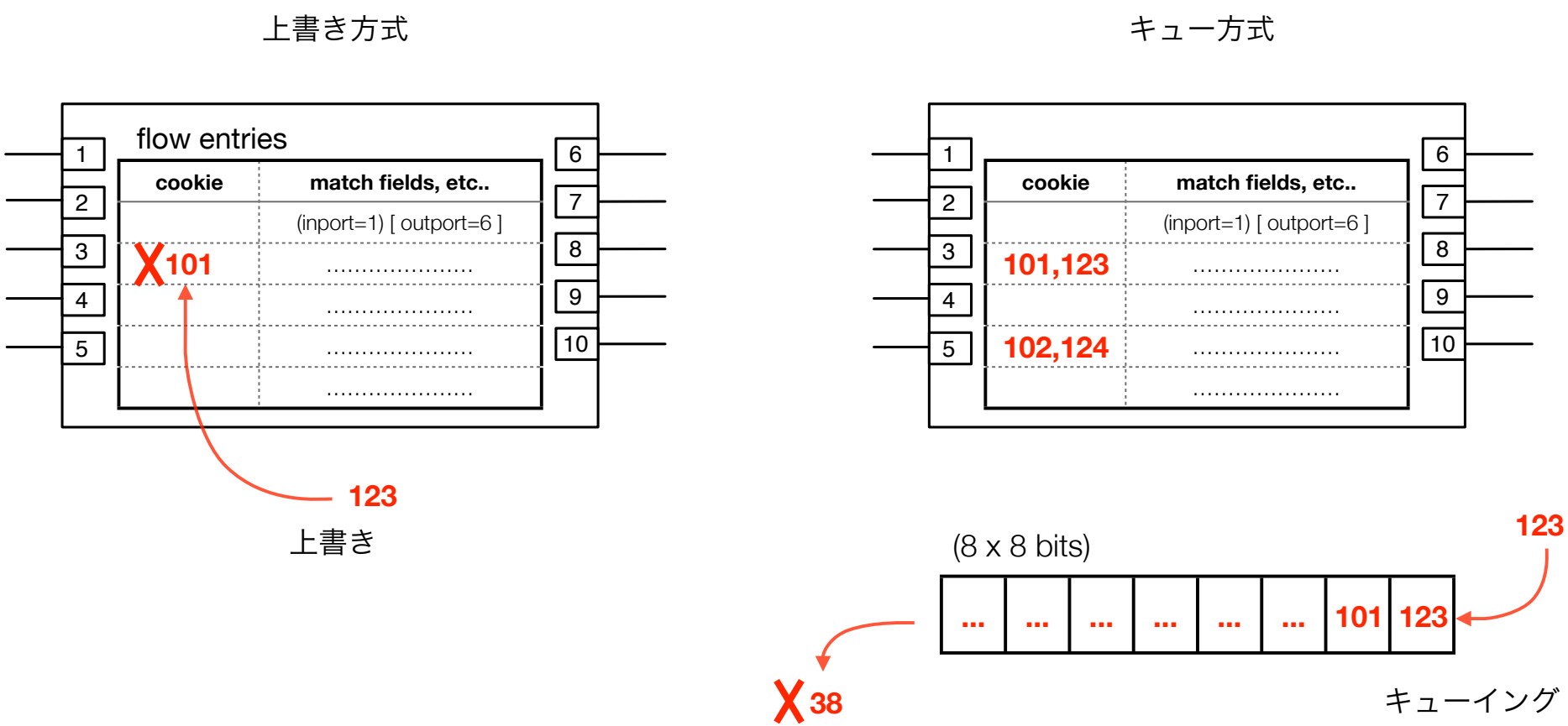
記録すべき重要な分岐が二重にあった場合、その両方（右図では101と123）を記録する必要がある。
単純に直近のラベル（123）で上書きしてはならない。




```
def getOutport(dpid, vport, packet):
    if not OPORTS.has_key(dpid):
        log.err('Unknown dpid=%x on getout
        return False
    if not OPORTS[dpid].has_key(vport):
        log.err('Invalid (dpid,vport)=(%x,%d) o
        return False
    ops=OPORTS[dpid][vport]
    for (rule, outport, cookie) in ops:
        if rule == RULE80 and testTCPport(8
            inst.setLogicMark( 101 )
            return (outport, cookie)
        elif rule == RULE443 and testTCPpor
            inst.setLogicMark( 102 )
            return (outport, cookie )
        elif rule == RULE25 and testTCPport(
            return (outport, cookie)
        elif rule == ANY:
            return (outport, cookie)
    return False
```

```
def forward_l2_packet(dpid, inport, vport, p
    dstaddr = packet.dst.tostring()
    if not ord(dstaddr[0]) & 1 and inst.st[dpic
        prt = inst.st[dpid][dstaddr]
        if  prt[0] == vport:
            inst.setLogicMark( 122 )
            log.err('**warning** learned port = i
            myFlood(dpid, bufid, buf, vport, pa
        else:
            inst.setLogicMark( 123 )
            log.msg('installing flow for ' + str(pa
            flow = extract_flow(packet)
            flow[core.IN_PORT] = inport
            (outport, cookie) = getOutport(dpid
            actions = [[openflow.OFPAT_OUTP
            inst.install_datapath_flow(dpid, flow
                                       openflow.OFP_FL
                                       bufid, openflow.C
                                       inport, buf, cooki
    else:
        inst.setLogicMark( 124 )
        myFlood(dpid, bufid, buf, vport, pack
```

キューイングによる複数ラベルの記録



そのためにキューイングして最近64bitに収まる分だけ記録。右図の例では8bit幅でラベルを作り、8段階を保存している。bit幅と段数は必要に合わせて決めればよい。

制限事項：フローとロジックの結びつけ

- コントローラ SDK が Cookie を使う場合

  Floodlightは 32bit を予約（12bit AppID, 20bit reserve）

- フローエントリ収集によるスイッチ側の負荷

  “In short, the existing statistics mechanisms are (both) relatively high overhead” (J. Mogul et al., 2010)

  J. Mogul et al. “DevoFlow: Cost-Effective Flow Management for High Performance Enterprise Networks”, In Proc. ACM SIGCOMM HotNets-IX, 2010

step 2:

複数スイッチを越えたフローの追跡

フロー追跡（アイディア）

- 欲求：フローの行く末を知りたい

  コードとフローエントリの対照は可能になった

  それ以降のスイッチでの挙動を知りたい

- 問題：スイッチ間でのフローエントリの対照では不十分
- 理由：ワイルドカードの存在

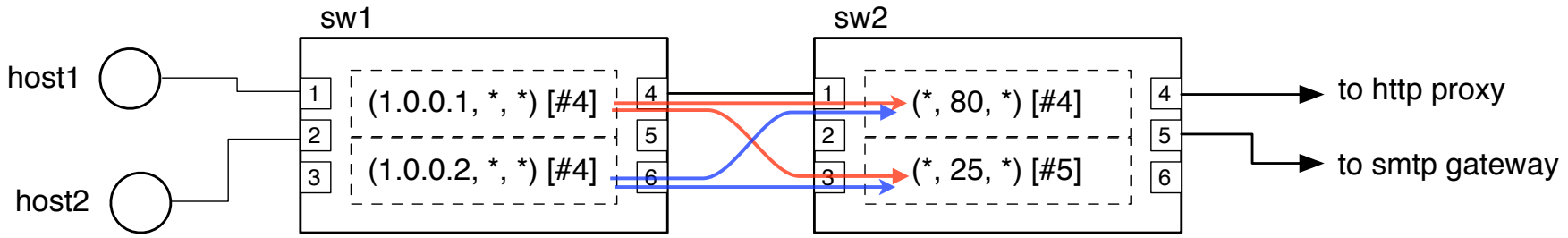
  フローエントリの照合だけでは追えないケースがある

wildcard / maskbit によるフローの流合

- フローエントリの照合だけでは追えないケース

  異なるマッチフィールドにワイルドカードがあり、フローエントリ照合ではフローが追跡できない

例：次のスイッチでフローが分かれるケース
（どちらのフローにどれだけ出たか判らない）



表記：(IP, src port, dst port) [outport]

もし全フローエントリが同じマッチフィールドを使っておれば照合可能
典型例：全マッチフィールドをexactに指定する

フロー追跡（方法）

- パケット自体にラベルを付けて運ばせる
- 設定・検出はスイッチ自身が行う
  →フローエントリに現れるフィールドにラベルを設定する
- Action によってラベルを設定し、次のスイッチに送る
- Match Field によってラベルが設定されたパケットを選別

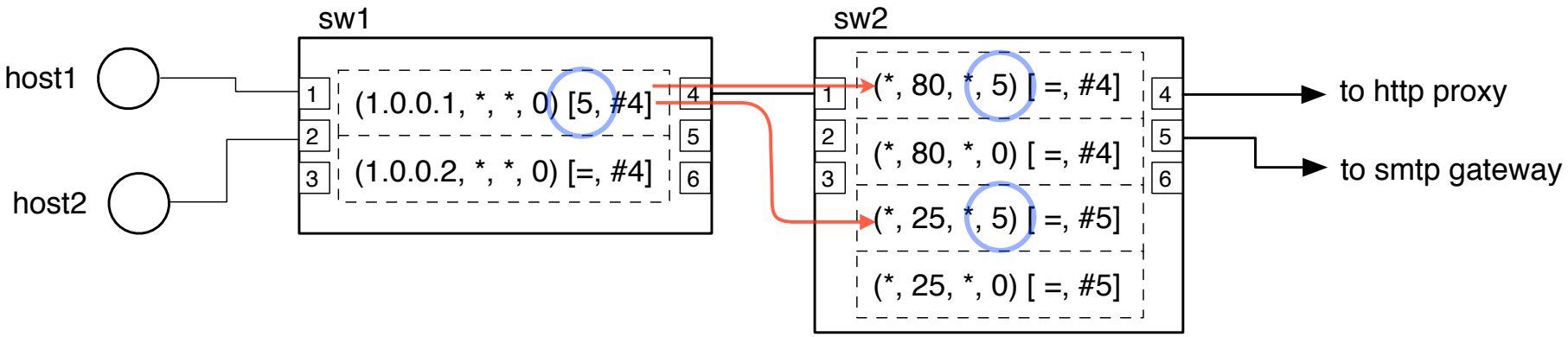
  識別のために「ラベル無し」の状態を用意する

wildcard / maskbit に対応したフローの分離

- パケットに含まれるラベルを用いたフローの分離

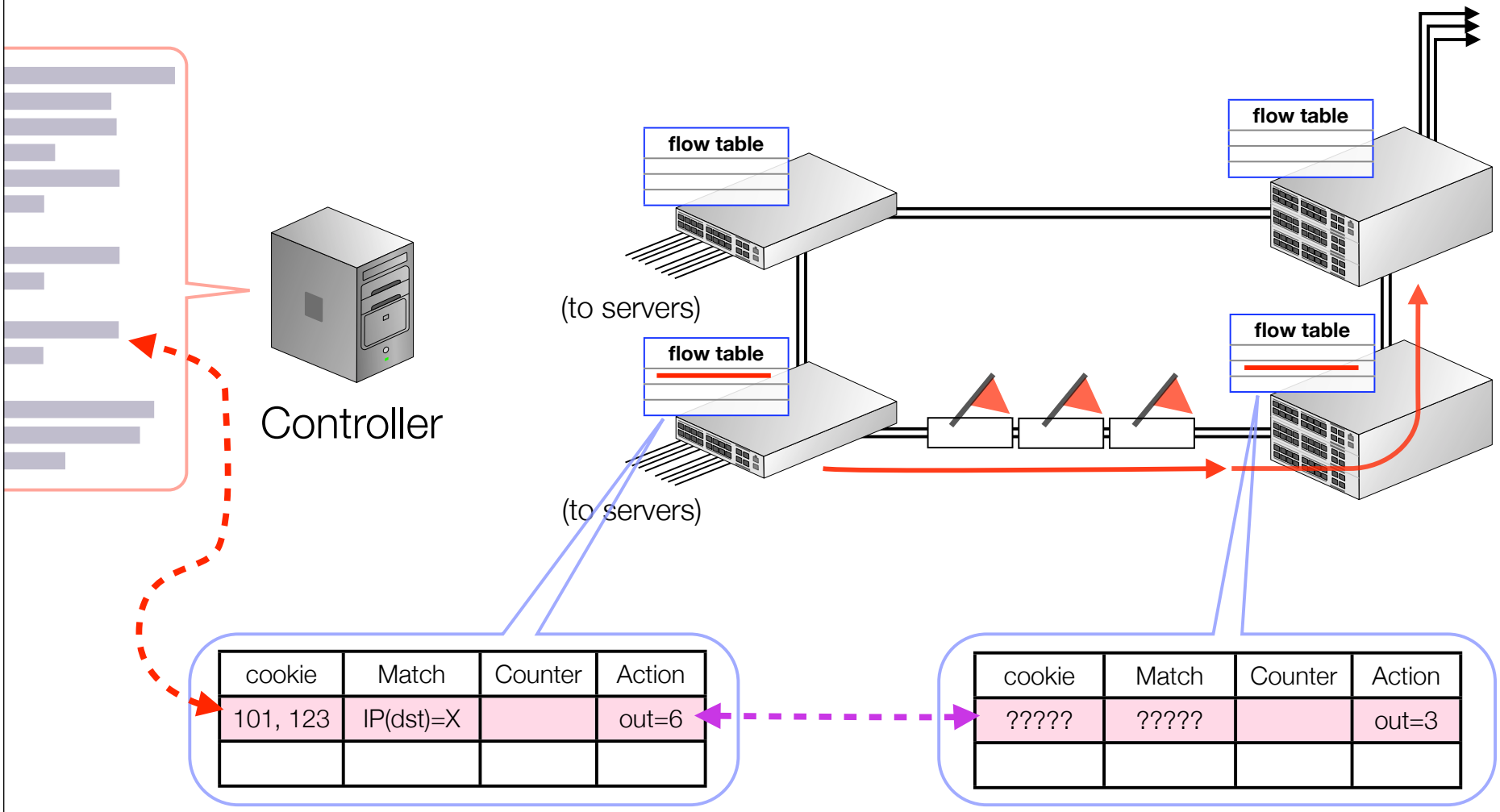
  転送先でエントリが新しく作られ、完全な追跡が可能

例：次のスイッチでフローが分かれるケースへの対応
（ラベルつきとラベル無しのフローエントリがそれぞれ作成される）



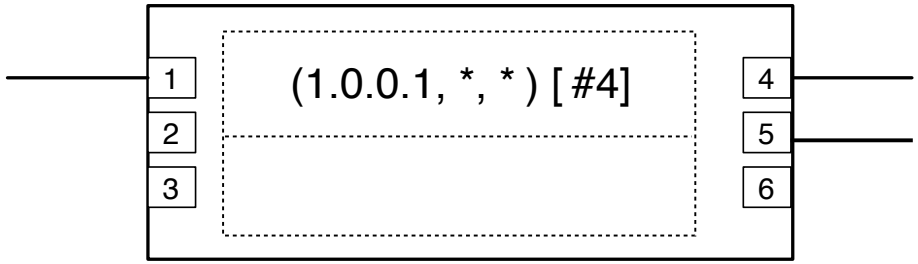
表記：(IP, src port, dst port, label) [label, outport]

フロー追跡（構成図）



追跡用の情報をパケット自体に持たせ、それを鍵にフローエントリを照合する

例：IPアドレスによって振り分けるコード



この場の記法：

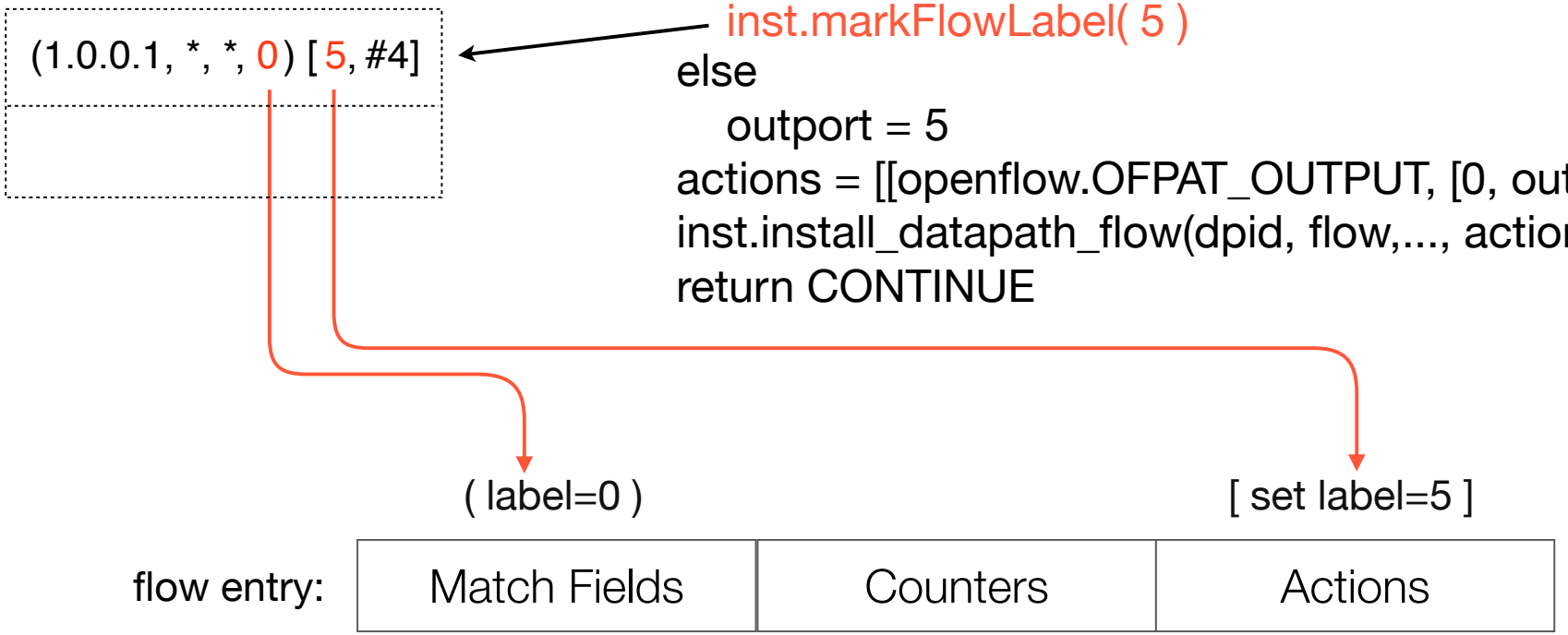
(IP, port-dst, port-src) [#outport]

```
def packet_in_callback(dpid, inport, reason, len, bufid, packet):
    match = extract_flow(packet)
    iph = packet.find('ipv4')
    if ip_to_str(iph.dstip) == "1.0.0.1" :
        outport = 4
    else
        outport = 5
    actions = [[openflow.OFPAT_OUTPUT, [0, outport]]]
    inst.install_datapath_flow(dpid, flow,..., actions, ...)
    return CONTINUE
```

宛先IPアドレスによって出力ポートを変えるスイッチを想定する

フロー識別ラベルの設定

```
def packet_in_callback(dpid, inport, reason, len, bufid, pac
    match = extract_flow(packet)
    iph = packet.find('ipv4')
    if ip_to_str(iph.dstip) == "1.0.0.1" :
        outport = 4
        inst.markFlowLabel( 5 )
    else
        outport = 5
    actions = [[openflow.OFPAT_OUTPUT, [0, outport]]]
    inst.install_datapath_flow(dpid, flow,..., actions, ...)
    return CONTINUE
```



ラベルをつける関数を追加し、Actionによって設定させる。

識別ラベルの記録先

- IPv4 では ToS フィールド(6bit)を利用

  Match Field, Action に含まれている

  そのようなフィールドの中では最も利用率が低い

- VLAN id, MPLS label による実現も可能だが・・・

  利用率の高さ

  予定外のタグ追加による既存プログラムの誤動作

- IPv6 では Flow Label (20bit) の利用を予定

ラベルの重なり：bit 割り当て

- ラベルの重なり：sw1, sw2の両方で別のラベルを設定

  既存のラベルを上書きしてはならない

- ラベルを bit パターンとして設定する

  → label = n のとき ToS = 2^(n-1) とすれば独立に設定可能

- 設定可能なラベルの種類が少ない (ToS なら 6)

  ラベルは文字列(名前)で指定→利用時に番号割当 (要対照表)

  コントローラをリモート操作して動的に(名前で) on/off

制限事項：フローの追跡

- 流用したフィールドが使えない（ v4: ToS / v6: flowlabel ）
- エッジでのラベル回収

  設定された ToS 情報による一般機器の誤動作防止

  トポロジ調査の研究例あり

  ・OFDP : OpenFlow Discovery Protocol by GENI

  ・Topology module by Trema (LLDP base)

- IP以外への対応

IP以外への対応

- 理想的には

  “OpenFlow Switch Spec に追跡用フィールドの追加を提案”

  普及が先か、標準化が先か

  既存のスイッチ機器に適用できない可能性

- 現実的には

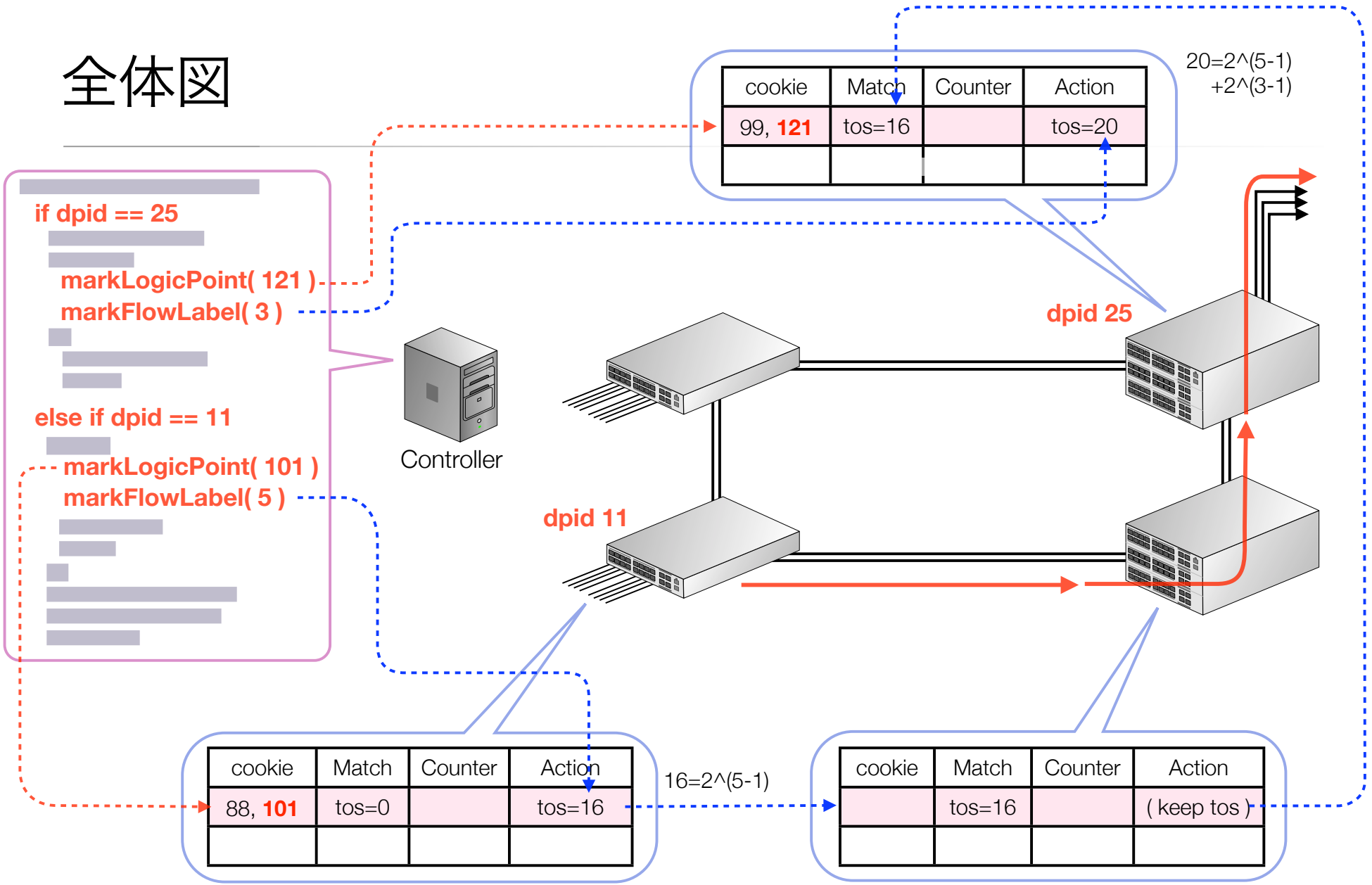
  “大きな問題ではない？”

  OpenFlow は IP（それもv4）に集中している

  L2 利用では MAC 完全マッチで処理する場合が多い

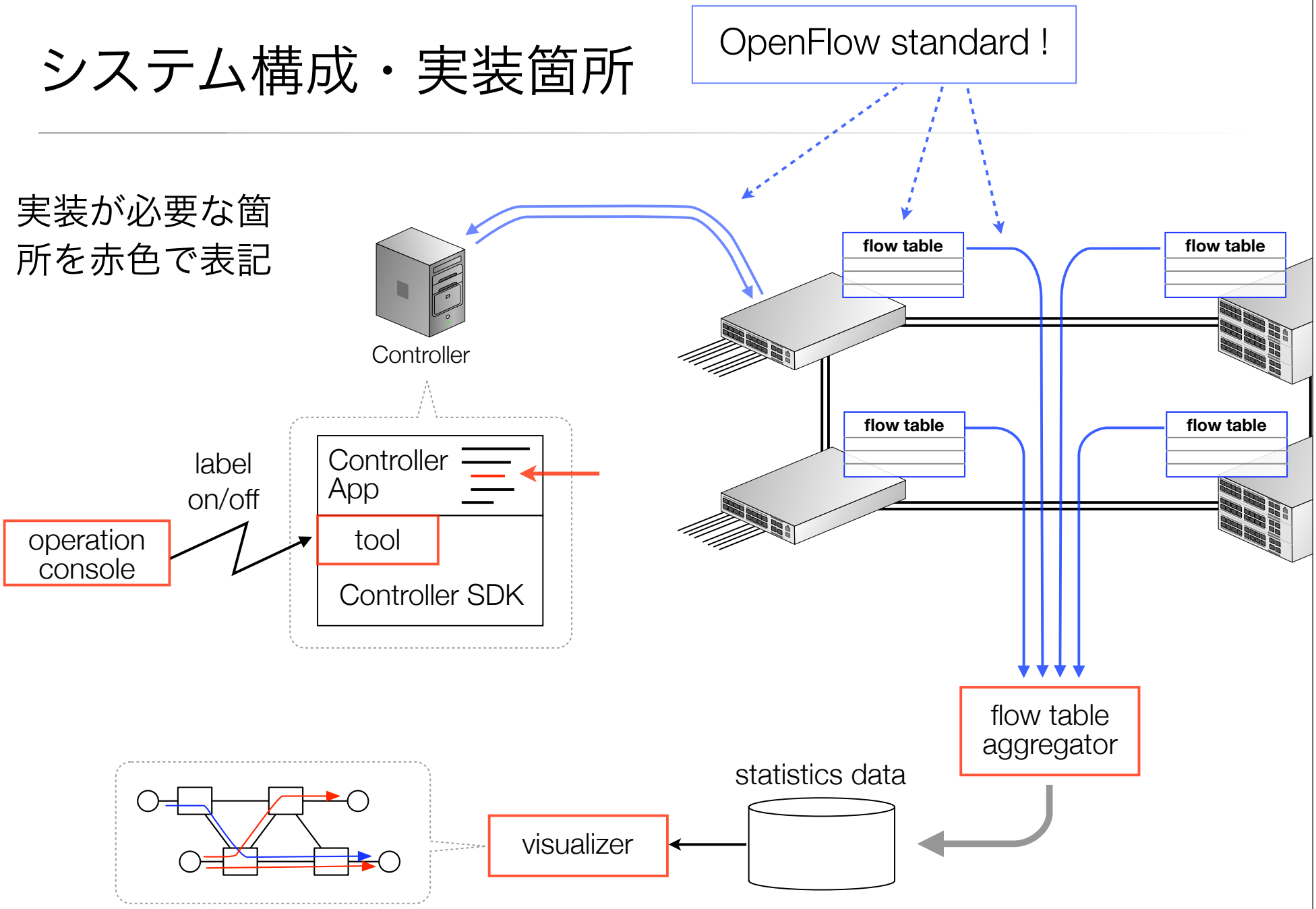
  →ラベル不要・単純にフローエントリ照合で済む

全体図


cookie
Match
Counter
Action
99, 121
tos=16
tos=20
20=2^(5-1)
+2^(3-1)
if dpid == 25
markLogicPoint(121)
markFlowLabel(3)
else if dpid == 11
markLogicPoint(101)
markFlowLabel(5)
Controller
dpid 25
dpid 11
cookie
Match
Counter
Action
88, 101
tos=0
tos=16
16=2^(5-1)
cookie
Match
Counter
Action
tos=16
(keep tos)

システム構成・実装箇所

実装が必要な箇所を赤色で表記



まとめ

- OpenFlow コントローラ開発支援ツールの提案
- コードとフローを対応づける仕組み
  - ロジックとフローの相互参照を可能にする
  - スイッチを越えたフローの追跡を可能にする
- 必要な処置
  - コントローラSDKに機能追加
- 利用
  - 開発中のコードに関数を挿入
  - スイッチは OpenFlow 標準であればよい

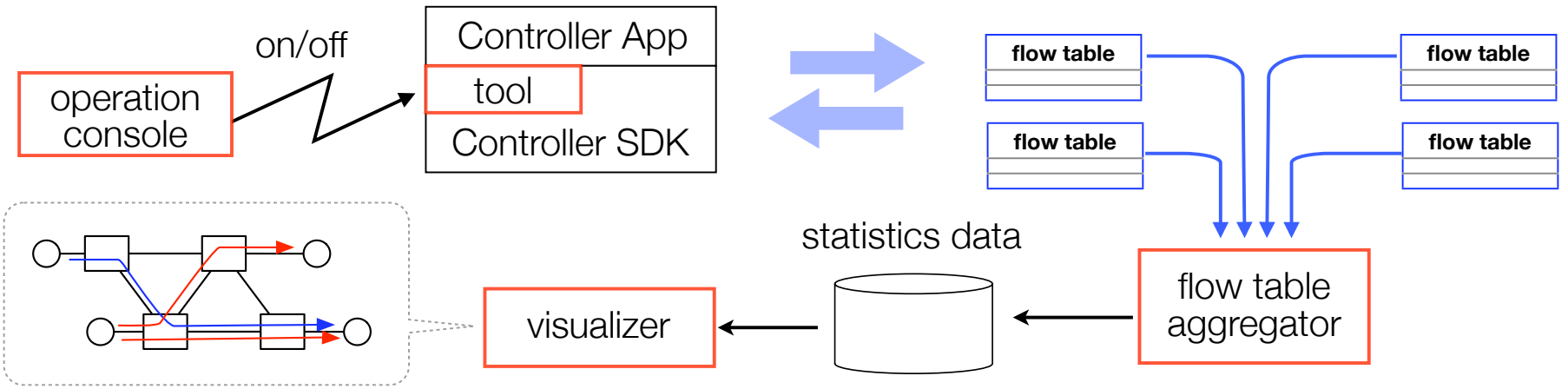
まとめ

- 今後

  NOX 実装の残り・汎用性の確認（他SDKへの移植）

  可視化処理

  新フィールドの OpenFlow 標準への追加提案



Thank you.

共同研究・実装等ご興味あれば一報ください

（商品化も歓迎します）