1 問題

複数スイッチ対応版（multi_learning_switch.rb）の動作解説

2 解答

2.1 コードの解説

multi_learning_switch.rb のメソッド毎にコメントとしてコード（1）の解説を追記する．また，コード中に追記された logger.info メソッドは動作確認のために挿入されたものである

ソースコード 1　multi_learning_switch.rb

```
$LOAD_PATH.unshift __dir__

require 'fdb'

class MultiLearningSwitch < Trema::Controller
  timer_event :age_fdbs, interval: 5.sec

# ======================================================================
# Trema の開始時に全ての FDB (Forwarding DB)を管理する連想配列を作成
# 開始メッセージを表示
# ======================================================================
  def start(_argv)
    @fdbs = {}
    logger.info 'MultiLearningSwitch started.'
  end

# ======================================================================
# 各スイッチ（lsw1-4）が接続されたらそれぞれについて FDB を作成
# 各 FDB の識別はスイッチの ID で行う
# ======================================================================
  def switch_ready(datapath_id)
    @fdbs[datapath_id] = FDB.new
  end

# ======================================================================
# PacketIn が発生したら呼ばれるメソッド
# 宛先が 802.1D 予約済みMACアドレスである場合は何もしない
# Packet In が発生したスイッチの FDB を取得し，
# (宛先アドレス，転送先ポート) =
# (Packet In の送信元アドレス，Packet In の受信ポート)を記録・更新する
# 最後に flow_mod_and_packet_out メソッドを実行
# ======================================================================
  def packet_in(datapath_id, message)
    return if message.destination_mac.reserved?
    logger.info "スイッチ #{datapath_id} にて Packet\_In が発生しました"
    @fdbs.fetch(datapath_id).learn(message.source_mac, message.in_port)
    flow_mod_and_packet_out message
```

```
  end

# ========================================================================
# 5 秒毎に実行されるメソッド
# 有効期限が切れた FDB のエントリがあれば削除する
# ========================================================================
  def age_fdbs
    @fdbs.each_value(&:age)
  end

  private

# ========================================================================
# PacketIn の宛先 Mac アドレスから FDB により転送先ポート番号を得る
# 転送先ポートが見つかれば flow_mod を実行
# さらに転送先ポートが見つかればそのポートからパケットを転送し，
# 転送先ポートが見つからなければフラッディングする（packet_out で処理）
# ========================================================================
  def flow_mod_and_packet_out(message)
    port_no = @fdbs.fetch(message.dpid).lookup(message.destination_mac)
    flow_mod(message, port_no) if port_no
    packet_out(message, port_no || :flood)
  end

# ========================================================================
# PacketIn の転送先ポートが見つかった場合は
# 今後同じパケットをスイッチのみで同様に処理できるよう
# send_flow_mod_add を実行する
# ========================================================================
  def flow_mod(message, port_no)
    send_flow_mod_add(
      message.datapath_id,
      match: ExactMatch.new(message),
      actions: SendOutPort.new(port_no)
    )
    logger.info "スイッチ #{message.datapath_id} の
                 FlowTable にルールを追加しました"
  end

# ========================================================================
# PacketIn を指定のポート（port_no）から送信するメソッド
# port_no に :flood が指定された場合はフラッディングする
# ========================================================================
  def packet_out(message, port_no)
    send_packet_out(
      message.datapath_id,
      packet_in: message,
      actions: SendOutPort.new(port_no)
    )
    logger.info "スイッチ #{message.datapath_id} の
                 ポート #{port_no} からパケットを送信しました"
  end
end
```

2.2 動作確認

以下の節の順にパケットを転送し，その際のログメッセージと動作概要を示す．trema.multi.conf に記述されているトポロジは図 1 の通りである．FDBn，FTn はそれぞれスイッチ n における FDB，FlowTable を意味する（FDB はコントローラ，FlowTable はスイッチで実装されている）．なお，パケットの送信については Trema の send_packets 機能，パケットの到達確認には show_stats 機能を用いた．

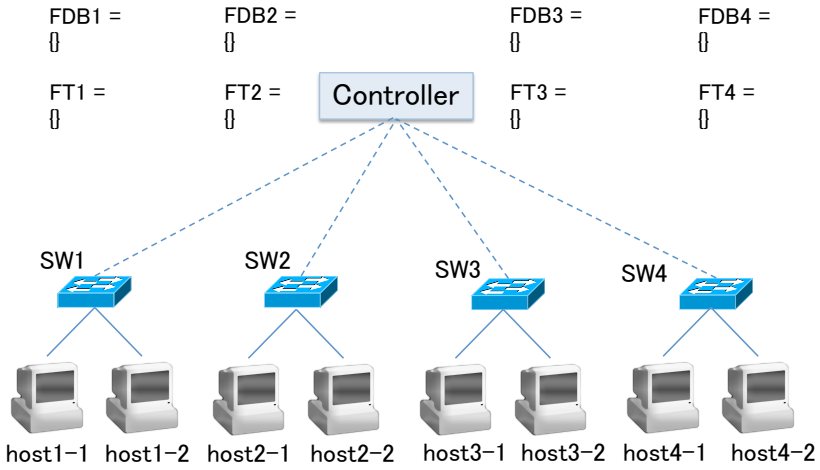

図 1　trema.multi.conf に定義されたトポロジ

2.2.1 (1) host1-1 から host1-2 へパケットを送る

ログメッセージ

> **スイッチ** `1` **にて** `Packet_In` **が発生しました**
> **スイッチ** `1` **のポート** `flood` **からパケットを送信しました**

動作概要

host1-1 から host1-2 宛てのパケットが送られると，SW1 にて Packet In が生じる．コントローラはパケットの送信元情報から SW1 の FDB に host1-1 宛てのエントリを生成し，そして host1-2 宛てのエントリは SW1 の FDB に登録されていないため，SW1 に当該パケットをフラッディングさせる．host1-2 は当該パケットが自身宛てのものであるため，受信する（図 2）．

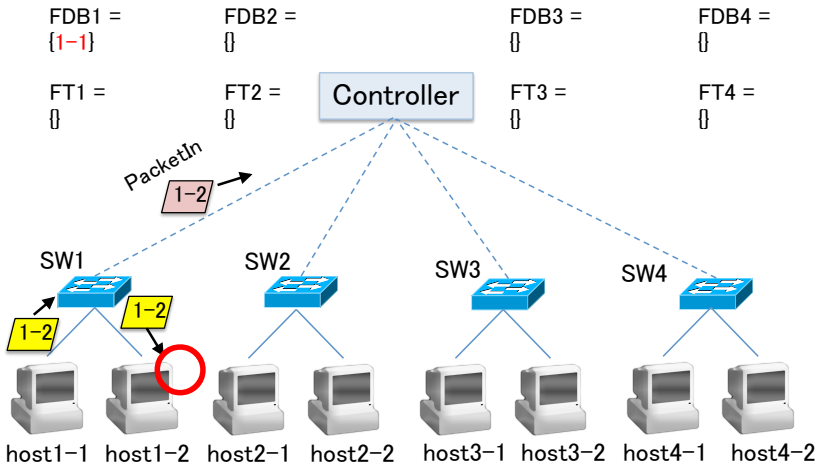

図 2　host1-1 から host1-2 へパケットを送る

2.2.2 (2) host1-2 から host1-1 へパケットを送る

ログメッセージ

```
スイッチ 1 にて Packet_In が発生しました
スイッチ 1 の FlowTable にルールを追加しました
スイッチ 1 のポート 1 からパケットを送信しました
```

動作概要

host1-2 から host1-1 宛てのパケットが送られると，SW1 にて Packet In が生じる．コントローラはパケットの送信元情報から SW1 の FDB に host1-2 宛てのエントリを生成し，そして既に host1-1 宛てのエントリが SW1 の FDB に登録されているため，SW1 に当該パケットを適切な転送先（host1-1 が接続されているポート）に転送させる．このとき，同様パケットを今後スイッチのみで転送処理するため SW1 の FlowTable に転送ルールを書き込む（図 3）．

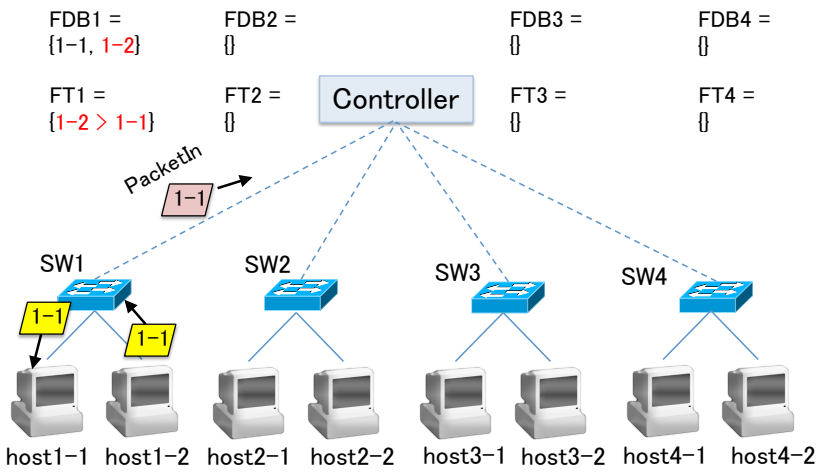

図 3　host2 から host1 へパケットを送る

2.2.3 (3) host1-2 から host1-1 へパケットを送る

ログメッセージ

（なし）

動作概要

SW1 の FlowTable に当該パケットの転送ルールが存在するため Packet In は生じない．したがってログメッセージは表示されず，SW1 によって当該パケットは適切に転送される（図 5）．

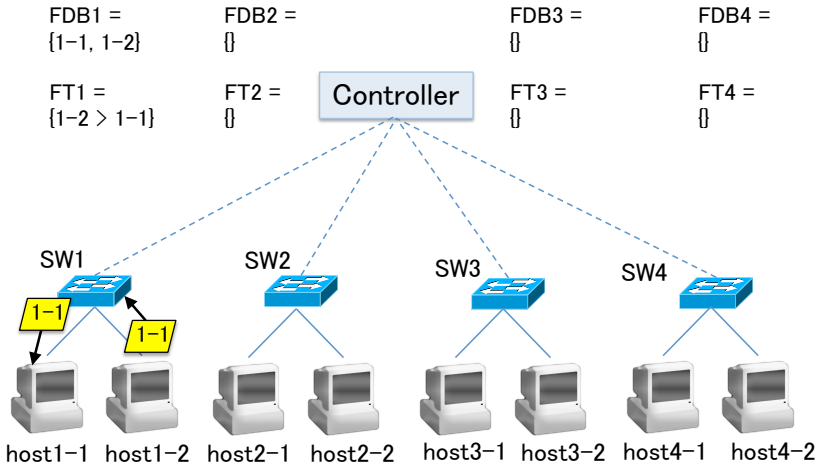

図 4　host1-2 から host1-1 へパケットを送る

2.2.4 (4) host2-1 から host2-2 へパケットを送る

ログメッセージ

```
スイッチ 2 にて Packet_In が発生しました
スイッチ 2 のポート flood からパケットを送信しました
```

動作概要

スイッチ，コントローラともに (1) と同様の動作をする（SW1 が SW2 に置き換わっただけである）．このことから，コントローラが複数のスイッチを独立に管理できていることがわかる．

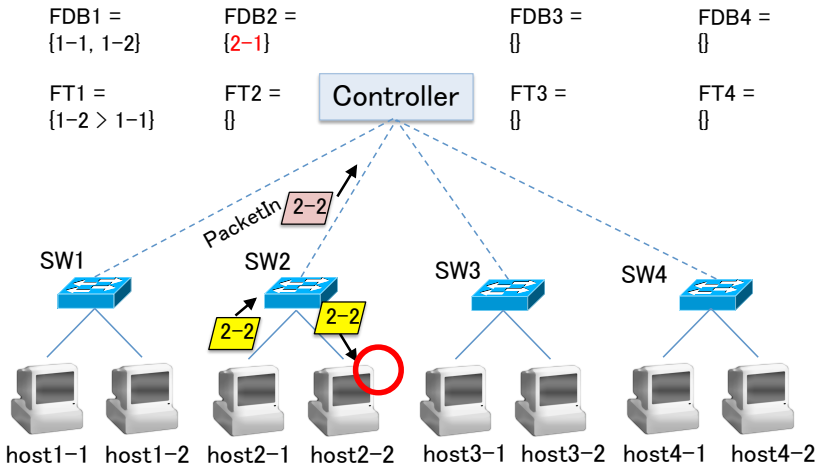

図 5　host2-1 から host2-2 へパケットを送る

2.2.5 (5) host3-1 から host1-1 へパケットを送る

ログメッセージ

> **スイッチ** 3 **にて** Packet_In **が発生しました**
> **スイッチ** 3 **のポート** flood **からパケットを送信しました**

動作概要

スイッチ，コントローラともに (1)(4) と同様の動作をする（スイッチが SW3 に置き換わっただけである）．ただし，SW3 にてフラッディングされた当該パケットは host3-2 および host1-1 にて受信しないことを確認した．これは，host3-2 が当該パケットが自身宛てでないため破棄したこと，host3-1 が host1-1 とネットワークで接続されていないことを意味する．

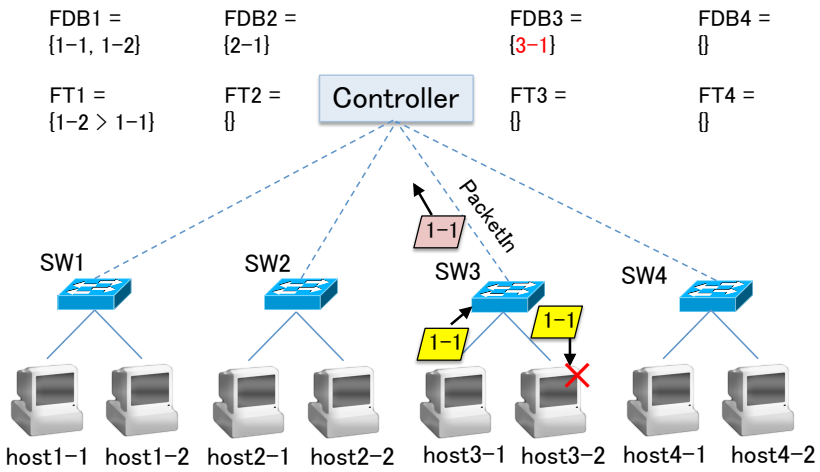

図 6　host3-1 から host1-1 へパケットを送る