沖縄オープンラボラトリ主催
第1回ハンズオンセミナー
(RYU, vSwitch編)

2014年2月6日

エヌ・ティ・ティ・コミュニケーションズ株式会社
古澤　徹、西江　将男

目次

目次

ー座学パートー

- **第1章　はじめに**
- 第2章　SDN/OpenFlowの動向
- 第3章　OpenFlowプロトコルの基礎

ーハンズオンパートー

- 第4章　ハンズオンの概要説明
- 第5章　Part1：Mininet(Open vSwitch)の基本操作
- 第6章　Part2：Open vSwitchの基本操作
- 第7章　Part3：Ryuの基本操作
- 第8章　Part4：OpenFlowメッセージのキャプチャ
- 第9章　Part5：簡易Firewallアプリケーションの実装
- 第10章 Part6：応用問題

本セミナーのゴール

- OpenFlowはSDN（Software Defined Networking）を実現する技術として着目されています。
- オープンソースのOpenFlowスイッチ「Open vSwitch」とOpenFlowコントローラ「Ryu」を用いて簡単なOpenFlowアプリケーション開発を体験して頂くことで、OpenFlowの基礎技術とOpenFlowアプリケーションの開発方法を学ぶことができます。

資料の場所

■ 無線アクセスポイント
SSID: ol-hands-on
Pass：okinawa0206
(WPA2パーソナル)

■ 共有フォルダ
¥¥10.0.48.1¥Data¥
user：handson
Pass：okinawa0206

後半の「ハンズオンパート」が始まるまでに
各自、下記のファイルをダウンロードしておいてください
・第1回ハンズオンセミナー.pdf（講義資料）
・vm.ova（ハンズオンで使用するVMイメージ）
・VirtualBoxインストールメディア　※インストールされてない方のみ対象
※vm.ovaの容量が大きいため、早めにダウンロードするようにしてください

アジェンダ

■ 13:00－14:00　SDN/OpenFlowの動向
OpenFlowプロトコルの基礎
■ 14:00－17:00　Open vSwitchとRyuを使った
OpenFlowアプリケーション開発

適宜、休憩をはさみます。
質問がございましたら、随時講師に気軽にお尋ねください。

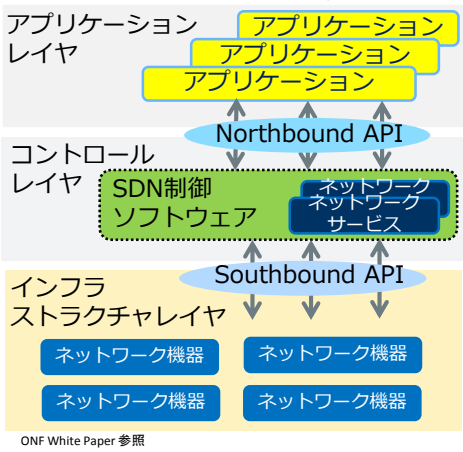
目次

ー座学パートー
- 第1章　はじめに
- **第2章　SDN/OpenFlowの動向**
- 第3章　OpenFlowプロトコルの基礎

ーハンズオンパートー
- 第4章　ハンズオンの概要説明
- 第5章　Part1：Mininet(Open vSwitch)の基本操作
- 第6章　Part2：Open vSwitchの基本操作
- 第7章　Part3：Ryuの基本操作
- 第8章　Part4：OpenFlowメッセージのキャプチャ
- 第9章　Part5：簡易Firewallアプリケーションの実装
- 第10章 Part6：応用問題

SDNのコンセプト

SDNのアーキテクチャ概念図



ONF White Paper 参照

Programable

<抽象化層を通したプログラミング性>
→ネットワークの複雑性を隠蔽・抽象化してステートの一元的な管理を容易に

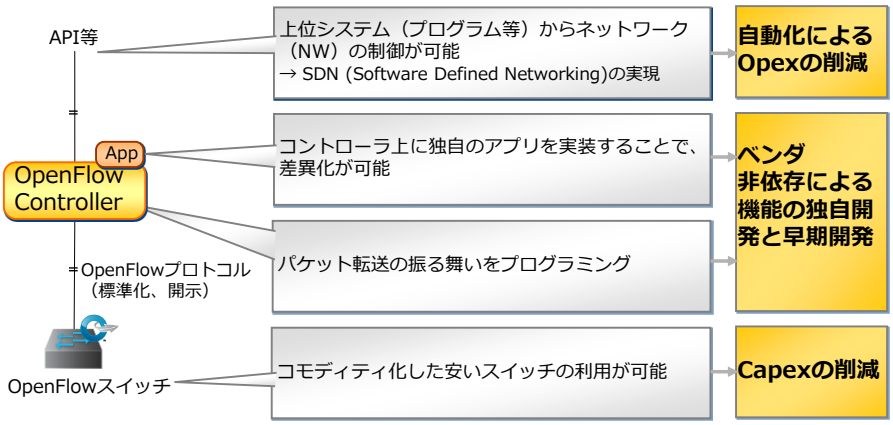
Openness

<オープン化による共通性汎用性>
→多様な機器・機種での構築と一元的に管理を容易に

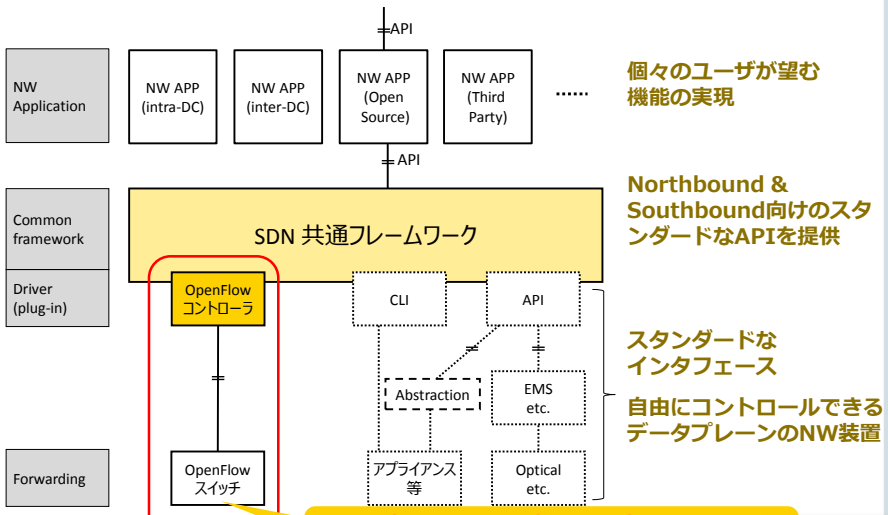
ネットワーク制御を一元的に管理し、ソフトウェアによってプログラム可能にすることで、高度なサービスを迅速に開発・導入可能にする

OpenFlowのコンセプトと期待

OpenFlowはSDNを実現するための１つの要素技術



SDNの将来アーキテクチャ



OpenFlow仕様の歴史

v1.0からv1.3までは早いペースで仕様の改定が行われてきた

OpenFlow仕様の歴史

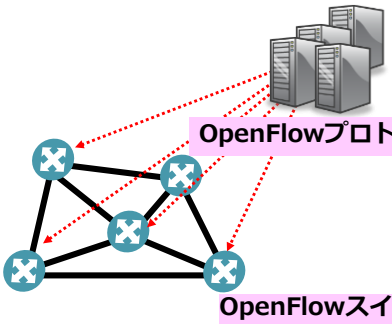
- **2009年12月　: version 1.0.0** (2012年6月：version 1.0.1)
- **2011年2月　: version 1.1.0**
  - ✓ マルチテーブル、グループテーブル、MPLS、QinQ
- **2011年12月　: version 1.2.0**
  - ✓ マルチコントローラ、可変長マッチフィールド、IPv6
- **2012年6月　: version 1.3.0**
  - ✓ 帯域情報、OpenFlow Channel分散、ミスマッチ時のPacket-in見直し
- **2012年9月　: version 1.3.1**
  - ✓ バグフィックス
- **2013年4月　: version 1.3.2**
  - ✓ バグフィックス
- **2013年10月　: version 1.4.0**
  - ✓ バグフィックス、細かな仕様の追加、変更

v1.3以降で一旦Interoperabilityや実装の普及にしばらく注力されている。
本セミナーでもv1.3を使用します

目次

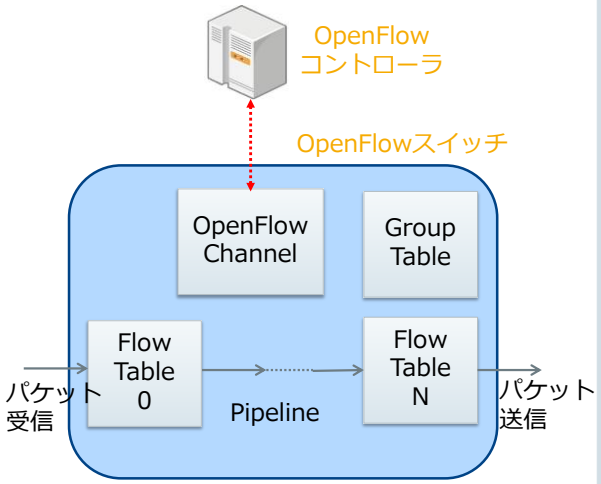
OpenFlowの構成要素

OpenFlowには3つの要素がある



OpenFlowスイッチ

- OpenFlow Channel：
  コントローラとOpenFlowメッセージをやりとりするチャネル
- Flow Table：
  パケットの処理方法を記述するテーブル。1つないし複数のTableを保持する。
- Group Table：
  マルチキャスト等、複数のポートを扱う特殊な処理を行う。
  （本セミナーでは割愛）



Flow Table

Flow Tableは複数のFlow Entryから構成

Match Field	Priority	Counters	Instruction	Timeouts	Cookie	Flags

- **Match Field**：”**Flow**”が記述される。スイッチが受信したパケットがある**Match Field**にマッチした場合、対応する**Instruction**が実行される。
- Priority：マッチの優先度
- Counters：マッチしたパケットに関する統計情報
- **Instruction**：**Match Field**にマッチしたパケットに対して処理を実行する

  Instructionの例：

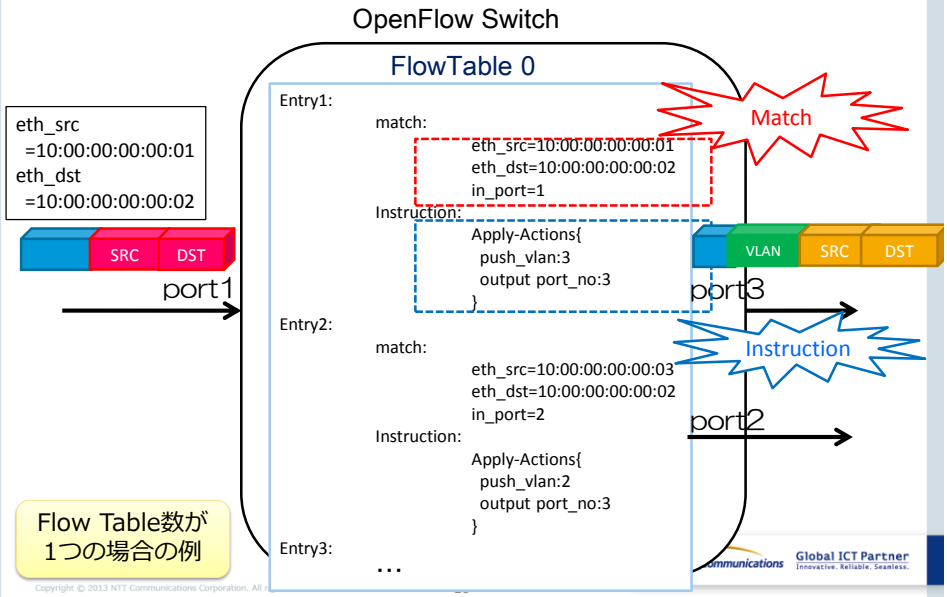
  Apply-Actions *action(s)*: パケットヘッダを書き換える、特定のポートから送信する等、1つまたは複数の*Action(s)*を実行する

  Goto-Table *next-table-id*: *next-table-id*番目のFlow Tableに遷移し、引き続きマッチングを実行する

- Timeouts：Flow Entryの自動削除タイマー値
- Cookie：Flow Entryの識別子
- Flags：制御用フラグ

NTT Communications Global ICT Partner Innovative. Reliable. Seamless.

パケットフォワーディングプロセス

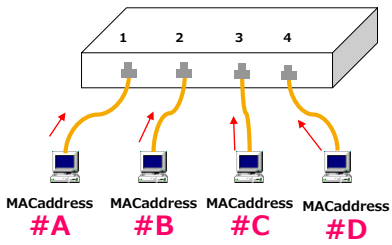


NTT Communications Global ICT Partner Innovative. Reliable. Seamless.

参考：既存のL2スイッチの処理

MAC table

Port	MAC address
1	00:0D:60:00:00:01
2	Unknown
3	00:0D:60:00:00:03
…	…



- 既存のL2スイッチはMAC tableに基づき転送
- 既存のL3スイッチはさらにARP tableやRouting tableを保持
- OpenFlowスイッチはFlow tableのみ保持

Flowの定義

L1からL4までのパケットヘッダ情報の
組み合わせからFlowが定義される

Ingress Port	Eth src	Eth dst	Ether type	VLAN ID	VLAN priority	IPv4 src	Pv4 dst	Proto Col Type	IPv4 ToS	TCP/ UDP src port	TCP/ UDP dst port

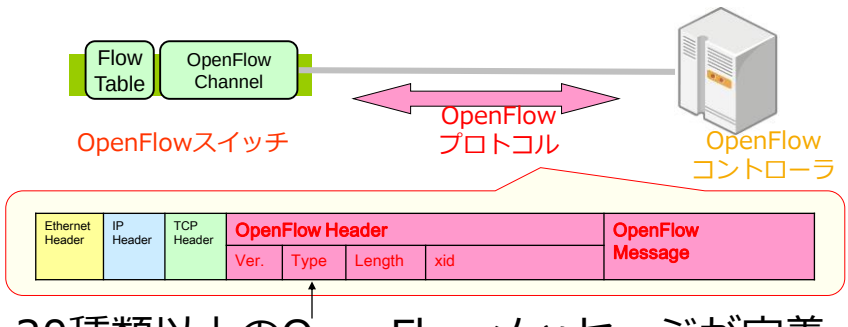
※この他、MPLS、PBB等のヘッダも対応可能

Flowの例

マッチングに用いるヘッダのみ記述
その他はWildcard扱い

Ingress Port	Eth src	Eth dst	Ether type	VLAN ID	VLAN priority	IPv4 src	Pv4 dst	Proto Col Type	IPv4 ToS	TCP/ UDP src port	TCP/ UDP dst port	Instruction
*	*	00:1f...	*	*	*	*	*	*	*	*	*	Apply-Actions {Set_dl_src=00:1f... Output:port6}
Ingress Port	Eth src	Eth dst	Ether type	VLAN ID	VLAN priority	IPv4 src	Pv4 dst	Proto Col Type	IPv4 ToS	TCP/ UDP src port	TCP/ UDP dst port	Instruction
*	*	*	*	*	*	* 10.10.10.0/24		*	*	*	*	Apply-Actions {Set_dl_src=AA:BB... Output:port6}
Ingress Port	Eth src	Eth dst	Ether type	VLAN ID	VLAN priority	IPv4 src	Pv4 dst	Proto Col Type	IPv4 ToS	TCP/ UDP src port	TCP/ UDP dst port	Instruction
*	*	*	*	*	*	*	*	*	* TCP:22		*	Apply-Actions {DROP}

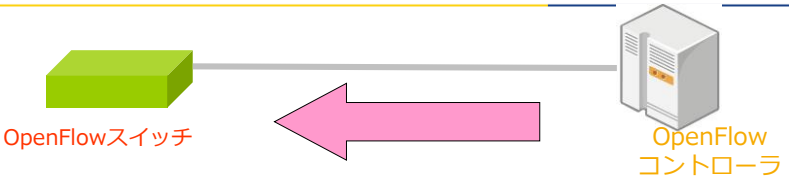
OpenFlow プロトコル



20種類以上のOpenFlowメッセージが定義

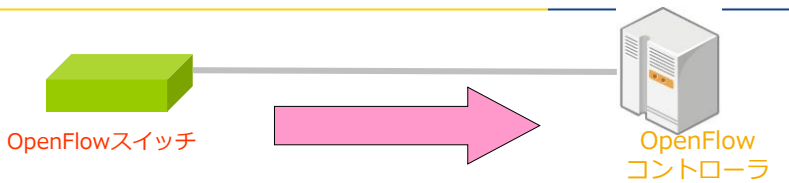
3種類のメッセージ:
- Controller-to-switch
- Asynchronous
- Symmetric

Controller-to-Switchメッセージ



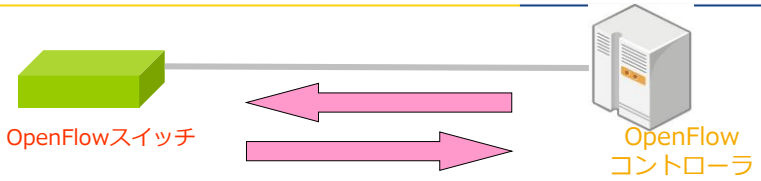
- コントローラからスイッチに対して送信されるOpenFlowメッセージ
  - Features：対応機能の調査
  - Configuration：パラメータ設定
  - **Modify-State：Flow Tableを更新**
  - Read-State：スイッチの統計情報等を取得
  - **Packet-out：ペイロードを含むパケットをスイッチに送信し、指定ポートから転送**

Asynchronousメッセージ



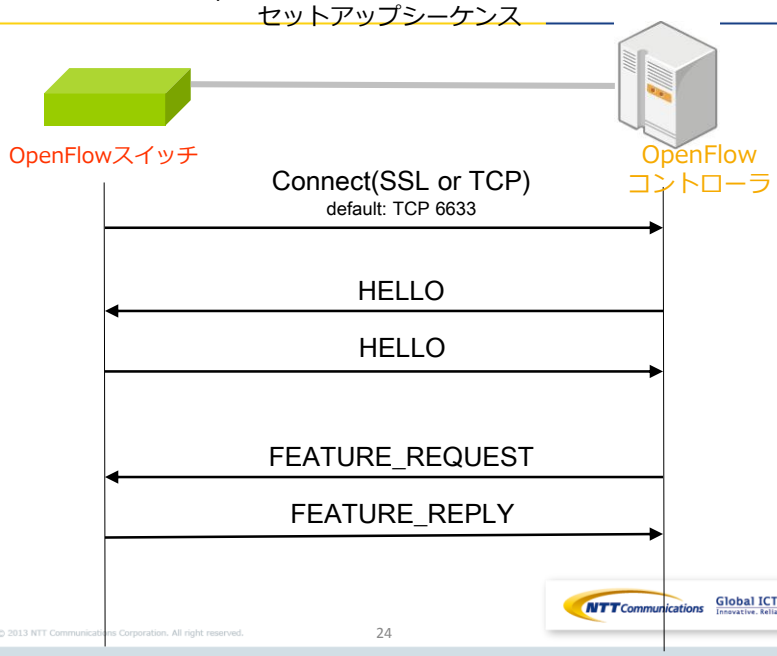
- スイッチからコントローラに対して送信されるOpenFlowメッセージ
  - **Packet-in：スイッチが受信したパケットをコントローラに転送**
  - Flow-Removed：Flow entry削除イベントをコントローラに通知
  - Port-Status：スイッチのポート情報をコントローラに通知
  - Error：エラーメッセージを通知

Symmetricメッセージ

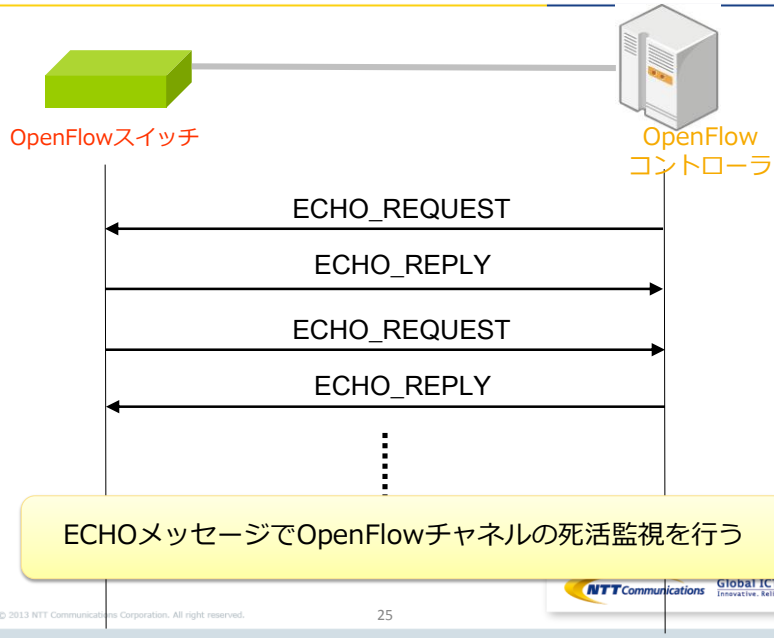


- 双方向でやりとりされるOpenFlowメッセージ
  - Hello：OpenFlow Channelコネクションのセットアップ
  - Echo：スイッチとコントローラ間の死活監視

OpenFlow Channelコネクションの セットアップシーケンス

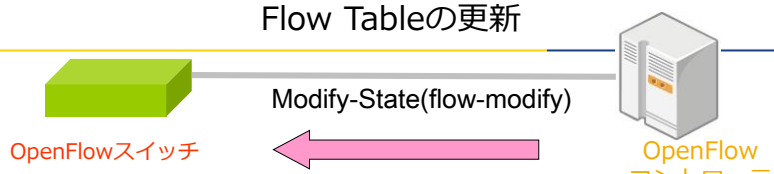


コネクションの維持



ECHOメッセージでOpenFlowチャネルの死活監視を行う

Flow Tableの更新



ADD	新しいFlow Entryの登録
MODIFY	マッチするFlow Entryの更新
MODIFY_STRICT	マッチするFlow Entryの更新(Wildcardも考慮される)
DELETE	マッチするFlow Entryの削除
DELETE_STRICT	マッチするFlow Entryの削除(Wildcardも考慮される)

MODIFY/DELETEと
MODIFY_STRICT/DELETE_STRICTの違い

例 (1) If **DELETE "in_port=*"** is received:

In_port	MAC
1	AA:BB:…
*	*

→

In_port	MAC
(empty)	

All flows deleted

例 (2) If **DELETE_STRICT "in_port=*"** is received:

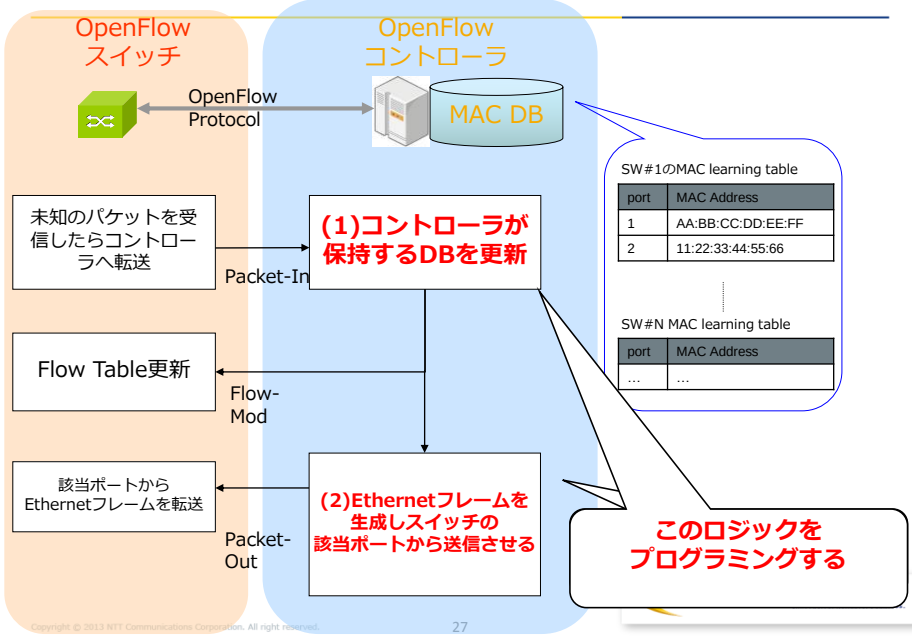
In_port	MAC
1	AA:BB:…
*	*

→

In_port	MAC
1	AA:BB:…

Flows whose in_port tuple is "*" deleted

L2スイッチアプリケーション開発のケーススタディ（1/3）



L2スイッチアプリケーション開発のケーススタディ（2/3）

参考：既存のL2スイッチのフレーム送信動作

(3)MAC Tableから
”00:0D:60:00:00:02”を探索

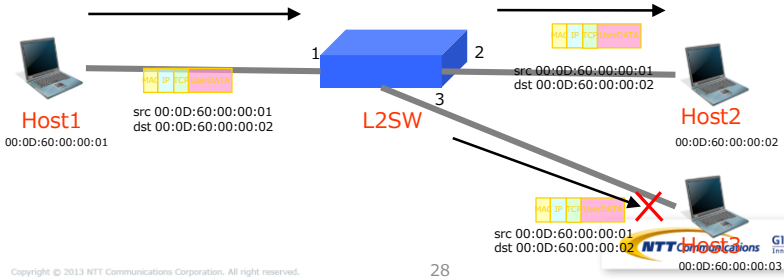
MAC table

Port	MAC address
1	00:0D:60:00:00:01
2	-
3	-

(2)ホスト1の送信元
MACアドレスを学習

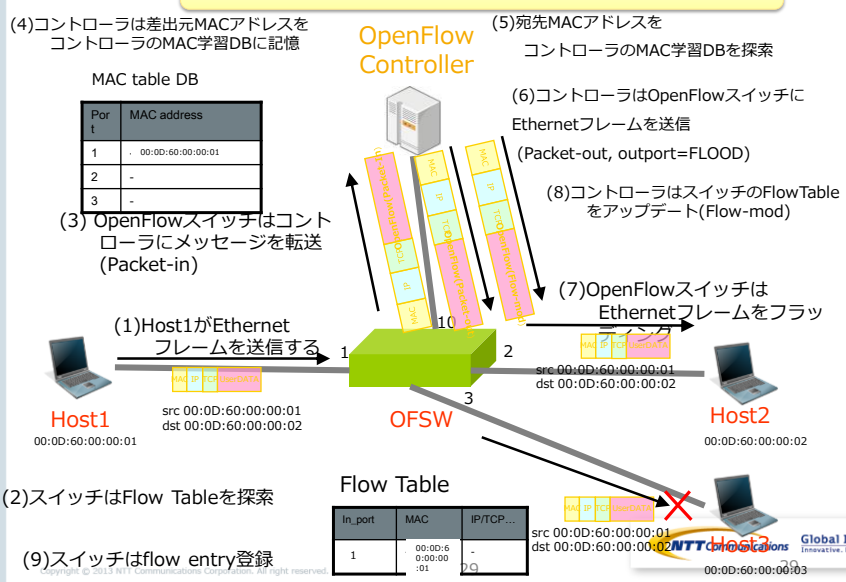
(1)Host1がEthernet
フレームを送信する

(4)MAC Table上で宛先MACアドレスが
見つからない場合、受信ポート以外の
全ポートからフラッディング



L2スイッチアプリケーション開発のケーススタディ（3/3）

OpenFlowで同様の処理を行う場合のシーケンス



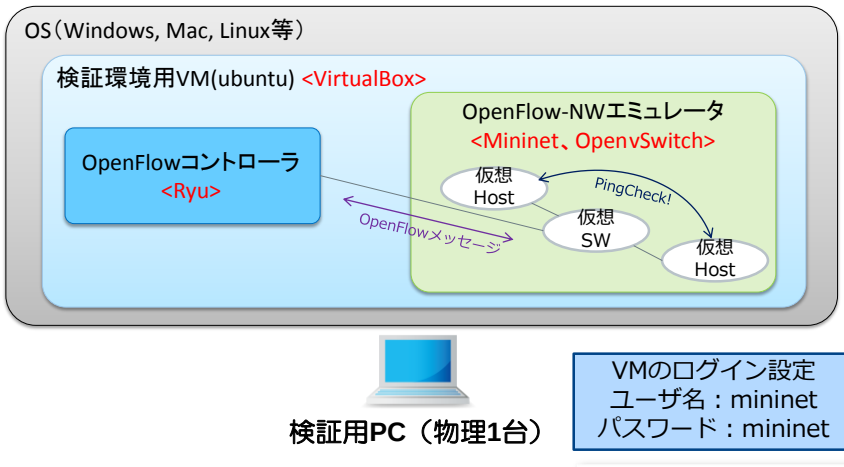
目次

Hands-On

- 内容
  - オープンソースを使用し、OpenFlowネットワークを構築
  - OpenFlowコントローラやOpenFlowスイッチはVMで構築
- 使用するオープンソース
  - Ryu
    NTT研究所が開発したSDNコントローラを実装するためのPythonベースのフレームワーク（OpenFlowプロトコルをサポート）
  - Open vSwitch
    仮想ソフトウェアスイッチ（OpenFlowスイッチとして動作可）
  - Mininet
    ネットワークエミュレータ（一つのLinuxカーネル上で、複数のホスト、スイッチ、ルーターを組み合わせて仮想的にネットワークを構築可能）

Hands-On環境について

- Hands-On環境は、VirtualBox互換のVMイメージとして配布



検証用PC（物理1台）

VMのログイン設定
ユーザ名：mininet
パスワード：mininet

目次

ー座学パートー
- 第1章　はじめに
- 第2章　SDN/OpenFlowの動向
- 第3章　OpenFlowプロトコルの基礎

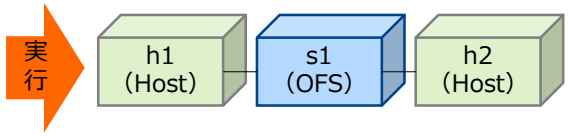
ーハンズオンパートー
- 第4章　ハンズオンの概要説明
- **第5章　Part1：Mininet(Open vSwitch)の基本操作**
- 第6章　Part2：Open vSwitchの基本操作
- 第7章　Part3：Ryuの基本操作
- 第8章　Part4：OpenFlowメッセージのキャプチャ
- 第9章　Part5：簡易Firewallアプリケーションの実装
- 第10章 Part6：応用問題

Mininet

- Mininetとは
  - ネットワークエミュレータ（一つのLinuxカーネル上で、複数のホスト、スイッチ、ルーターを組み合わせて仮想的にネットワークを構築可能）
  - スタンフォード大学で開発されたオープンソース
- メリット
  - 複数台のサーバやOpenFlowスイッチを準備してテスト環境を構築するのはコスト、リソース的に簡単ではないが・・・
  - Mininetを使えば、簡単に仮想のスイッチ、ホスト、リンクを作成することができる！

```
switch = addSwitch('s1')
host1 = addHost('h1')
host2 = addHost('h2')
addLink(switch,host1)
addLink(switch,host2)
```

Mininetのトポロジ設定 (pythonで記述)



Mininetのイメージ

Mininetの基本操作(1)

■ トポロジの定義（pythonスクリプトを使用）

/home/mininet/handson/mininet_sample.py

接続先のコントローラの設定
→IP：127.0.0.1
port: 6633(OpenFlow)

スイッチ(名前=s1)の追加

ホスト(名前=h1)の追加
ホスト(名前=h2)の追加

リンク(s1-h1間)の追加
リンク(s1-h2間)の追加

s1のversionをOF1.3に指定

```
from mininet.cli import CLI
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import Link
from mininet.node import RemoteController, OVSSwitch
from mininet.util import dumpNodeConnections
from mininet.log import setLogLevel

def ofp_version(switch,protocols):
    protocols_str=','.join(protocols)
    command='ovs-vsctl set Bridge %s protocols=%s' % (switch,protocols_str)
    switch.cmd(command)

def sampleTopo():
    net = Mininet(switch=OVSSwitch)
    net.addController('c0',controller=RemoteController,ip='127.0.0.1',port=6633)

    s1 = net.addSwitch('s1')

    h1 = net.addHost('h1')
    h2 = net.addHost('h2')

    net.addLink(s1,h1)
    net.addLink(s1,h2)

    net.start()
    print "Dumping node connections"
    dumpNodeConnections(net.hosts)
    ofp_version(s1,['OpenFlow13'])
    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    sampleTopo()
```


Mininetの基本操作(2)

■ Mininetの起動
- 作成したPythonスクリプトを実行する

```
$ cd /home/mininet/handson
$ sudo python mininet_sample.py
```

■ MininetのCLIコマンド（１）
- Help表示

```
mininet> help
```

- トポロジー（接続関係）の確認

```
mininet> net
```

```
<出力結果の例>
c0
s1 lo:  s1-eth1:h1-eth0 s1-eth2:h2-eth0
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
```

h1のeth0とs1のeth1
h2のeth0とs1のeth2
が接続していることがわかる

Mininetの基本操作(3)

■ MininetのCLIコマンド（２）

- 仮想ホスト・スイッチ上で任意コマンドを実行することも可能

例：

h1からh2に対してping

```
mininet> h1 ping h2
```

s1でifconfig

```
mininet> s1 ifconfig
```

h2でhttpサーバ起動

```
mininet> h2 python SimpleHTTPServer
```

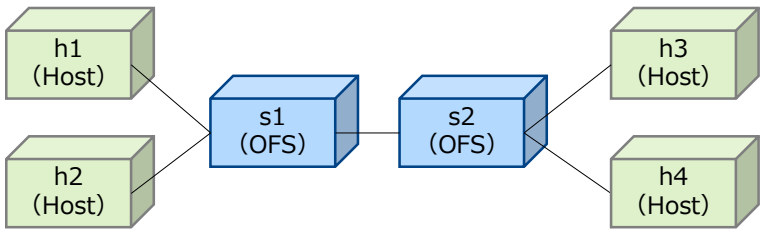
- ターミナル（xterm）の起動

```
mininet> xterm h1
```


Hands-On Part1

（課題）下図のようなNWトポロジーをMininetで作成してください

- ポイント
  - ✓ mininet_sample.pyを改造すれば作成可能
  - ✓ ファイル名は”mininet_handson1.py”に変更すること
  - ✓ s1とs2はOpenFlow1.3で動作するようにすること



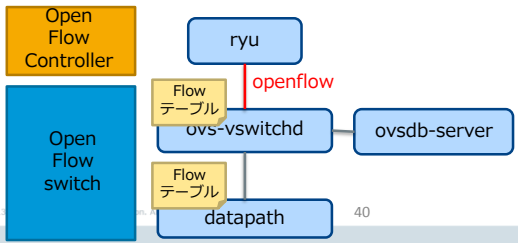
目次

ー座学パートー
- 第1章　はじめに
- 第2章　SDN/OpenFlowの動向
- 第3章　OpenFlowプロトコルの基礎

ーハンズオンパートー
- 第4章　ハンズオンの概要説明
- 第5章　Part1：Mininet(Open vSwitch)の基本操作
- **第6章　Part2：Open vSwitchの基本操作**
- 第7章　Part3：Ryuの基本操作
- 第8章　Part4：OpenFlowメッセージのキャプチャ
- 第9章　Part5：簡易Firewallアプリケーションの実装
- 第10章 Part6：応用問題

Open vSwitch

- Open vSwitchとは
  - オープンソースの仮想ソフトウェアスイッチ
  - OpenFlowに対応　※本ハンズオンではOpenFlow1.3を対象
- アーキテクチャ(主要コンポーネント)
  - ユーザ空間で動作
    - ✓ ovs-vswitchd：Open vSwitchデーモン(複数SWを管理可能)
    - ✓ ovsdb-server：構成情報を管理するDBサーバ
  - カーネル空間上で動作
    - ✓ datapath(openvswitch_mod.ko)：
      パケットの転送処理を行うカーネルモジュール



Open vSwitchの初期設定

※ハンズオン環境（VM）には既にOpen vSwitchをインストールしてあります

■ Open vSwitchの起動

```
$ sudo /etc/init.d/openvswitch-switch start
```

ovs-vswitchd, ovsdb-server等のプロセスが起動

■ ブリッジ（OpenFlowスイッチ1台相当）の作成

```
$ sudo ovs-vsctl add-br s1                                  # create s1 bridge
$ ifconfig s1                                               # check if br0 is created
$ sudo ovs-vsctl set bridge s1 protocols=OpenFlow13         # set OpenFlow version
$ sudo ovs-vsctl list-br
```

■ ポート（OpenFlowスイッチのポート）の作成

```
$ sudo ovs-vsctl add-port s1 eth2                           # add eth2 port to br0
$ sudo ovs-vsctl add-port s1 eth3                           # add eth3 port to br0
$ sudo ovs-vsctl list-ports s1                              # show the ports of br0
$ sudo ifconfig s1 up                                       #
enable IFs to work
$ sudo ifconfig eth2 up
$ sudo ifconfig eth3 up
$ sudo ovs-vsctl show                                       # show the bridge
settings
$ ifconfig
```

※これらの作業は、初期設定時にのみ実行するものです

Open vSwitchのフロー登録コマンド

■ フローの確認とフローの削除

```
$ sudo ovs-ofctl –O OpenFlow13 dump-flows s1  # Show Flow Table of s1. 1 entry at first.
$ sudo ovs-ofctl –O OpenFlow13 del-flows s1   # Delete default flows
```

■ フローエントリーの登録

```
$ sudo ovs-ofctl –O OpenFlow13 add-flow s1 'priority=100,in_port=1,actions=output:2'
$ sudo ovs-ofctl –O OpenFlow13 add-flow s1 'priority=100,in_port=2,actions=output:1'
```

■ フローの確認　※登録後のため、フローが確認できるはず

```
$ sudo ovs-ofctl –O OpenFlow13 dump-flows s1      # You can see 2 flow entries.
```

（参考）コントロールプレーンの設定

■ DatapathID（SWの識別するためのID）の設定

```
$ sudo ovs-vsctl set bridge s1 other-config:datapath-id=0000000000000001
```

■ 接続先のOpenFlowコントローラのIPアドレスを設定

```
$ sudo ovs-vsctl set-controller s1 tcp:127.0.0.1:6633
```


Hands-On Part2

（課題）Mininetで作成した下図のようなNWトポロジーに対して、
OpenFlowスイッチ s1 にフローを登録することで、
h1 と h2 間でPingが疎通することを確認してください

- 留意事項
  - ✓ mininet_sample.pyがそのまま使える
  - ✓ Mininetを起動しているのとは別に新しくターミナルを起動し、Open vSwitchのコマンドで s1 にフローを登録
  - ✓ OpenFlowスイッチのコマンドでフローが登録されていることを確認
  - ✓ MininetのCLIコマンドを使用してPingの疎通性を確認
  - ✓ 確認がとれたらフローを削除してください

目次

Ryu

- Ryuとは
  - SDNコントローラを開発するためのPythonベースのフレームワーク
  - NTT研究所が開発し、オープンソースとして公開されている
  - OpenFlowをはじめ、様々なプロトコルに対応
    - ✓ OpenFlow、Netconf、OF-config、SNMP等

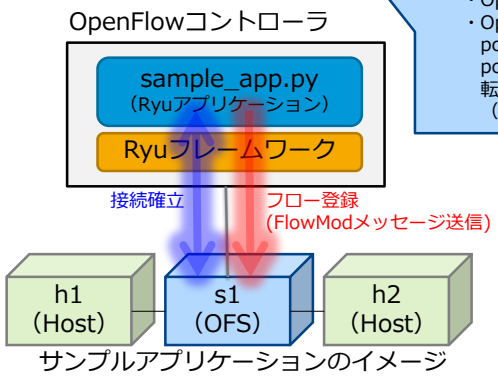
Ryuの起動方法

※ハンズオン環境（VM）には既にRyuをインストールしてあります

■ Ryuアプリケーションの起動

```
$ cd /home/mininet/handson/ryu-sample
$ ryu-manager sample_app.py
```

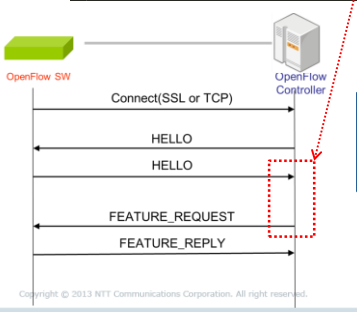
sample_app.pyの内容
・OpenFlowコントローラ(6633ポート)が起動
・OpenFlowスイッチと接続を確立すると、
port1からのARPパケットとIPパケットをport2へ
port2からのARPパケットをIPパケットをport1へ
転送するためのフローをスイッチに登録
（FlowModメッセージを送信）



サンプルアプリケーションのイメージ

（コード解説）Helloメッセージを受け取った際の処理

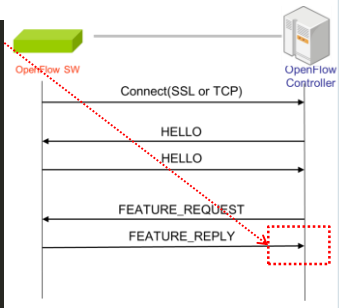
```
@set_ev_cls(ofp_event.EventOFPHello, HANDSHAKE_DISPATCHER)
def _handle_hello(self, ev):
    datapath = ev.msg.datapath
    print "[Hello received]"
    print "   * version: %d" % ev.msg.version
    if ev.msg.version not in [ofproto_v1_3.OFP_VERSION]:
        print '<WARN> Cannot connect with ofs. Unsupported version %d' % ev.msg.version
        # Send an error message.
        error_msg = datapath.ofproto_parser.OFPErrorMsg(datapath)
        error_msg.type = datapath.ofproto.OFPET_HELLO_FAILED
        error_msg.code = datapath.ofproto.OFPHFC_INCOMPATIBLE
        error_msg.data = 'unsupported version 0x%x' % ev.msg.version

        datapath.send_msg(error_msg)
        return
    else:
        datapath.set_version(ev.msg.version)
```



接続対象のスイッチの
OpenFlowバージョンを確認しています
（本セミナーではOpenFlow1.3を対象）

（コード解説）Featureメッセージを受け取った際の処理

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def _handle_features(self, ev):
    msg = ev.msg
    print '[SwitchFeatures received] '
    print "datapath_id=0x%016x" % msg.datapath_id
    print "n_buffers=%d" % msg.n_buffers
    print "n_tables=%d" % msg.n_tables
    print "auxiliary_id=%d" % msg.auxiliary_id
    print "capabilities=0x%08x" % msg.capabilities
    print

    datapath = msg.datapath
    datapath.id = msg.datapath_id
    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    set_config = ofp_parser.OFPSetConfig(
        datapath, ofp.OFPC_FRAG_NORMAL ,
        ofp.OFPCML_NO_BUFFER
    )
    datapath.send_msg(set_config)
    time.sleep(1)
    print "<INFO> OpenFlow Channel is established dpid=0x%016x" % datapath.id

    self.remove_flows(datapath,0)
    print "<INFO> delete flow entries dpid=0x%016x" % datapath.id

    self.add_output_flow(datapath, 1, 2, ether.ETH_TYPE_ARP, 0)
    self.add_output_flow(datapath, 2, 1, ether.ETH_TYPE_ARP, 0)
    self.add_output_flow(datapath, 1, 2, ether.ETH_TYPE_IP, 0)
    self.add_output_flow(datapath, 2, 1, ether.ETH_TYPE_IP, 0)
    print "<INFO> Set flow entries dpid=0x%016x" % datapath.id
```



スイッチのフィーチャを取得

スイッチのフローを初期化（削除）

フローの登録 (arp & ip)
“add_output_flow” Functionを使用
※Functionの内容は次ページ

スイッチとの接続確立後に
Flowの初期化とFlowの登録を実施
する処理を記述しています

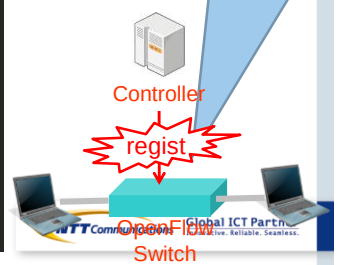
（コード解説）FlowModメッセージを送信するための処理

```
def add_output_flow(self, datapath, in_port, out_port, ether_type , ip_proto):
    # Match
    match = datapath.ofproto_parser.OFPMatch()
    match.set_in_port(in_port)
    match.set_dl_type(ether_type)

    if ip_proto != 0:
        match.set_ip_proto(ip_proto)

    ofp = datapath.ofproto
    ofp_parser = datapath.ofproto_parser

    # Action
    actions = [ofp_parser.OFPActionOutput(out_port, ofp.OFPCML_NO_BUFFER)]

    # Instruction
    inst = [ofp_parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS, actions)]

    # Flow Mod
    flowmod = datapath.ofproto_parser.OFPFlowMod(
        cookie=0,
        cookie_mask=0,
        flags=ofp.OFPFF_SEND_FLOW_REM,
        table_id=0,
        command=ofp.OFPFC_ADD,
        datapath=datapath,
        idle_timeout=0,
        hard_timeout=0,
        priority=0xff,
        buffer_id=OFP_NO_BUFFER,
        out_port=ofp.OFPP_ANY,
        out_group=ofp.OFPG_ANY,
        match=match,
        instructions=inst
    )

    # Send Message
    datapath.send_msg(flowmod)
```

登録するフローの内容を定義

Matchの内容
If inport = “in_port”
& ether type = “ether_type”

Instructionの内容
Then
output packet
from port “out_port”



（コード解説）PacketInを受け取った際の処理

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _handle_packet_in(self, ev):
    print "<INFO> handling_packet_in..."

    msg = ev.msg
    dp = msg.datapath
    ofproto_parser = dp.ofproto_parser
    ofproto = dp.ofproto

    # Find out in port.
    inport = -1
    match = msg.match
    for field in match.fields:
        if isinstance(field, ofproto_parser.MTInPort) or isinstance(field, ofproto_parser.MTInPhyPort):
            inport = field.value
            break

    if inport < 0:
        print "<WARN> Incoming port information not found for a packet in message from dpid: %s." % dp.id
        return

    # Find out ether type.
    packet = Packet(msg.data)
    eth = packet.next()
    ethertype = eth.ethertype

    while ethertype == ether.ETH_TYPE_8021Q:
        eth = packet.next()
        ethertype = eth.ethertype

    # Move back to 1st header.
    packet.protocol_idx = 0

    # Deprecated.
    self.handle_packetin(
        datapath=dp,
        inport=inport,
        packet=packet,
        table_id=msg.table_id,
        reason=msg.reason,
        total_len=msg.total_len
    )


def handle_packetin(self, datapath, inport, packet, table_id, reason, total_len):
    dpid = datapath.id
    efm = packet.next()
    #self.print_efm( efm )

    if efm.ethertype == ether.ETH_TYPE_ARP:
        arppkt = packet.next()
        #self.print_arp( arppkt )
    elif efm.ethertype == ether.ETH_TYPE_IP:
        ippkt = packet.next()
        #self.print_ip( ippkt )
```

OpenFlowスイッチから
コントローラに対してPacketIn
した際の処理を記述しています

NTT Communications Global ICT Partner Innovative. Reliable. Seamless.

Hands-On Part3

（課題）Mininetで作成した下図のようなNWトポロジーに対して、
OpenFlowスイッチ s1 にフローを登録することで、
h1 と h2 間でPingが疎通することを確認してください

- 留意事項
  - ✓ mininet_sample.pyがそのまま使える
  - ✓ Open vSwitchではなく、Ryuのサンプルアプリケーションを実行してフローをs1に登録
  - ✓ OpenFlowスイッチのコマンドでフローが登録されていることを確認
  - ✓ MininetのCLIコマンドを使用してPingの疎通性を確認
  - ✓ 確認がとれたらフローを削除

NTT Communications Global ICT Partner Innovative. Reliable. Seamless.

目次

ーー座学パートーー
- 第1章　はじめに
- 第2章　SDN/OpenFlowの動向
- 第3章　OpenFlowプロトコルの基礎

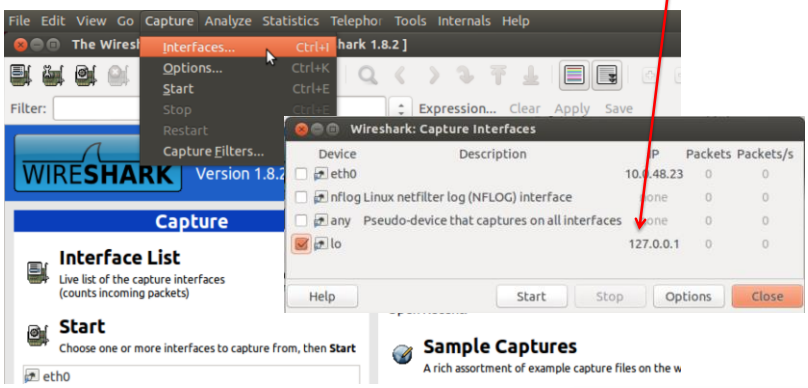
ーーハンズオンパートーー
- 第4章　ハンズオンの概要説明
- 第5章　Part1：Mininet(Open vSwitch)の基本操作
- 第6章　Part2：Open vSwitchの基本操作
- 第7章　Part3：Ryuの基本操作
- **第8章　Part4：OpenFlowメッセージのキャプチャ**
- 第9章　Part5：簡易Firewallアプリケーションの実装
- 第10章 Part6：応用問題

パケットキャプチャ(1)

- wiresharkの起動

```
$ wireshark
```

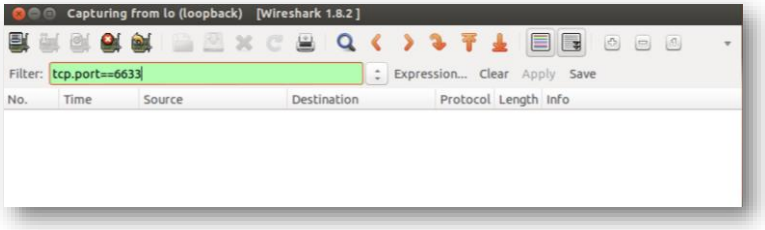
対象のインタフェースを選択



パケットキャプチャ(2)

■ Filterの例

TCPポート＝6633（OpenFlow）に指定しておくと、
表示がOpenFlowメッセージに絞られる



Hands-On Part4

（課題）OpenFlowスイッチとOpenFlowコントローラ間の
OpenFlowメッセージを確認してください

- 留意事項
  - ✓ Mininetで1台のOpenFlowスイッチを起動
  - ✓ RyuでOpenFlowコントローラ(sample_app.py)を起動
  - ✓ Wiresharkでメッセージをキャプチャ(コントローラ起動前から実施)
    - ・Helloメッセージ
    - ・Featureメッセージ
    - ・Echoメッセージ
    - ・FlowModメッセージ

      等を確認

目次

ー座学パートー
- 第1章　はじめに
- 第2章　SDN/OpenFlowの動向
- 第3章　OpenFlowプロトコルの基礎

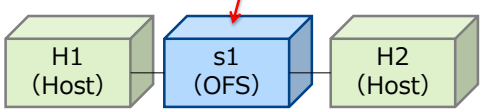
ーハンズオンパートー
- 第4章　ハンズオンの概要説明
- 第5章　Part1：Mininet(Open vSwitch)の基本操作
- 第6章　Part2：Open vSwitchの基本操作
- 第7章　Part3：Ryuの基本操作
- 第8章　Part4：OpenFlowメッセージのキャプチャ
- **第9章　Part5：簡易Firewallアプリケーションの実装**
- 第10章 Part6：応用問題

Hands-On Part5

簡易Firewallアプリケーションの実装

Firewall Policy

Traffic	Connection
Arp	Accept
ICMP (Ping)	Accept
Other (e.g. SSH)	Drop



sample_app.pyを改良して
ArpとICMPのパケットのみを転送する
アプリケーションを実装してください

ヒント

■マッチ条件を指定するためのサンプルコード

<inport> (例：port="1")
```
match.set_in_port(1)
```

<ether type> (例：ether type="IP")
```
match.set_dl_type(ether.ETH_TYPE_IP)
        or
match.set_dl_type(0x0800)
```

<ip protocol> (例：protocol="TCP")
```
match.set_ip_proto(inet.IPPROTO_TCP)
        or
match.set_ip_proto(6)
```

■ehter types
- ETH_TYPE_IP = 0x0800
- ETH_TYPE_ARP = 0x0806
- ETH_TYPE_802.1Q = 0x8100
- ETH_TYPE_IPV6 = 0x86dd
- ETH_TYPE_MPLS = 0x8847

etc...

■ip protocols
- IPPROTO_IP = 0
- IPPROTO_ICMP = 1
- IPPROTO_TCP = 6
- IPPROTO_UDP = 17
- IPPROTO_VRRP = 112

etc...

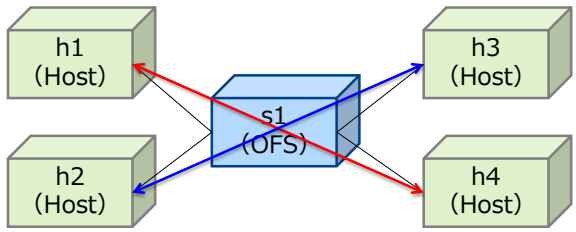
目次

ー座学パートー
- 第1章　はじめに
- 第2章　SDN/OpenFlowの動向
- 第3章　OpenFlowプロトコルの基礎

ーハンズオンパートー
- 第4章　ハンズオンの概要説明
- 第5章　Part1：Mininet(Open vSwitch)の基本操作
- 第6章　Part2：Open vSwitchの基本操作
- 第7章　Part3：Ryuの基本操作
- 第8章　Part4：OpenFlowメッセージのキャプチャ
- 第9章　Part5：簡易Firewallアプリケーションの実装
- **第10章 Part6：応用問題**

応用問題 1

- 以下の様なOpenFlowネットワークを構築してください

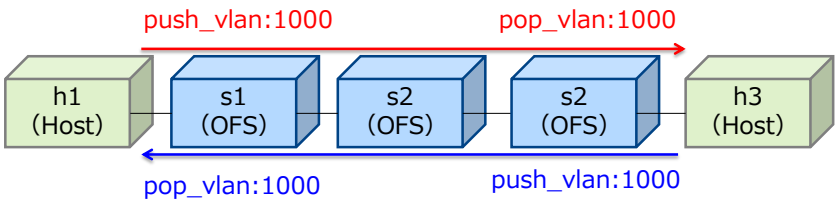


H1とH4同士は疎通可能
H2とH3同士は疎通可能
H1とH3 or H2は疎通不可
H2とH1 or H4は疎通不可

ポイント：
・ホストを識別するにはMacアドレスが使える？
・VLANやMPLSを使わなくても実現可能

応用問題2

■ 以下の様なOpenFlowネットワークを構築してください



ポイント：
- VLANのサンプルコードは次ページ
- 完成したらwiresharkでs2のインタフェースをキャプチャし、パケットにVLANがpushされていることを確認してください

参考：VLANに関わるサンプルコード

■vlan:1000でMatch条件を記述したいときのサンプル

```
match = datapath.ofproto_parser.OFPMatch()
match.set_vlan_vid(1000)
```

■vlan:1000をPushするInstructionを記述したいときのサンプル

```
vlan_field = datapath.ofproto_parser.OFPMatchField.make(ofp.OXM_OF_VLAN_VID, 1000)
actions = [ofp_parser.OFPActionPushVlan(ether.ETH_TYPE_8021Q),
           ofp_parser.OFPActionSetField(vlan_field)]
inst = [ofp_parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS,actions)]
```

■vlanをPOPするInstructionを記述したいときのサンプル

```
actions = [ofp_parser.OFPActionPopVlan(ether.ETH_TYPE_8021Q)]
inst = [ofp_parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS, actions)]
```

応用問題３

■ 自身の好きなOpenFlowネットワークを構築してください
- 分からない点等がございましたら講師にお尋ねください
  - ✓ PacketInのコードの記述方法
  - ✓ MPLSをPush・Popするコードの記述方法

等

お疲れ様でした

64

NTT Communications Global ICT Partner Innovative. Reliable. Seamless.

さいごに

- 前半の座学パートでは
  - SDN（Software Defined Networking）を実現する技術として着目されているOpenFlowの「動向」と「プロトコルの仕様」について紹介しました
- 後半のハンズオンパートでは
  - Ryu、Open vSwitch、Mininetを使って、実際に手を動かしながらOpenFlowネットワークを構築しました
- 今後、より詳しく勉強されたい方へ
  - Ryu、Open vSwitch、Mininetの使い方は、本セミナーで紹介したものが全てではありません（一例のみです）
  - 各オープンソースには公式WebページやWiki等、様々なリファレンスがありますので、参照していただければ幸いです
  - OpenFlowネットワークは本日のセミナーの環境のように1台のPC上で簡易に構築・試験が可能ですので、より詳しく勉強されたい方は、引き続き手を動かしていただければと思います

リファレンス

- OpenFlow Switch Specification(version 1.3)
  - https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf
- Ryu SDN Framework
  - http://osrg.github.io/ryu/
- Open vSwitch
  - http://openvswitch.org/
- Mininet
  - http://mininet.org/
- プログラミング言語 python
  - http://www.python.jp/