2.1 OpenFlowはアカデミア出身

OpenFlow スイッチには「コントローラさえ書けばどんなネットワーク機器にも化けられる」という特長があります。乱暴に言うと、スイッチのような単純な機器はもちろん、ルータやロードバランサ、ファイアウォールや NAT など複雑な機器も、コントローラの実装をがんばれば実現できてしまいます。もちろん、専用機と同じ機能をすべて実装するのは大変ですし、機能の一部をソフトウェアとして実装することになるので実装が悪いと性能は落ちます。しかし、ソフトウェア次第で何でもできることに変わりはありません。

この何でもできるという特長は、もともとは大学や研究所などアカデミアからのニーズによって生まれたものでした。今までのスイッチやルータにとらわれない、まったく新しいインターネットを研究したい。でもすでにあるスイッチやルータのファームウェアを改造するのは大変だ。そうかといって一からハードウェアは作りたくないし……、大規模な仮想ネットワーク上で実験してもいいけれど、それだと実際のインターネット環境とあまりにも違いすぎる。こうしたジレンマを解消するために考え出されたのが OpenFlow だったわけです。

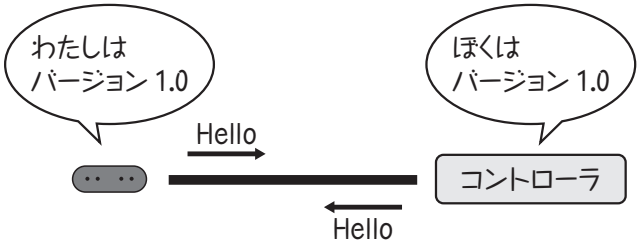
2.2 なぜOpenFlowが注目されているのか？

では、インターネットの研究用だった OpenFlow が巨大データセンターに代表される産業界でも注目されるようになったのはなぜでしょうか？

データセンターで最も重視されるのはスループット（単位時間あたりの処理能力）とコストです。Google などの巨大データセンターでは、世界中から届くたくさんのリクエストを短時間でさばくために（つまり、スループットを上げるために）物理的にたくさんのスイッチやサーバをそろえる必要があります。もし１台１台の価格が高いと、ハードウェア代だけでコストが膨大になってしまいます。そのため、秋葉原でも買えるコモディティ（普及品）から構成される安いハードウェアを使います。

バージョンのネゴシエーション

次に使用するOpenFlowプロトコルのバージョンを確認するステップ、いわゆるバージョンネゴシエーションが始まります。セキュアチャンネルを確立すると、スイッチとコントローラはお互いに自分のしゃべれるバージョン番号を乗せたHelloメッセージを出し合います（**図3-2**）。



▲図3-2　Helloメッセージを出し合うことで相手のOpenFlowプロトコルバージョンを確認

相手と同じバージョンを話せるようであればネゴシエーションに成功で、本格的におしゃべりを始められるようになります。

スイッチのスペックの確認

次にコントローラは接続したスイッチがどんなスイッチかを確認します。ネゴシエーション直後はまだバージョンしか確認できていないので、コントローラはスイッチにFeatures Requestメッセージを送って次の各情報をリクエストします（**図3-3**））。

- **スイッチのユニークID（Datapath IDと呼ぶ）**
- **スイッチポートの一覧情報**
- **サポートする機能の一覧**

ルータ A の次の転送先となるルータは、パケットの宛先ごとに異なります。たとえばホスト A からホスト C へパケットを送る場合には、ルータ A はそのパケットをルータ C へと転送します。

次の転送先へと正しくパケットを送るために、各ルータは、宛先と次の転送先の対応を記録したルーティングテーブルを持っています。たとえば、ルータ A のルーティングテーブルは、図 10-4 に示すようになります。

ここまでで、ルータの基本動作の説明はおしまいです。それでは、基本的なルータの機能を実装した、“ シンプルルータ ” のソースコードを読んでいきましょう。

10.4 SimpleRouter のソースコード

シンプルルータ（**SimpleRouter**）のソースコードは、いくつかのファイルからなります。紙面の都合上、以下ではメインのソースコード（**リスト 10.1**）を中心に説明します。残りのソースコードについては、Trema の **src/examples/simple_router/** 以下を参照してください。

▲リスト 10-1　シンプルルータ（simple-router.rb）のソースコード

```
require "arp-table"
require "interface"
require "router-utils"
require "routing-table"

class SimpleRouter < Controller
  include RouterUtils

  def start
    load "simple_router.conf"
    @interfaces = Interfaces.new( $interface )
    @arp_table = ARPTable.new
    @routing_table = RoutingTable.new( $route )
  end

  def packet_in( dpid, message )
    return if not to_me?( message )
```