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9回 テストファーストでアジャイルに

はじめに

初期ファイナル・ファンタジーの伝説的プログラマ、ナーシャ・ジベリの「早撃ち」エピソードを知っていますか？ 彼はヒーローのようにさっそうと現れ、どんなプログラムでも電光石火で書いてしまいます。「見てくれの悪さは気にしねえ。誰よりも早くやってやるぜ」。やがていくつかの伝説を残し、彼もプロジェクトを去るときがやってきました。残った同僚たちは困りました。彼の超絶プログラムは彼にしか理解できず、少しでもいじるとバグがあっても修正できないのです。それに、変更しようとすると動かなくなってしまいます。「ナーシャ、カムバック！」でも彼はもう戻ってきません……。

こうした悲劇を防ぐ方法の1つがソフトウェアテストです。OpenFlowコントローラのように動作シーケンスが複雑なソフトウェアが壊れていないことを確認するためには、ソフトウェアにより自動化されたテストが不可欠です。それに、きちんとしたテスト一式があればプログラム本体の理解もしやすく、修正も簡単です。

TremaはOpenFlowコントローラをテストするためのテストツールが充実しています。今回はこれを使って、簡単なコントローラ（リピータハブ）をテストファースト形式で実装していきます。

ではさっそく実際の例を見ていきましょう。

リピータハブの設計

まずは、リピータハブがどのように動作するかを説明しましょう。ホスト3台のネットワークを考えてください。あるホストからパケットを送信すると、リピータハブは入ってきたパケットを複製してほかのすべてのホストにばらまきます。

OpenFlowプロトコル的に何が起こっているかを**図1**に示します。host1がパケットを送信すると、スイッチからコントローラにpacket_inが起こります。ここでコントローラは「今後は同様のパケットをほかの全ポートへばらまけ（FLOOD）」というflow_modを打ちます。また、packet_inを起こした最初のパケットをほかのすべてのホスト（host2とhost3）に明示的にpacket_outで届けます。

最初のテスト

ではさっそく、リピータハブのテストコードを書いていきましょう。Tremaのテストフレームワークは Rubyのユニットテストツール RSpec（https://github.com/rspec）と統合されています。まだインストールしていない人は、「gem install rspec」でインストールしてください。また、TremaのAPIについては、http://rubydoc.info/github/trema/trema/を参照してください。

テストコードの最初のバージョンは**リスト1**のとおりです(spec/repeater-hub_spec.rb)。最初の行は、テストに必要なTremaのライブラリを読み込みます。describeで始まるdo…endブロックはテストの本体で、RepeaterHubコントローラのふるまいをここに記述(describe)する、という意味です。

RepeaterHubを定義していないのでエラーになることはわかりきっていますが、テストを実行してみましょう。次のコマンドを実行すると、Tremaを起動したうえでspec/repeater-hub_spec.rb(リスト1)のテストを実行します。

```
$ rspec -fs -c ./spec/repeater-hub_spec.rb
…/spec/repeater-hub_spec.rb:3:  ↵
uninitialized constant RepeaterHub (Name Error)
```

予想どおり、定数RepeaterHubが未定義というエラーで失敗しました。エラーを修正するために、RepeaterHubクラスの定義を追加してみましょう(**リスト2**)。本来、コントローラクラスは独立した.rbファイルに書きますが、今回は簡便さを優先し、テストコード上に直接書いているので注意してください。それでは実行してみましょう。今度はパスするはずです。

```
$ rspec -fs -c spec/repeater-hub_spec.rb
No examples found.
Finished in 0.00003 seconds 0 examples, ↵
0 failures
```

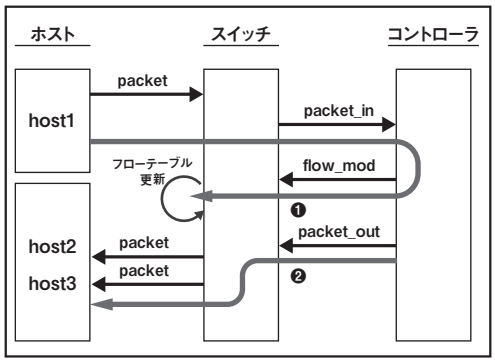
やった！ これで最初のテストにパスしました。

このようにテストファーストでは、最初にテストを書き、わざとエラーを起こしてからそれを直すためのコードをちょっとだけ追加します。テストを実行した結果からのフィードバックを得ながら「テスト書く、コード書く」を何度もくりかえしつつ最終的な完成形に近づけていくのです。

パケット受信のテスト

では、リピータハブの動作をテストコードにしていきましょう。どんなテストシナリオが思いつくでしょうか？ とりあえず、こんなのはどうでしょう。

★図1　ホスト3台をつなげたリピータハブの動作



「ホスト3台(host1、host2、host3)がスイッチにつながっているとき、宛先をhost2としたパケットをhost1が送ると、host2とhost3がパケットを受け取る。」

テストコードは**リスト3**のようにitブロックの中に記述します。テストシナリオをテストコードに置き換えるには、シナリオの各ステップをGiven(前提条件)、When(○○したとき)、Then(こうなる)の3つに分解するとうまく整理できます。

【Given】ホスト3つ(host1、host2、host3)がスイッチにつながっているとき、

★リスト1　リピータハブのテストのひな形

```
require File.join(File.dirname(__FILE__), "spec_helper")

describe RepeaterHub do
end
```

★リスト2　空のRepeaterHubクラスを追加してNameErrorを修正

```
require File.join(File.dirname(__FILE__), "spec_helper")

class RepeaterHub < Controller   ← 空のクラスを追加した
end

describe RepeaterHub do
end
```

【When】host1がhost2にパケットを送ると、
【Then】host2とhost3がパケットを受け取る

では、Given、When、Thenの順にテストコードを書いていきます。

【Given】ネットワークの構成

シナリオの前提条件(Given)として、テストを実行するホスト3台のネットワーク構成(図1)を**リスト4**のように定義します。これはネットワーク設定とまったく同じ文法ですね！ ここで、それぞれの仮想ホストがpromiscオプション(自分宛でないパケットを受け取る)を"on"にしていることに注意してください。リピータハブではパケットがすべてのポートにばらまかれるので、こうすることでどこからのパケットでも受信できるようにしておきます。

【When】パケットの送信

Whenは「○○したとき」というきっかけになる動作を記述します。ここでは、Givenで定義されたホストhost1からhost2にパケットを送ります。パケットを送るコマンドは、trema send_packetsでした。テストコード中でもこれに似たAPIを使うことができます(**リスト5**)。

run(RepeaterHub)は、Givenで定義されたネットワークの上でRepeaterHubコントローラを動かし、続くブロックを実行するという意味です。

【Then】受信パケット数のテスト

Thenには「最終的にこうなるはず」というテストを書きます。ここでは、「host2とhost3にパケットが1つずつ届くはず」を書けばよいですね(**リスト6**)。

vhost("*ホスト名*")は仮想ホストにアクセスするためのメソッドで、仮想ホストの受信パケットなどさまざまなデータを見ることができます。ここでは、受信したパケットの数、つまり受信パケットカウンタstats(:rx)が1ということをテストしています。

テストを実行

ではさっそく実行してみましょう。

```
Failure/Error: vhost("host2").stats( ア
:rx).should have( 1 ).packets
    expected 1 packets, got 0
```

失敗しました。「host2はパケットを1つ受信するはずが、0個だった」というエラーです。RepeaterHubの中身をまだ実装していないので当たり前ですね。すぐにはなおせそうにないので、ひとまずこのテストは保留(pending)とし、あとで復活することにしましょう(**リスト7**)。

今度は実行結果が次のように変わり、エラーが出なくなります。

```
Pending:
  1) は、入ってきたパケットを他のすべての ポートに転送する
  # あとで実装する
```

ここでの失敗の原因は、いきなりすべてを実装しようとしたことでした。以降では、リピータハブの動作を図1の❶と❷の2段階に分け、1つずつテストファーストで実装していくことにしましょう。

★リスト3　テストシナリオの定義

```
describe RepeaterHub do
  it "は、入ってきたパケットを他のすべてのポートに転送する" do
    (テストコードをここに書く)
  end
end
```

★リスト4　テストシナリオを実行するネットワーク構成の定義(Given)

```
describe RepeaterHub do
  it "は、入ってきたパケットを他のすべてのポートに転送する" do
    network {                                    ← ホスト3台、スイッチ1台のネットワーク
      vswitch("switch") { dpid "0xabc" }
      vhost("host1") { promisc "on" }            ← 自分宛でないパケットも受け取る
      vhost("host2") { promisc "on" }
      vhost("host3") { promisc "on" }
      link "switch", "host1"
      link "switch", "host2"
      link "switch", "host3"
    }
  end
end
```

フローエントリのテスト

まずは、スイッチにフローエントリができることをテストしてみましょう(図1の❶)。テストシナリオは次のようになります。

【Given】ホスト3つ(host1、host2、host3)がスイッチにつながっているとき、
【When】host1がhost2にパケットを送ると、
【Then】パケットをばらまくフローエントリをスイッチに追加する

では、これをテストコードにしてみましょう。GivenとWhenは最初のテストシナリオと同じで、Thenだけが異なります。パケットをばらまく処理はFLOODですので、**リスト8**のようになります。

ネットワーク構成のコード(network { …… }の部分)をコピペしてしまっていますが、あとできれいにするので気にしないでください。エラーになることを見越しつつ、さっそく実行すると、次のエラーになります。

```
Failure/Error: switch("switch").should ➥
have(1).flows
     expected 1 flows, got 0
```

「スイッチにフローエントリが1つあるはずがなかった」というエラーです。では、flow_modを打ち込むコードをRepeaterHubクラスに追加して、もう一度テストしてみましょう(**リスト9**)。

```
Failure/Error: switch("switch").flows. ➥
first.actions.should == "FLOOD"
     expected: "FLOOD"
          got: "drop" (using ==)
```

別のエラーになりました。「アクションが"FLOOD"でなく"drop"だった」と怒られています。たしかに、さきほどのflow_modにはアクションを設定していなかったので、デフォルトのアクションであるdrop(パケットを破棄する)になってしまっています。flow_modにパ

★リスト5　テストパケットを送信(When)

```
describe RepeaterHub do
  it "は、入ってきたパケットを他のすべてのポートに転送する" do
    network {
      (省略)
    }.run(RepeaterHub) {
      send_packets "host1", "host2"
    }
  end
end
```

★リスト6　受信パケット数のテスト

```
describe RepeaterHub do
  it "は、入ってきたパケットを他のすべてのポートに転送する" do
    network {
      (省略)
    }.run(RepeaterHub) {
      send_packets "host1", "host2"

      vhost("host2").stats(:rx).should have(1).packets
      vhost("host3").stats(:rx).should have(1).packets
    }
  end
end
```

★リスト7　すぐに修正できないテストを保留(pending)にする

```
describe RepeaterHub do
  it "は、入ってきたパケットを他のすべてのポートに転送する" do
    pending "あとで実装する"   ← この行を追加する
    network {
      (省略)
```

★リスト8　フローエントリのテスト

```
it "は、パケットをばらまくフローエントリをスイッチに追加する" do
  network {
    (省略)
  }.run(RepeaterHub) {
    send_packets "host1", "host2"
    switch("switch").should have(1).flows
    switch("switch").flows.first.actions.should == "FLOOD"
  }
end
```

★リスト9　flow_modをスイッチに打ち込む

```
class RepeaterHub < Controller
  def packet_in dpid, message
    send_flow_mod_add dpid
  end
end
```

ケットをばらまくアクションを定義してみましょう(**リスト10**)。

今度はテストが通りました! それでは、もう少しThenを詳細化し、フローの特徴を細かくテストしてみましょう(**リスト11**)。

ここではホストにIPアドレスを振り、フローのsrcとdstがこのアドレスに正しく設定されているかをチェックしています。実行してみましょう。

```
Failure/Error: flow.nw_src. ↵
should == "192.168.0.1"
     expected: "192.168.0.1"
          got: nil (using ==)
```

失敗しました。フローのsrcには、パケット送信元であるhost1のIPアドレス192.168.0.1がセットされるべきですが、何もセットされていません。では、flow_modで:matchを指定して、この値がセットされるようにします(**リスト12**)。

テストにパスしました! これで、フローエントリが正しくスイッチに書き込まれていることまで(図1の❶)をテストできました。

テストコードのリファクタリング

テストが通ったので、最後にコードの重複部分をまとめておきましょう。同じnetwork{ …… }が重複しているので、aroundブロックを使って1箇所にまとめます(**リスト13**)。

再びパケット受信のテスト

いよいよ完成間近です。パケットがhost2とhost3に届くことをテストします(図1の❷)。最初のテストの保留マーク(pending)を消して、再び実行してみましょう。

```
Failure/Error: vhost("host2").stats(:rx).
should have( 1 ).packets
    expected 1 packets, got 0
```

失敗してしまいました。host2がパケットを受信できていません。そういえば、flow_modしただけではパケットは送信されないので、明示的にpacket_outしてやらないといけないのでしたね(**リスト14**)。さっそく実行してみましょう。

```
RepeaterHub
  は、入ってきたパケットを他のすべてのポートに転送する
  は、パケットをばらまくフローエントリをスイッチに追加する

Finished in 15.66 seconds
2 examples, 0 failures
```

すべてのテストに通りました! これでリピータハブとテストコード一式が完成です。

★リスト10　flow_modにアクションを追加

```
class RepeaterHub < Controller
  def packet_in dpid, message
    send_flow_mod_add(
      dpid,
      :actions => ActionOutput.new( OFPP_FLOOD )
    )
  end
end
```

★リスト11　フローのsrcとdstのテストを追加

```
describe RepeaterHub do
  it "は、入ってきたパケットを他のすべてのポートに転送する" do
    network {
      vswitch("switch") { dpid "0xabc" }
      vhost("host1") { promisc "on"; ip "192.168.0.1" }
      vhost("host2") { promisc "on"; ip "192.168.0.2" }
      vhost("host3") { promisc "on"; ip "192.168.0.3" }
      link "switch", "host1"
      link "switch", "host2"
      link "switch", "host3"
    }.run(RepeaterHub) {
      send_packets "host1", "host2"
      switch("switch").should have(1).flows
      flow = switch("switch").flows.first
      flow.actions.should == "FLOOD"
      flow.nw_src.should == "192.168.0.1"
      flow.nw_dst.should == "192.168.0.2"
    }
  end
end
```

まとめ

Tremaのユニットテストフレームワークを使ってリピータハブを作りました。Tremaのsrc/examplesディレクトリの下にはテストコードのサンプルがいくつかありますので、本格的にテストコードを書く人は参考にしてください[注1]。今回学んだことは次の2つです。

- コントローラをユニットテストする方法を学びました。TremaはRubyのユニットテストフレームワークRSpecと統合されており、仮想スイッチのフローテーブルや仮想ホストの受信パケット数などについてのテストを書けます。
- テストをGiven、When、Thenの3ステップに分けて分析／設計する方法を学びました。それぞれのステップをRSpecのテストコードに置き換えることで、テストコードが完成します。

次回はTremaプロジェクト入門と題して、開発の舞台裏やメンバーの紹介、またTremaに付属するサンプルアプリを解説します。Tremaに加わりたい人や、参考になるソースコードを探している人に役立つ情報となる予定です。

最後に、OpenFlowプログラミングコンテストのお知らせです。Interop Tokyo 2012において、ソフトウェアルータの実装コンテストである「オープンルータ・コンペティション(http://www.interop.jp/2011/orc/)」が開催されます。Tremaで書いたコントローラを試す良い機会ですので、腕に覚えのある方はぜひ参加してみてください！SD

★リスト12　flow_modメッセージに:matchをセット

```
class RepeaterHub < Controller
  def packet_in dpid, message
    send_flow_mod_add(
      dpid,
      :match => ExactMatch.from(message),
      :actions => ActionOutput.new( OFPT_FLOOD )
    )
  end
end
```

★リスト13　共通部分をaroundブロックに移すことでコードの重複をなくす

```
describe RepeaterHub do
  around do |example|
    network {
      (省略)
    }.run(RepeaterHub) {
    example.run ←――― それぞれのitブロックをここで実行
    }
  end

  it "は、入ってきたパケットを他のすべてのポートに転送する" do
    send_packets "host1", "host2"
    pending "あとで実装する"
      ……
  end

  it "は、パケットをばらまくフローエントリをスイッチに追加する" do
    send_packets "host1", "host2"
    switch("switch").should have(1).flows
      ……
  end
end
```

★リスト14　RepeaterHubにpacket_out処理を追加

```
class RepeaterHub < Controller
  def packet_in dpid, message
    send_flow_mod_add(
      dpid,
      :match => ExactMatch.from(message),
      :actions => ActionOutput.new(OFPT_FLOOD)
    )
    send_packet_out(
      dpid,
      :packet_in => message,
      :actions => ActionOutput.new(OFPT_FLOOD)
    )
  end
end
```

注1）リピータハブの完全なテストコードは、Tremaのsrc/examples/repeater_hub/ディレクトリにあります。