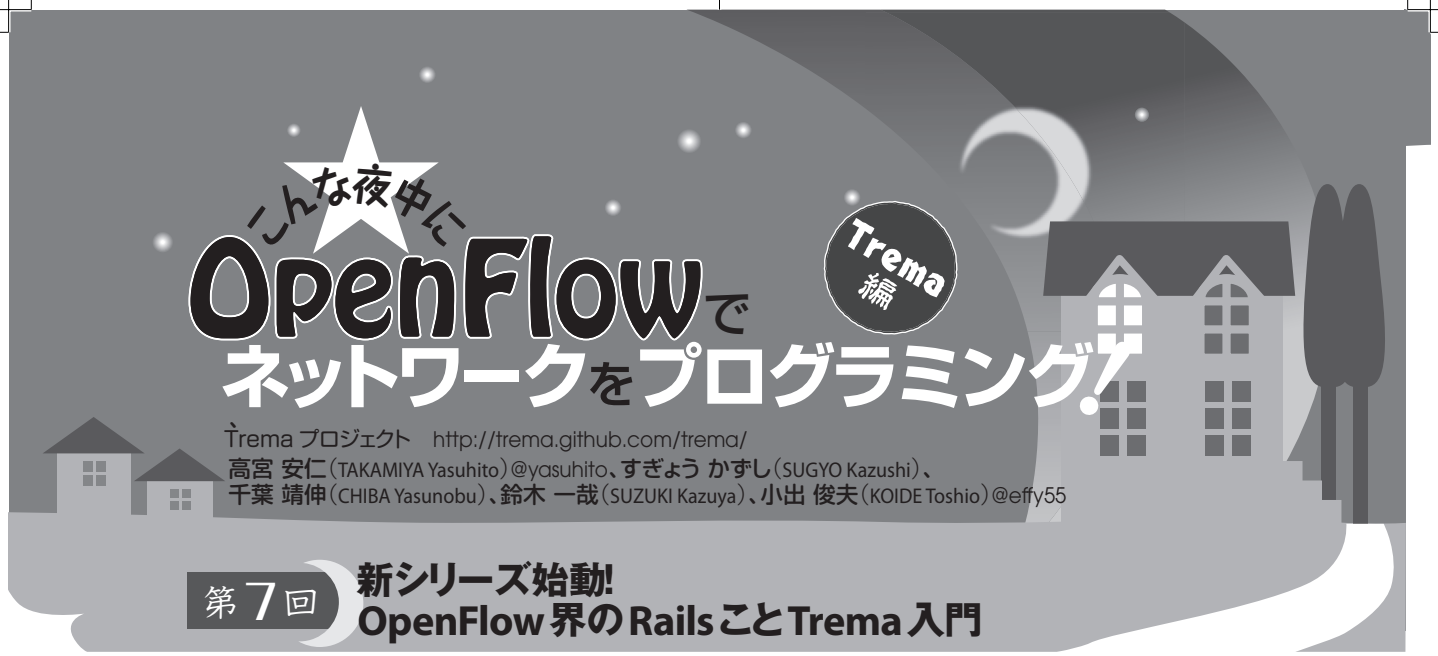


Tremaプロジェクト　http://trema.github.com/trema/
高宮 安仁（TAKAMIYA Yasuhito）@yasuhito、すぎょう かずし（SUGYO Kazushi）、
千葉 靖伸（CHIBA Yasunobu）、鈴木 一哉（SUZUKI Kazuya）、小出 俊夫（KOIDE Toshio）@effy55

第7回 新シリーズ始動! OpenFlow界のRailsことTrema入門

はじめに

効率とは賢く怠けることである
作者不詳

無精：エネルギーの総支出を減らすために、
多大な努力をするように、
あなたをかりたてる性質。
Larry Wall

優れたプログラマが持つハッカー気質のひとつに「無精」があります。大好きなコンピュータの前から一時も離れずにどうやってジャンクフードを手に入れるか――普通の人からするとただの横着に見えるかもしれませんが、ハッカー達にとってそれはいつでも大きな問題でした。たとえば、ハッカーの巣窟として有名なMITのAIラボにはかつて、UNIXのコマンド一発でピザをFAX注文するxpizzaコマンドが存在しました[注1]。また、RFC 2325として公開されているコーヒーポットプロトコルでは、遠隔地にあるコーヒーポットのコーヒーの量を監視したり、コーヒーを自動的に淹れたりするための半分冗談のインターフェースを定義しています。

こうした「ソフトウェアで楽をする」ハックのうち、もっとも大規模な例が最新鋭の巨大データセンターです。クラウドサービスの裏で動く巨大データセンターは極めて少人数の管理者によって運用されており、大部分の管理はソフトウェアによって極限まで自動化されているという記事を読んだことがある人も多いでしょう。ピザやコーヒーのようなお遊びから、巨大データセンターのように一筋縄ではいかない相手まで、プログラムで「モノ」を思いどおりにコントロールするのはもっとも楽しいハックの一種です。

OpenFlowの登場

その中でもネットワークをハックする技術の1つが、本連載で取り上げるOpenFlowです。OpenFlowはネットワークスイッチの内部動作を変更するプロトコルを定義しており[注2]、スイッチをコントロールするソフトウェア（OpenFlowの世界では**コントローラ**と呼ばれます）によってネットワーク全体をプログラム制御できる世界を目指しています（**図1**）。

OpenFlowの登場によって、今までは専門のオペレータによって管理されていたネットワークがついにプログラマ達にも開放されました。ネットワークをソフトウェアとして記述することにより、たとえば「アプリに合わせて勝手に最適化するネットワーク」や「障害が起こっても自

注1）「xpizza MIT」でググると、1991年当時のmanページが読めます。http://bit.ly/mYAJwZ

注2）OpenFlowの仕様書や標準化に関する情報は、http://www.openflow.org/で得られます。

己修復するネットワーク」といった究極の自動化も夢ではなくなります！

本連載では、このOpenFlowプロトコルを使ってネットワークを「ハック」する方法を数回に渡って紹介します。職場や自宅のような中小規模ネットワークでもすぐに試せる実用的なコードを通じて、「OpenFlowって具体的に何に使えるの？」というよくある疑問に答えていきます。OpenFlowやネットワークの基礎から説明しますので、ネットワークの専門家はもちろん、普通のプログラマもすんなり理解できると思います。

まずは、OpenFlowプログラミングのためのフレームワーク「Trema（トレマ）」を紹介しましょう。

OpenFlowプログラミングフレームワークTrema

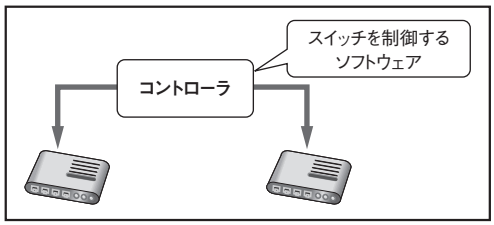
Tremaは、OpenFlowコントローラを開発するためのRubyおよびC用のプログラミングフレームワークです。ノートPC1台でアジャイルにOpenFlow開発をしたいなら、「OpenFlow界のRails」ことTremaで決まりです。GitHub上で開発されており、GPLv2ライセンスのフリーソフトウェアとして公開されています。公開は今年の4月と非常に新しいソフトウェアですが、その使いやすさから国内外の大学や企業および研究機関などですでに採用されています。

Tremaの情報は次のサイトから入手できます。

- Tremaホームページ：http://trema.github.com/trema/
- GitHubのページ：https://github.com/trema/
- メーリングリスト：https://groups.google.com/group/trema-dev
- Twitterアカウント：@trema_news

Tremaを使うと、ノートPC1台でOpenFlowコントローラの開発とテストができます。本連載では、実際にTremaを使っていろいろと実験しながらOpenFlowコントローラを作っていきます。それでは早速Tremaをセットアップして、簡単なプログラムを書いてみましょう。

★図1　OpenFlowスイッチとコントローラ



セットアップ

TremaはLinux上で動作し、Ubuntu 10.04以降およびDebian GNU/Linux 6.0の32ビットおよび64ビット版での動作が保証されています。テストはされていませんが、その他のLinuxディストリビューションでも基本的には動作するはずです。本連載では、Ubuntuの最新バージョンである11.04（デスクトップエディション32ビット版）を使います。

tremaコマンドの実行にはroot権限が必要です。まずは、sudoを使ってroot権限でコマンドを実行できるかどうか、sudoの設定ファイルを確認してください。

```
% sudo visudo
```

sudoができることを確認したら、Tremaが必要とするgccなどの外部ソフトウェアを次のようにインストールします。

```
% sudo apt-get install git gcc make ruby ↵
ruby-dev libpcap-dev libsqlite3-dev
```

次にTrema本体をダウンロードします。TremaはGitHub上で公開されており、gitを使って最新版が取得できます。

```
% git clone git://github.com/trema/trema.git
```

Tremaのセットアップには、「make install」のようなシステム全体へインストールする手順は不要です。ビルドするだけで使い始めることができます。ビルドは次のコマンドを実行するだけです。

```
% ./trema/build.rb
```

それでは早速、入門の定番Hello, Trema!コントローラをRubyで書いてみましょう。なお、本連載ではおもにTremaのRubyライブラリを使ったプログラミングを取り上げます。Cライブラリを使ったプログラミングの例については、Tremaのsrc/examples/ディレクトリ以下を参照してください。本連載で使ったRubyコードに加えて、同じ内容のCコードをみつけることができます。

Hello, Trema!

tremaディレクトリの中にhello_trema.rbというファイルを作成し、エディタで**リスト1**のコードを入力してください。

それでは早速実行してみましょう！ 作成したコントローラはtrema runコマンドで実行できます。この世界一短いOpenFlowコントローラ(？)は画面に「Hello, Trema!」と出力します。

```
% cd trema
% ./trema run ./hello_trema.rb
Hello, Trema!  ←―― Ctrl-cで終了
```

いかがでしょうか？ Tremaを使うと、とても簡単にコントローラを書いて実行できることがわかると思います。えっ？ これがいったいスイッチの何を制御したかって？ 確かにこのコントローラはほとんど何もしてくれませんが、Tremaでコントローラを書くのに必要な知識がひととおり含まれています。スイッチをつなげるのはちょっと辛抱して、まずはソースコードを見ていきましょう。

★リスト1　Hello Trema! コントローラ

```
class HelloController < Controller ――①
  def start ――――――――――――――――②
    puts "Hello, Trema!"
  end
end
```

コントローラクラスを定義する

Rubyで書く場合、すべてのコントローラはControllerクラスを継承して定義します(**リスト1**①)。

Controllerクラスを継承することで、コントローラに必要な基本機能がHelloControllerクラスにこっそりと追加されます。

ハンドラを定義する

Tremaはイベントドリブンなプログラミングモデルを採用しています。つまり、OpenFlowメッセージの到着など各種イベントに対応するハンドラを定義しておくと、イベントの発生時に対応するハンドラが呼び出されます。たとえばstartメソッドを定義しておくと、コントローラの起動時にこれが自動的に呼ばれます(**リスト1**②)。

さて、これでTremaの基本はおしまいです。次は、いよいよ実用的なOpenFlowコントローラを書いて実際にスイッチをつないでみます。今回のお題はスイッチのモニタリングツールです。「今、ネットワーク中にどのスイッチが動いているか」をリアルタイムに表示しますので、何らかの障害で落ちてしまったスイッチを発見するのに便利です。

スイッチモニタリングツールの概要

スイッチモニタリングツールは**図2**のように動作します。

OpenFlowスイッチは、起動するとOpenFlowコントローラへ接続しに行きます。Tremaでは、スイッチとの接続が確立すると、コントローラのswitch_readyハンドラが呼ばれます。コントローラはスイッチ一覧リストを更新し、新しく起動したスイッチをリストに追加します。逆にスイッチが何らかの原因で接続を切った場合、コントローラのswitch_disconnectedハンドラが呼ばれます。コントローラはリストを更新し、いなくなったスイッチをリストから削除します。

仮想ネットワーク

それでは早速、スイッチの起動を検知するコードを書いてみましょう。なんと、Tremaを使えばOpenFlowスイッチを持っていなくてもこうしたコードを実行してテストできます。いったいどういうことでしょうか？

その答えは、Tremaの強力な機能の1つ、仮想ネットワーク構築機能にあります。これは仮想OpenFlowスイッチや仮想ホストを接続した仮想ネットワークを作る機能です。この仮想ネットワークとコントローラを接続することによって、物理的なOpenFlowスイッチやホストを準備しなくとも、開発マシン1台でOpenFlowコントローラと動作環境を一度に用意して開発できます。もちろん、開発したコントローラは実際の物理的なOpenFlowスイッチやホストで構成されたネットワークでもそのまま動作します！それでは仮想スイッチを起動してみましょう。

仮想OpenFlowスイッチを起動する

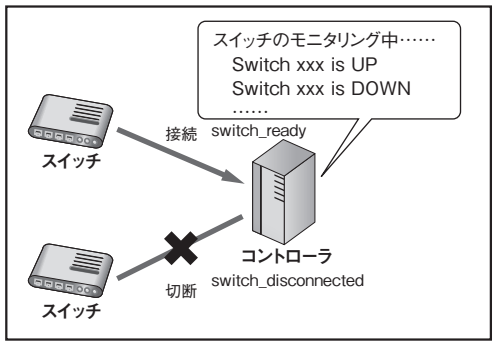
仮想スイッチを起動するには、仮想ネットワークの構成を記述した設定ファイルをtrema runに渡します。たとえば、**リスト2**の設定ファイルでは仮想スイッチ(vswitch)を2台定義しています。

それぞれに指定されているdatapath_id(0xabc、0xdef)はネットワークカードにおけるMACアドレスのような存在で、スイッチを一意に特定するIDとして使われます。OpenFlowの規格によると、64ビットの一意な整数値をOpenFlowスイッチ1台ごとに割り振ることになっています。仮想スイッチでは好きな値を設定できるので、かぶらないように適当な値をセットしてください。

スイッチの起動を捕捉する

それでは、さきほど定義したスイッチを起動してコントローラから捕捉してみましょう。スイッチの起動イベントを捕捉するにはswitch_readyハンドラを書きます(**リスト3**①)。

★図2　スイッチモニタリングツールの動作



@switchesは現在起動しているスイッチのリストを管理するインスタンス変数で、新しくスイッチが起動するとスイッチのdatapath_idが追加されます。また、putsメソッドでdatapath_idを表示します。

スイッチの切断を捕捉する

同様に、スイッチが落ちて接続が切れたイベントを捕捉してみましょう。このためのハンドラはswitch_disconnectedです(**リスト3**②)。

★リスト2　仮想ネットワークに仮想スイッチを2台追加

```
vswitch { datapath_id 0xabc }
vswitch { datapath_id 0xdef }
```

★リスト3　SwitchMonitorコントローラ

```
class SwitchMonitor < Controller
  periodic_timer_event :show_switches, 10 ――③

  def start
    @switches = []
  end

  def switch_ready datapath_id ――――――――――①
    @switches << datapath_id.to_hex
    info "Switch #{ datapath_id.to_hex } is UP"
  end

  def switch_disconnected datapath_id ――――②
    @switches -= [datapath_id.to_hex ]
    info "Switch #{ datapath_id.to_hex } is DOWN"
  end

  private ――――――――――――――――――――――――③
  def show_switches
    info "All switches = " + @switches.sort.join( ", " )
  end
end
```

スイッチの切断を捕捉すると、切断したスイッチのdatapath_idをスイッチ一覧@switchesから除きます。また、datapath_idをputsメソッドで表示します。

スイッチの一覧を表示する

最後に、スイッチの一覧を定期的に表示する部分を作ります。一定時間ごとに何らかの処理を行いたい場合には、タイマー機能を使います。**リスト3**③ように、一定の間隔で呼びたいメソッドと間隔(秒数)をperiodic_timer_eventで指定すると、指定されたメソッドが呼ばれます。ここでは、スイッチの一覧を表示するメソッドshow_switchesを10秒ごとに呼び出します。

実行

それでは早速実行してみましょう。仮想スイッチを3台起動する場合、**リスト4**の内容のファイルをswitch-monitor.confとして保存し、設定ファイルをtrema runの-cオプションに渡してください。実行結果は次のようになります。

```
% ./trema run ./switch-monitor.rb ↵
-c ./switch-monitor.conf
```

★リスト4　仮想スイッチを3台定義

```
vswitch { datapath_id 0x1 }
vswitch { datapath_id 0x2 }
vswitch { datapath_id 0x3 }
```

```
Switch 0x3 is UP
Switch 0x2 is UP
Switch 0x1 is UP
All switches = 0x1, 0x2, 0x3
All switches = 0x1, 0x2, 0x3
All switches = 0x1, 0x2, 0x3
……
```

switch-monitorコントローラが起動すると設定ファイルで定義した仮想スイッチ3台が起動し、switch-monitorコントローラのswitch_readyハンドラによって捕捉され、このメッセージが出力されました。

それでは、スイッチの切断がうまく検出されるか確かめてみましょう。スイッチを停止するコマンドはtrema killです。別ターミナルを開き、次のコマンドでスイッチ0x3を落としてみてください。

```
% ./trema kill 0x3
```

すると、trema runを動かしたターミナルに次の出力が表示されているはずです。

```
% ./trema run ./switch-monitor.rb ↵
-c ./switch-monitor.conf
Switch 0x3 is UP
Switch 0x2 is UP
Switch 0x1 is UP
All switches = 0x1, 0x2, 0x3
All switches = 0x1, 0x2, 0x3
All switches = 0x1, 0x2, 0x3
……
Switch 0x3 is DOWN
```

うまくいきました！ おわかりのとおり、この

COLUMN

友太郎(ゆうたろう)の質問　datapathってなに?

Q.「こんにちは！ 僕は最近OpenFlowに興味を持ったプログラマ、友太郎です。スイッチに付いているIDをdatapath IDって呼ぶのはわかったけど、いったいdatapathってなに？ スイッチのこと？」

A. 実用的には「datapath＝OpenFlowスイッチ」と考えて問題ありません。

「データパス」でググると、「CPUは演算処理を行うデータパスと、指示を出すコントローラから構成されます」というハードウェア教科書の記述がみつかります。つまり、ハードウェアの世界では一般に「筋肉にあたる部分＝データパス」「脳にあたる部分＝コントローラ」という分類をするようです。

OpenFlowの世界でも同じ用法が踏襲されています。OpenFlowのデータパスはパケット処理を行うスイッチを示し、その制御を行うソフトウェア部分をコントローラと呼びます。

メッセージはswitch_disconnectedハンドラによって表示されたものです。

まとめ

すべてのコントローラのテンプレートとなるHello, Trema!コントローラを書きました。また、これを改造してスイッチの動作状況を監視するスイッチモニタを作りました。学んだことは次の3つです。

- OpenFlowネットワークはパケットを処理するスイッチ(datapath)と、スイッチを制御するソフトウェア(コントローラ)から構成される。Tremaは、このコントローラを書くためのプログラミングフレームワークである
- Tremaは仮想ネットワーク構築機能を持っており、OpenFlowスイッチを持っていなくてもコントローラの開発やテストが可能。たとえば、仮想ネットワークに仮想スイッチを追加し、任意のdatapath IDを設定できる
- コントローラはRubyのControllerクラスを継承し、OpenFlowの各種イベントに対応するハンドラを定義することでスイッチをコントロールできる。たとえば、switch_readyとswitch_disconnectedハンドラでスイッチの起動と切断イベントに対応するアクションを書ける

次回はいよいよ本格的なコントローラとして、トラフィック集計機能のあるレイヤ2スイッチを作ります。初歩的なレイヤ2スイッチング機能と、誰がどのくらいネットワークトラフィックを発生させているかを集計する機能をOpenFlowで実現します。SD

COLUMN

友太郎(ゆうたろう)の質問 switch_readyってなに?

Q.「OpenFlowの仕様を読んでみたけど、どこにもswitch_readyって出てこなかったよ? OpenFlowにそんなイベントが定義されてるの?」

A. switch_readyはTrema独自のイベントで、スイッチがTremaに接続し指示が出せるようになった段階でコントローラに送られます。実は、switch_readyの裏では**図A**の一連の処理が行われており、TremaがOpenFlowプロトコルの詳細をうまくカーペットの裏に隠してくれているのです。

最初に、スイッチとコントローラがしゃべるOpenFlowプロトコルが合っているか確認します。OpenFlowのHELLOメッセージを使ってお互いのプロトコルバージョンを確認し、うまく会話できそうか確認します。

次は、スイッチを識別するためのdatapath IDの取得です。datapath IDのようなスイッチ固有の情報は、スイッチに対してOpenFlowのFeatures Requestメッセージを送ることで取得できます。成功した場合、datapath IDやポート数などの情報がFeatures Replyメッセージに乗ってやってきます。

最後にスイッチを初期化します。スイッチに以前の状態が残っていると、コントローラが管理する情報と競合が起こるため、初期化することでこれを避けます。これら一連の処理が終わると、ようやくswitch_readyがコントローラに通知されます。

★図A switch_readyイベントが起こるまで