Trema Tutorial

Test-first OpenFlow programming with Trema

Yasuhito TAKAMIYA

@yasuhito

7/29/2011

My personal CV

Many years of exeperience in HPC and middleware (Satoshi MATSUOKA Lab. @ Tokyo Tech)

Keywords: MPI, Cluster, Grid, Cloud, Super Computing, Top500, TSUBAME @ Tokyo Tech

Only a few months of experience in OpenFlow (Trema)

Interests: Programming systems, agile and software testing in Ruby and C

Today's Goal

Introduction to Trema with hands-on session

How to develop in Trema

Designing, testing and debugging using Trema framework

"Different from NOX, Beacon and others?"

Why Trema?

... because we write it in C and Ruby

(NOX written in C++ and Python, Beacon written in Java)

This is the main reason!

Difficulties of OpenFlow development

Hard to setup execution environments

(Lots of hardware switches, hosts, and cables...)

Development environment in a box? (e.g., mininet)

=> Trema offers a similar emulation environment

Network emulation

Emulated execution environment in your laptop made by:

virtual switches: Open vSwitch

virtual hosts: phost (pseudo host)

virtual links: vlink (ip command of Linux)

You can construct your own topology using Ruby DSL (Domain Specific Language)

Network DSL

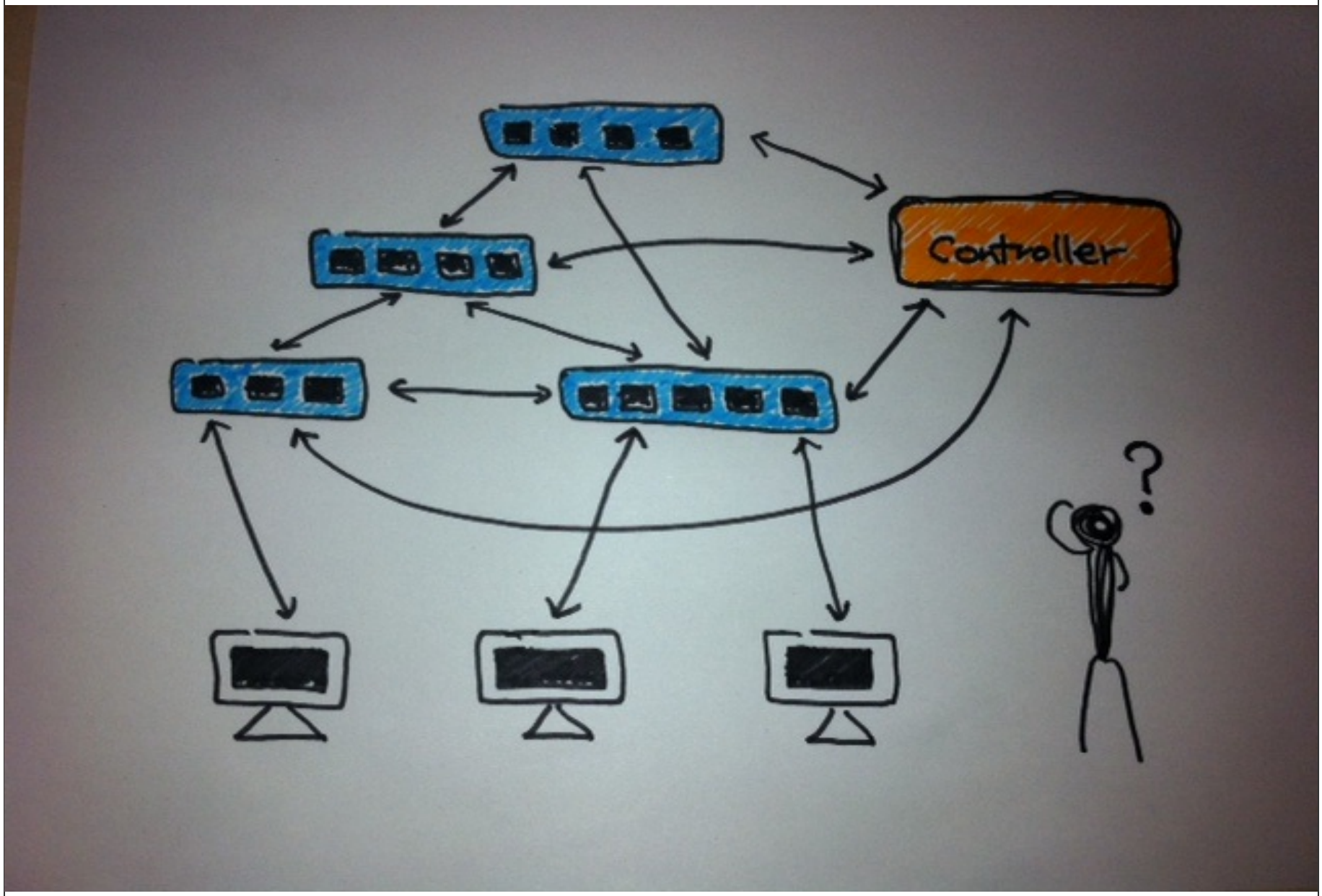
```
# virtual switches
vswitch("switch1") { datapath_id "0x1" }
vswitch("switch2") { datapath_id "0x2" }
vswitch("switch3") { datapath_id "0x3" }
vswitch("switch4") { datapath_id "0x4" }

# virtual hosts
vhost("host1")
vhost("host2")
vhost("host3")
vhost("host4")

# virtual links
link "switch1", "host1"
link "switch2", "host2"
link "switch3", "host3"
link "switch4", "host4"
link "switch1", "switch2"
link "switch2", "switch3"
link "switch3", "switch4"
```

Another difficulty of OpenFlow

"OpenFlow programming

==

Distributed programming"


Controller

OpenFlow == Distributed programming

Lots of switches, hosts, and links where

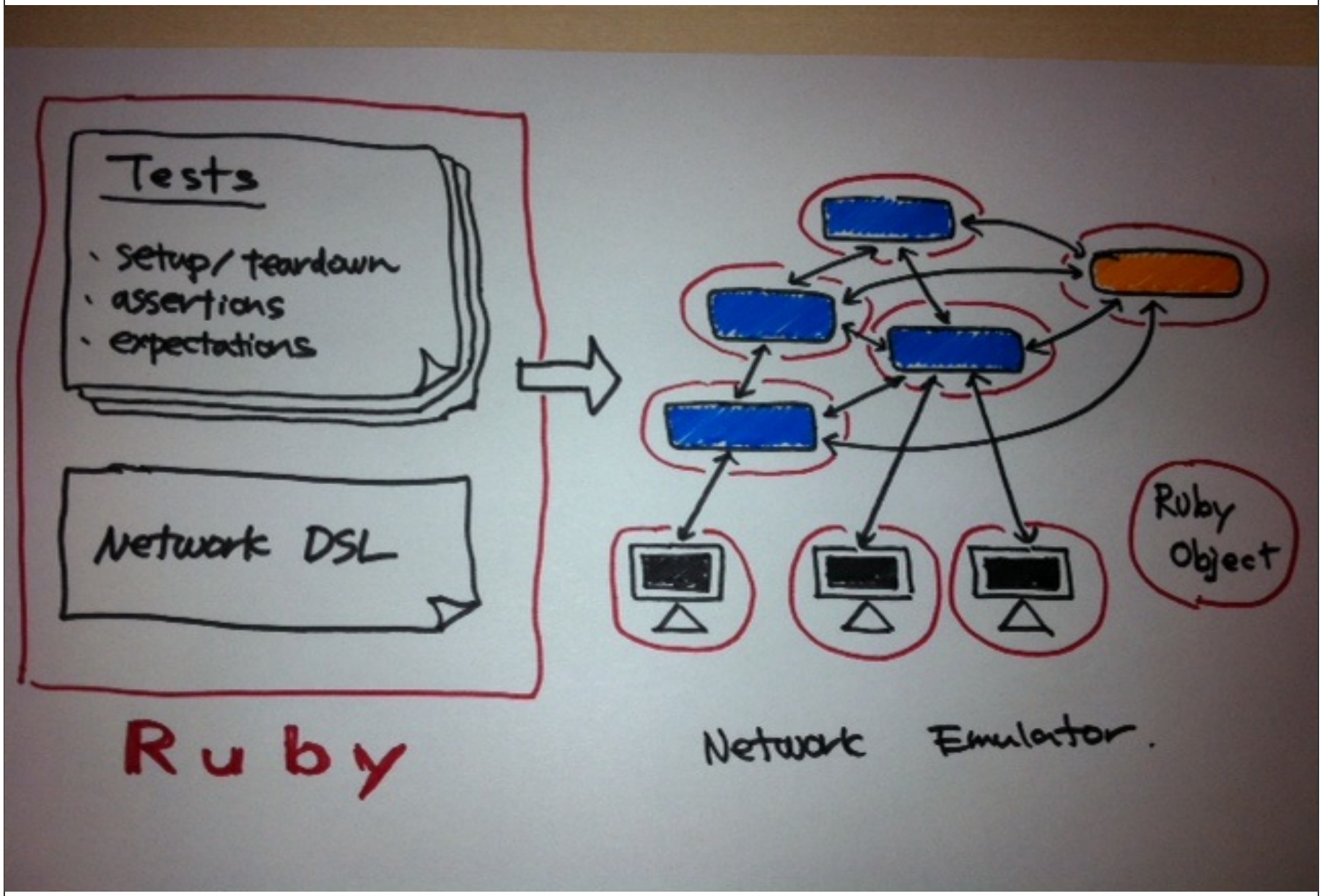
each runs in its own memory space (in a separate hardware or a process),

while changing its own state (flow table, stats etc.),

and communicating intricately with each other

=> Need an aid from programming frameworks and tools!

Trema test framework


Tests
· setup/teardown
· assertions
· expectations
Network DSL
Ruby
Ruby Object
Network Emulator.

Test framework (Ruby only)

Write network environment and controller, test using Ruby

Setup and teardown of network environment

Assertions and expectations over switches, hosts and your controller

Fault injection such as intentional link-down, latencies and packet-drops etc.

Test code example

```
# Test example: A unittest of MyController controller
#
#   The following tests that controller's packet_in handler
#   is invoked when a packet is arrived.

network { # Setup test environment
  vswitch("switch") { datapath_id "0xabc" }

  vhost("host1")
  vhost("host2")

  link "switch", "host1"
  link "switch", "host2"
}.run(MyController) { # Run tests
  # Expectation over the controller
  controller.should_receive(:packet_in)

  # Send a test packet
  send_packets "host1", "host2"
}
```

Summary

The integration of network emulation and test framework using Ruby enables developers to apply "well-known" testing techniques such as mocks, stubs and expectations to OpenFlow programming

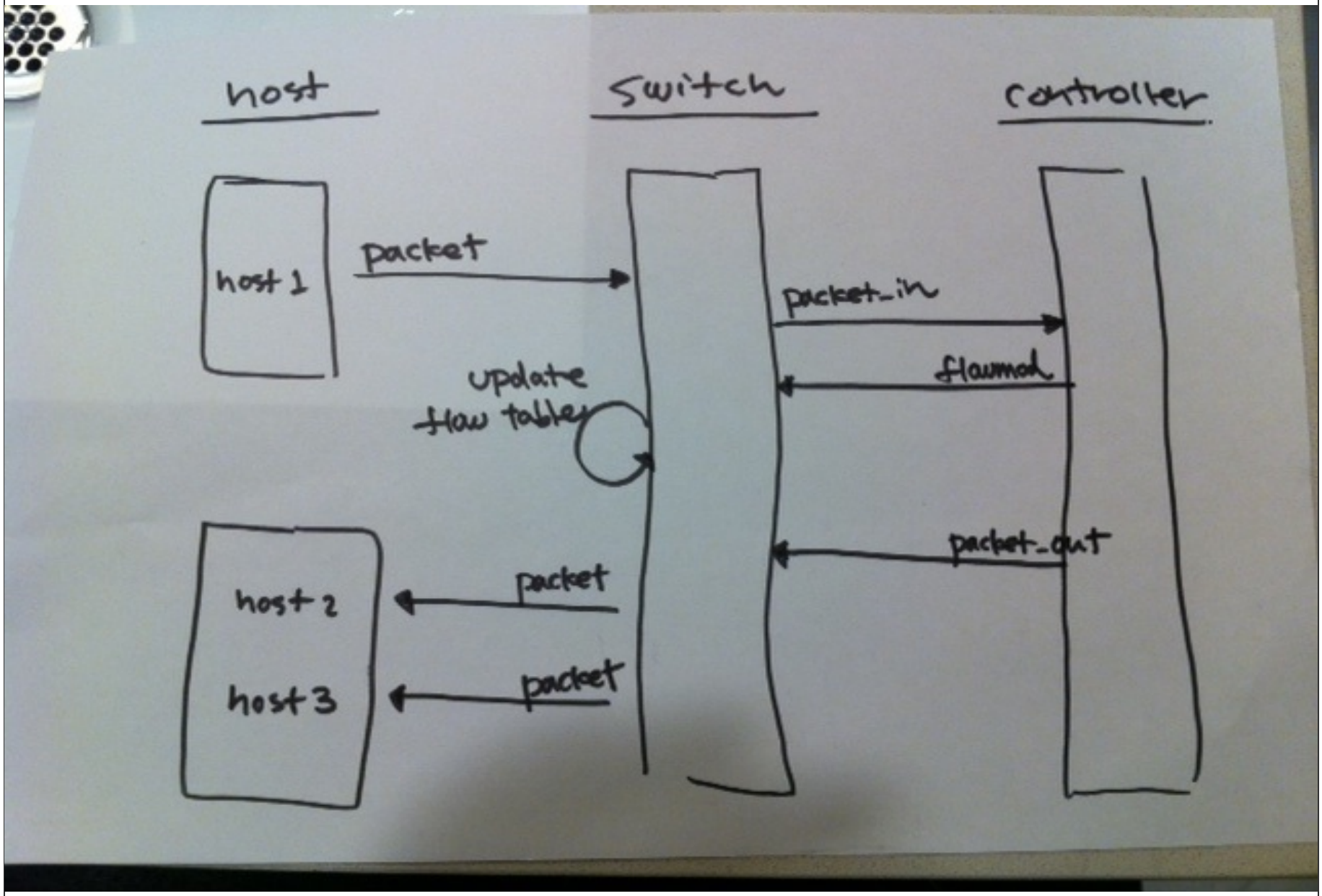
Setup Trema

```
$ git clone git://github.com/trema/trema
$ ./trema/build.rb
  (There is no Step Three!!)
```

Design phase

The theme for this tutorial is "repeater-hub"

By designing, testing and debugging repeater-hub with Trema, let's explore Trema framework.


host
switch
controller
host 1
packet
packet_in
flowmod
update
flow table
packet_out
host 2
packet
host 3
packet

Analysis

In order to test repeater-hub program, we need one switch and at least three hosts

=> How can we build a test environment?

The sequence looks surprisingly complex despite the simplicity of repeater-hub functionality (== flooding)

=> How can we test each arrow in the diagram?

Trema framework

Network DSL

Build emulation envrionment in your laptop

Test framework

Describe and run unittests of each arrow in execution sequence

Trema Ruby library

Write human-readable DSL and tests briefly and seamlessly


WELL, GOSH, I SUPPOSE I COULD TRY IT.
I ALREADY FEEL AN IMPROVEMENT IN MY QUALITY OF LIFE.
www.dilbert.com

Iteration #1

"Defining RepeaterHub class"

The first step

```
# ./spec/repeater-hub_spec.rb


# load helper libraries for testing
require File.join(File.dirname(__FILE__), "spec_helper")


describe RepeaterHub do
  # Write the spec description of repeater hub here
  # The spec is executed as unittest
end
```

NOTE: syntactic details are explained later

Test First

```
$ rspec -fs -c spec/repeater-hub_spec.rb
/home/yasuhito/play/trema/spec/repeater-hub_spec.rb:4: uninitialized constant RepeaterHub
(NameError)
from /var/lib/gems/1.8/gems/rspec-core-2.6.3/lib/rspec/core/configuration.rb:419:in
`load'
from /var/lib/gems/1.8/gems/rspec-core-2.6.3/lib/rspec/core/configuration.rb:419:in
`load_spec_files'
from /var/lib/gems/1.8/gems/rspec-core-2.6.3/lib/rspec/core/configuration.rb:419:in `map'
from /var/lib/gems/1.8/gems/rspec-core-2.6.3/lib/rspec/core/configuration.rb:419:in
`load_spec_files'
from /var/lib/gems/1.8/gems/rspec-core-2.6.3/lib/rspec/core/command_line.rb:18:in `run'
  ...


#=> FAIL ("RepeaterHub" is now known)
```

No problem, because we did't implement the class yet.

Let's add some code just enough to pass the test.

Changes to pass the test

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

# Add an empty class definition
class RepeaterHub
end

describe RepeaterHub do
end
```

Test again!

```
$ rspec -fs -c spec/repeater-hub_spec.rb
No examples found.

Finished in 0.00003 seconds
0 examples, 0 failures


#=> SUCCESS
```

We got the template of both RepeaterHub class and its test

Successfully completed iteration #1

RSpec

```
# The spec of Car class
describe Car do
  car = Car.new
  car.should respond_to(:run)
  car.should respond_to(:stop)
  car.should have(4).wheels
end
```

The de facto standard for unit test framework for Ruby

Used in Rails and other well-known products

Human-readable test DSL and its output (explained later)

Why Test First?

OpenFlow programming is complicated, because it's a sort of distributed programming

Unit-testing is helpful expecially for such a complicated problem

Rubyists love tests and are used to it well

=> Trema offers the OpenFlow extension of RSpec (explained later)

Iteration #2

"Flood incoming packets to every other port"

Let's write down unit tests

You can describe the flooding feature of RepeaterHub as follows:

```
describe RepeaterHub do
  it "should flood incoming packets to every other port"
end
```

"it" == an instance of RepeaterHub

It is still just a placeholder, because the body of this feature is not written yet.

Run

```
$ rspec -fs -c spec/repeater-hub_spec.rb

RepeaterHub
  should flood incoming packets to every other port (PENDING: Not Yet Implemented)

Pending:
  RepeaterHub should flood incoming packets to every other port
    # Not Yet Implemented
    # ./spec/repeater-hub_spec.rb:9

Finished in 0.00024 seconds
1 example, 0 failures, 1 pending


=> PENDING (Not Yet Implemented)
```

It (RSpec)

```
describe "Hello Trema" do
  it 'should be a String' do
    "Hello Trema".should be_a(String)
  end

  it 'should not == "Hello Frinfon"' do
    "Hello Trema".should_not == "Hello Frinfon"
  end
end

# =>
#   Hello Trema
#     should be a String
#     should not equal to "Hello Frinfon"
```

Each "it" corresponds to a feature

Test codes within the "it" block

You can get a human-readable spec output by running RSpec scripts

Breakdown

"Controller should flood incoming packets to every other port"

==

Given: one switch, and three hosts are connected to it

When: Host #1 sends a packet to host #2

Then: Host #2 and #3 should receive the packet

Given

```
describe RepeaterHub do
  it "should flood incoming packets to every other port" do

    # ************** Given **************
    network {
      # A switch
      vswitch("switch") { dpid "0xabc" }

      # Three hosts
      vhost("host1") { promisc "on" }
      vhost("host2") { promisc "on" }
      vhost("host3") { promisc "on" }

      # Connect these hosts to the switch
      link "switch", "host1"
      link "switch", "host2"
      link "switch", "host3"
    }
  end
end
```

Note that the syntax is fully compatible with Trema network DSL

Network DSL for RSpec

```
#
# Describe test environment in network { ... } block
#
network {
  # Virtual switches
  vswitch("name") { options }

  # Virtual hosts
  vhost("name") { options }

  # Virtual links
  link "peer#1", "peer#2"
}
```

Given, **When**

```
describe RepeaterHub do
  it "should flood incoming packets to every other port" do

    # ************** Given **************
    network {
      ...

    # ************** When **************
    }.run(RepeaterHub) { # Run a RepeaterHub in the Given network
      # Host #1 sends a packet to host #2
      vhost("host1").send_packet "host2"
    }
  end
end
```

"When" API

```
network {
  # ...
}.run(ControllerClass) {
  # vswitch("name").method
  # vhost("name").method
  # link("peer1", "peer2").method
}
```

Components defined in the network block (vswitch, vhost and link) are wrapped as Ruby objects in the "When" block.

In the "When" block, you can invoke any method of these wrapped objects

"When" Example

```
# Send 1,000 packets from host1 to host2
vhost("host1").send_packet "host2", :n_pkts => 1000

# Send packets from host1 to host2 for 5 seconds with pps = 10
vhost("host1").send_packet "host2", :pps => 10, :duration => 5

# (Other options are also available "./trema help send_packets")


# Fault injection
link("host1","host2").down
link("host1","host2").up

# Etc, etc...
```

Test result

```
$ rspec -fs -c spec/repeater-hub_spec.rb

RepeaterHub
  should flood incoming packets to every other port (FAILED - 1)

Failures:

  1) RepeaterHub should flood incoming packets to every other port
     Failure/Error: network {
     RuntimeError:
       RepeaterHub is not a subclass of Trema::Controller
     # ./spec/repeater-hub_spec.rb:11

Finished in 0.07034 seconds
1 example, 1 failure


=> FAIL (RepeaterHub is not a subclass of Trema::Controller)
```

Quick fix

```
# Inherit from Trema::Controller class
class RepeaterHub < Trema::Controller
end


describe RepeaterHub do
  it "should flood incoming packets to every other port" do
    network {
      # ...
    }.run(RepeaterHub) {
      # ...
    }
  end
end


#=> SUCCESS
```

Given, When, **Then**

```
describe RepeaterHub do
  it "should flood incoming packets to every other port" do

    # ************** Given **************
    network {
      # ...

    # ************** When **************
    }.run(RepeaterHub) {
      vhost("host1").send_packet "host2"

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

Run

```
$ rspec -fs -c spec/repeater-hub_spec.rb

RepeaterHub
  should flood incoming packets to every other port (FAILED - 1)

Failures:

  1) RepeaterHub should flood incoming packets to every other port
     Failure/Error: vhost("host2").stats(:rx).should have( 1 ).packets
       expected 1 packets, got 0
     # ./spec/repeater-hub_spec.rb:24

Finished in 4.18 seconds
1 example, 1 failure


=> FAIL (expected 1 packets, got 0)
```

Matchers

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

# vs.

vhost("host2").stats(:rx).packets.size.should == 1
```

Matchers (Error Message)

```
vhost("host2").stats(:rx).should have(1).packets
#=> expected 1 packets, got 0


# vs.


vhost("host2").stats(:rx).packets.size.should == 1
#=> expected: 1
#        got: 0 (using ==)
```

... Stuck?

Let's divide into smaller tests and implement one-by-one

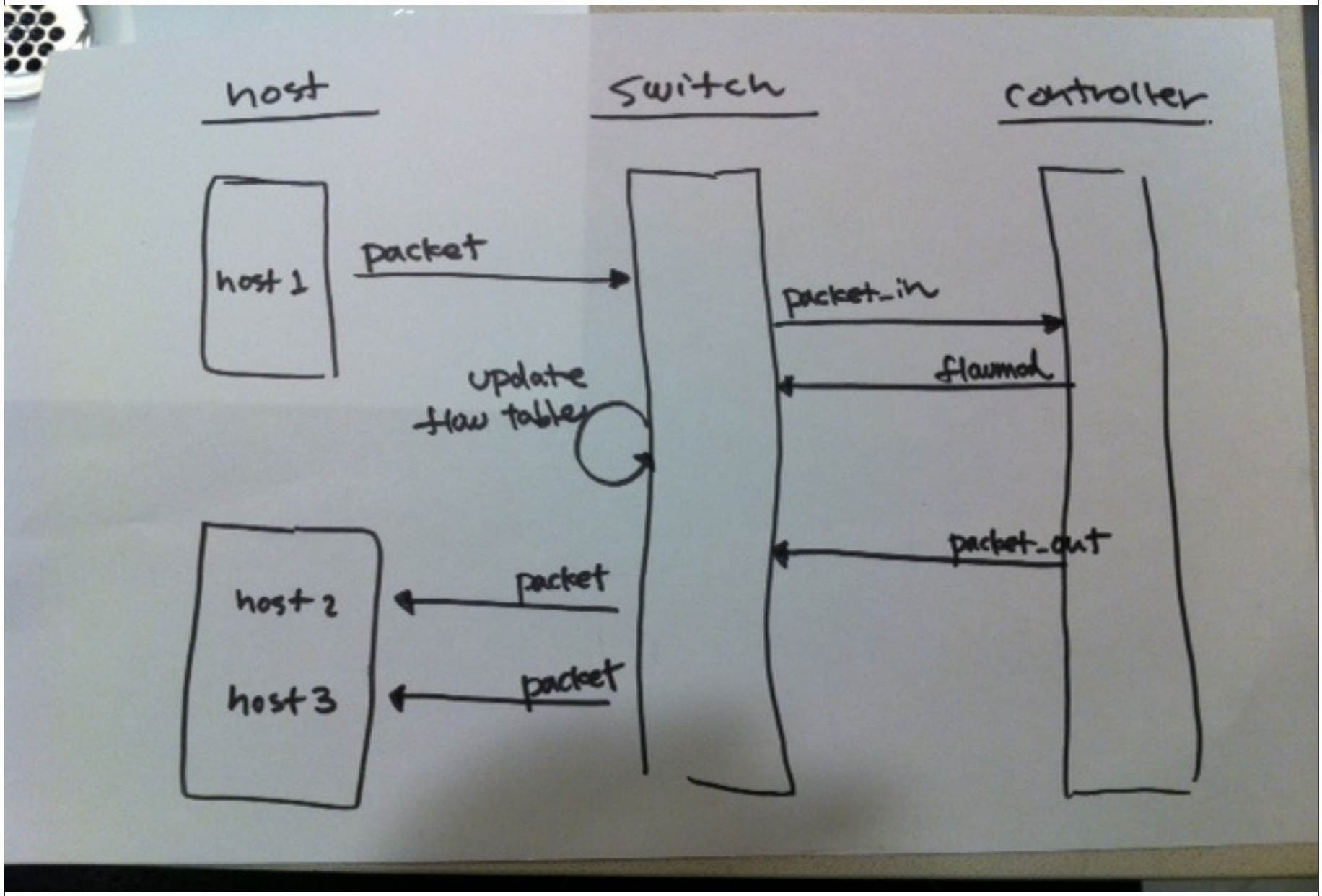
For now, mark this test as "pending" and give it the least priority

Pending

```
describe RepeaterHub do
  it "should flood incoming packets to every other port" do
    network {
      # ...
    }.run(RepeaterHub) {
      send_packets "host1", "host2"

      # mark as pending
      pending("Implement later")

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

Run

```
$ rspec -fs -c spec/repeater-hub_spec.rb

RepeaterHub
  should flood incoming packets to every other port (PENDING: あとで実装する)

Pending:
  RepeaterHub should flood incoming packets to every other port
    # Implement later
    # ./spec/repeater-hub_spec.rb:10

Finished in 3.99 seconds
1 example, 0 failures, 1 pending
```


host
switch
controller
host 1
packet
packet_in
flowmod
update
flow table
packet_out
host 2
packet
host 3
packet

Iteration #3

"Packet-in"

TODO

- packet_in (switch → controller)
- flow_mod (controller → switch)
- Update flow table. (switch)
- packet_out (controller → switch)

Breakdown

"RepeaterHub should receive a packet_in message"

==

Given: one switch, and three hosts are connected to it

When: Host #1 sends a packet to host #2

Then: the packet_in message should be delivered to the controller

Expectation

```
describe RepeaterHub do
  it "should receive a packet_in message" do
    network {
      # ...
    }.run(RepeaterHub) {
      # Expectation:
      #  packet_in message from 0xabc should be delivered to
      #  the controller only once
      controller("RepeaterHub").should_receive(:packet_in).with do |dpid, m|
        dpid.should == 0xabc
      end

      send_packets "host1", "host2"
    }
  end
end

# => SUCCESS
```

Message handlers

Controller#packet_in(datapath_id, message)

Controller#flow_removed(datapath_id, message)

Controller#switch_disconnected(datapath_id)

Controller#port_status(datapath_id, message)

Controller#stats_reply(datapath_id, message)

Controller#openflow_error(datapath_id, message)

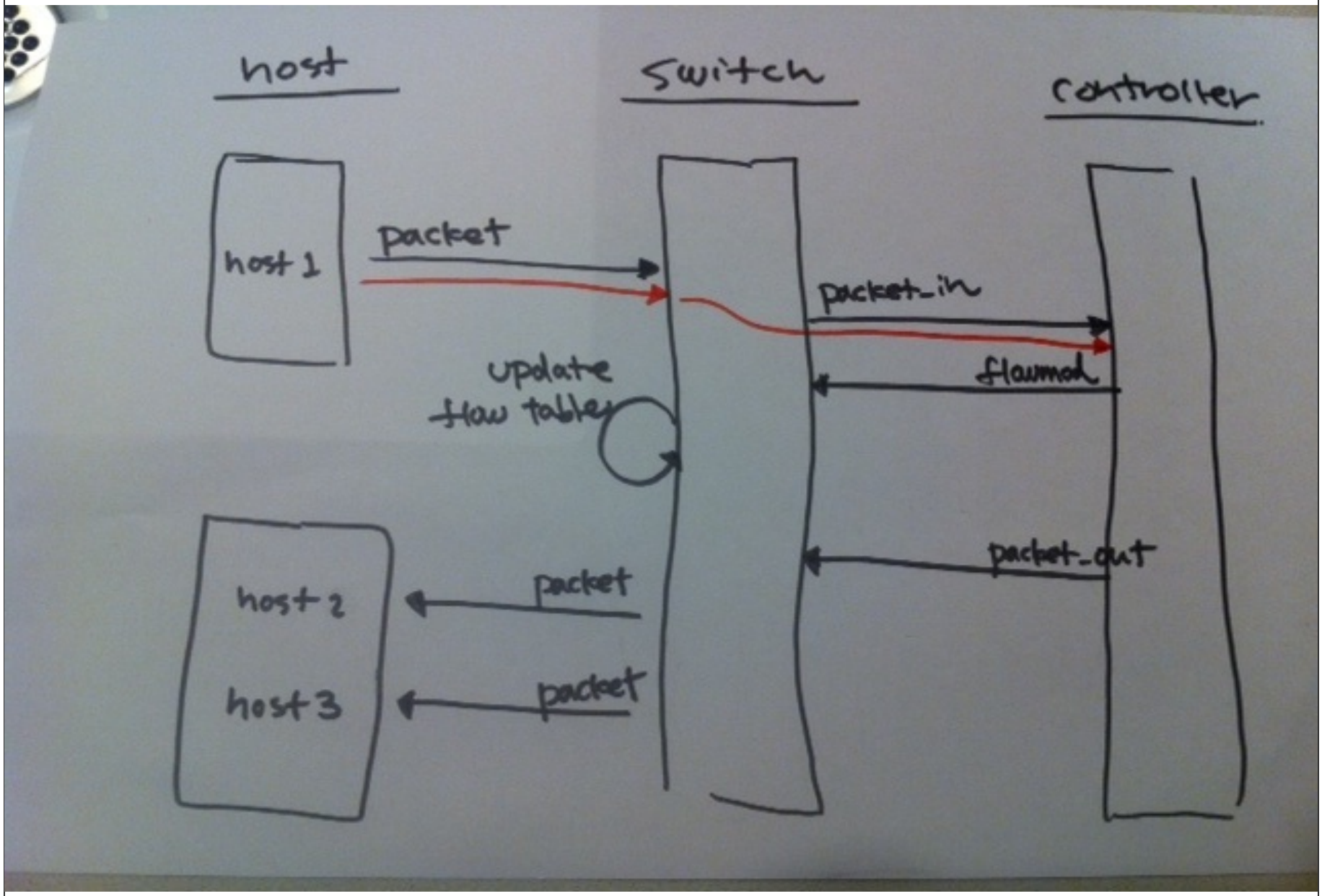
(See src/examples/dumper.rb for full list)

Don't Repeat Yourself

```
describe RepeaterHub do
  # common setup here
  around do |example|  # `example' is binded to each "it" block
    network {
      ...
    }.run(RepeaterHub) {
      example.run  # run "it" block
    }
  end

  it "should #packet_in" do
    controller("RepeaterHub").should_receive(:packet_in).with do |m, dpid|
      dpid.should == 0xabc
    end

    send_packets "host1", "host2"
  end

  it "should flood incoming packets to every other port" do
    send_packets "host1", "host2"

    pending("Implement later")
    vhost("host2").stats(:rx).should have(1).packets
    vhost("host3").stats(:rx).should have(1).packets
  end
end

# => SUCCESS
```

TODO

- ~~packet_in (switch → controller)~~
- flow_mod (controller → switch)
- Update flow table. (switch)
- packet_out (controller → switch)


host
switch
controller
host 1
packet
packet_in
flowmod
update
flow table
packet_out
host 2
packet
host 3
packet

Iteration #4

"Flow-mod"

TODO

- ~~packet_in (switch → controller)~~
- flow_mod (controller → switch)
- Update flow table. (switch)
- packet_out (controller → switch)

Breakdown

"Controller should send a flow_mod message"

==

Given: one switch, and three hosts are connected to it

When: Host #1 sends a packet to host #2

Then: the controller sends a flow_mod message to the switch

Test

```
it "should send a flow_mod message" do
  controller("RepeaterHub").should_receive(:send_flow_mod_add).with(0xabc)

  send_packets "host1", "host2"
end

# => FAIL!
```

Sending a flow-mod

```
class RepeaterHub < Trema::Controller
  def packet_in datapath_id, message
    # An empty flow_mod (no match, no actions)
    send_flow_mod_add datapath_id
  end
end

# => SUCCESS
```

Simplified incremental development

Add extra parameters to methods as you get to know more

```
# No match and no actions by default
send_flow_mod_add datapath_id

# Add a match
send_flow_mod_add datapath_id, :match => match1

# Add actions
send_flow_mod_add datapath_id,
                  :match => match1, :actions => [act1, act2]

# Set idle_timeout (default = 0)
send_flow_mod_add datapath_id,
                  :idle_timeout = 60,
                  :match => match1, :actions => [act1, act2]
```

Test the number of flow entries

```
# Testee is the switch
describe "switch" do
  it "should have one flow entry" do
    send_packets "host1", "host2"

    switch("switch").should have(1).flows
  end
end

# => SUCCESS
```

Flow-entry property

```
describe "switch" do
  it "should have one flow entry" do
    send_packets "host1", "host2"

    switch("switch").should have(1).flows
    switch("switch").flows.first.actions.should == "FLOOD"
  end
end

# => FAIL
#
#  1) RepeaterHub switch should have one flow entry
#     Failure/Error: switch("switch").flows.first.actions.should =
#       expected: "FLOOD"
#            got: "drop" (using ==)
```

Add actions

```
def packet_in message
  send_flow_mod_add(
    message.datapath_id,
    :actions => ActionOutput.new(OFPP_FLOOD)
  )
end

# => FAIL
#  1) RepeaterHub should send a flow_mod message
#     Failure/Error: network {
#       #<RepeaterHub:0xb7420d94> received :send_flow_mod_add with unexpecte
#         expected: (2748)
#              got: (2748, {:actions=>#<Trema::ActionOutput:0xb741a368 @port
```

Fix broken test

```
it "should send a flow_mod message" do
  controller("RepeaterHub").should_receive(:send_flow_mod_add)
                           .with(0xabc, hash_including(:actions))

  send_packets "host1", "host2"
end

# => SUCCESS
```

Flow-entry property

```
  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" }
  # ...

describe "switch" do
  it "should have one flow entry" do
    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

# => FAIL
#
#  1) RepeaterHub switch should have one flow entry
#     Failure/Error: flow.nw_src.should == "192.168.0.1"
#       expected: "192.168.0.1"
#            got: nil (using ==)
```

Set match structure

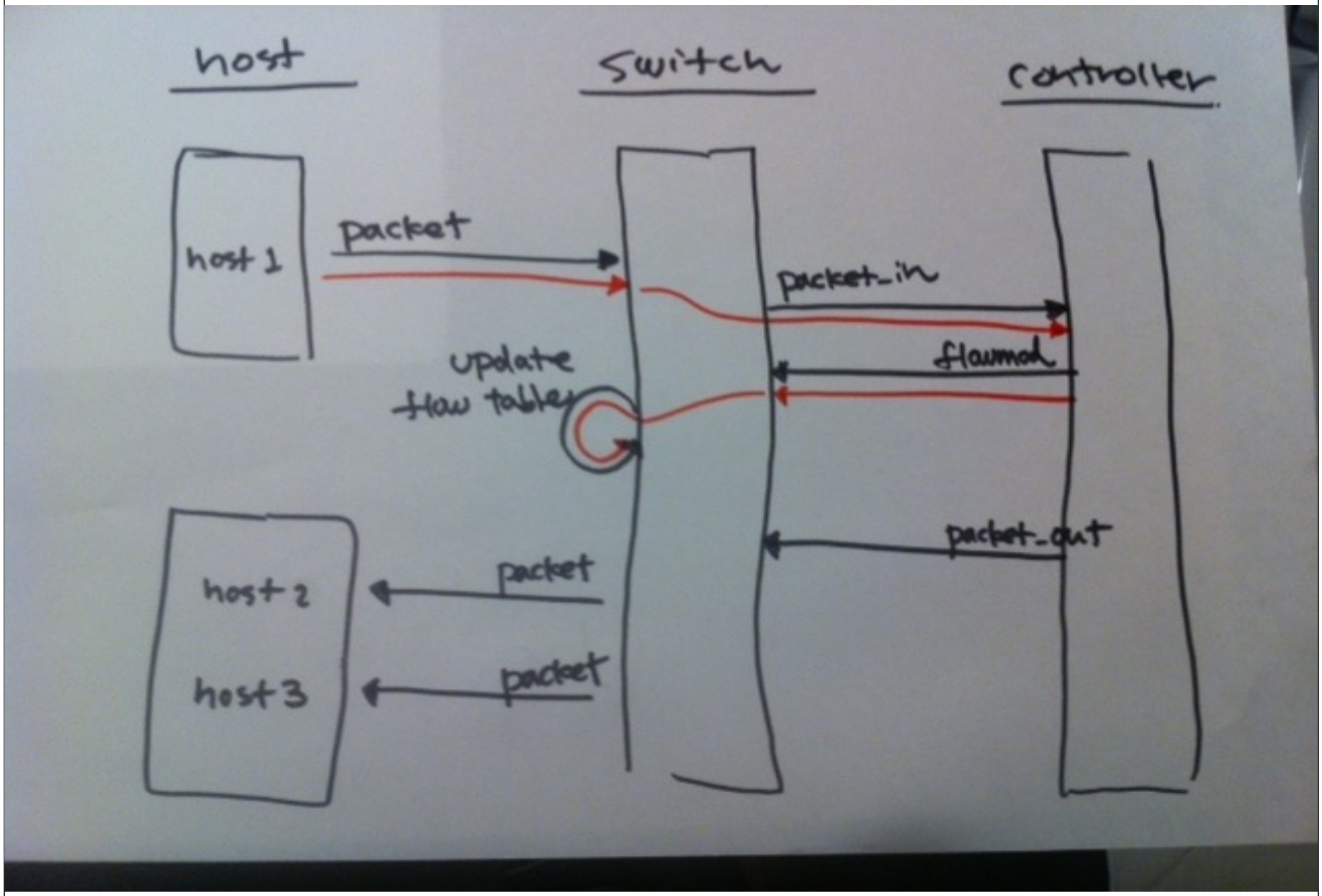
```
class RepeaterHub < Trema::Controller
  def packet_in datapath_id, message
    send_flow_mod_add datapath_id,
                      :match => ExactMatch.from(message),
                      :actions => ActionOutput.new(OFPP_FLOOD)
  end
end

# => SUCCESS
```

ExactMatch.from()

```
ExactMatch.from(message)

# vs.

Match.new(
  :in_port = message.in_port,
  :nw_src => message.nw_src,
  :nw_dst => message.nw_dst,
  :tp_src => message.tp_src,
  :tp_dst => message.tp_dst,
  :dl_src => message.dl_src,
  :dl_dst => message.dl_dst,
    ...
)
```

TODO

- ~~packet_in (switch → controller)~~
- ~~flow_mod (controller → switch)~~
- ~~Update flow table. (switch)~~
- packet_out (controller → switch)


host
switch
controller
host 1
packet
packet_in
flowmod
update
flow table
packet_out
host 2
packet
host 3
packet

Iteration #5

"Packet-out"

Again, "flooding packets" test

Remove "pending"

```
describe "host" do
  it "should flood incoming packets to every other port" do
    send_packets "host1", "host2"

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

# => FAIL!
```

Let's packet_out

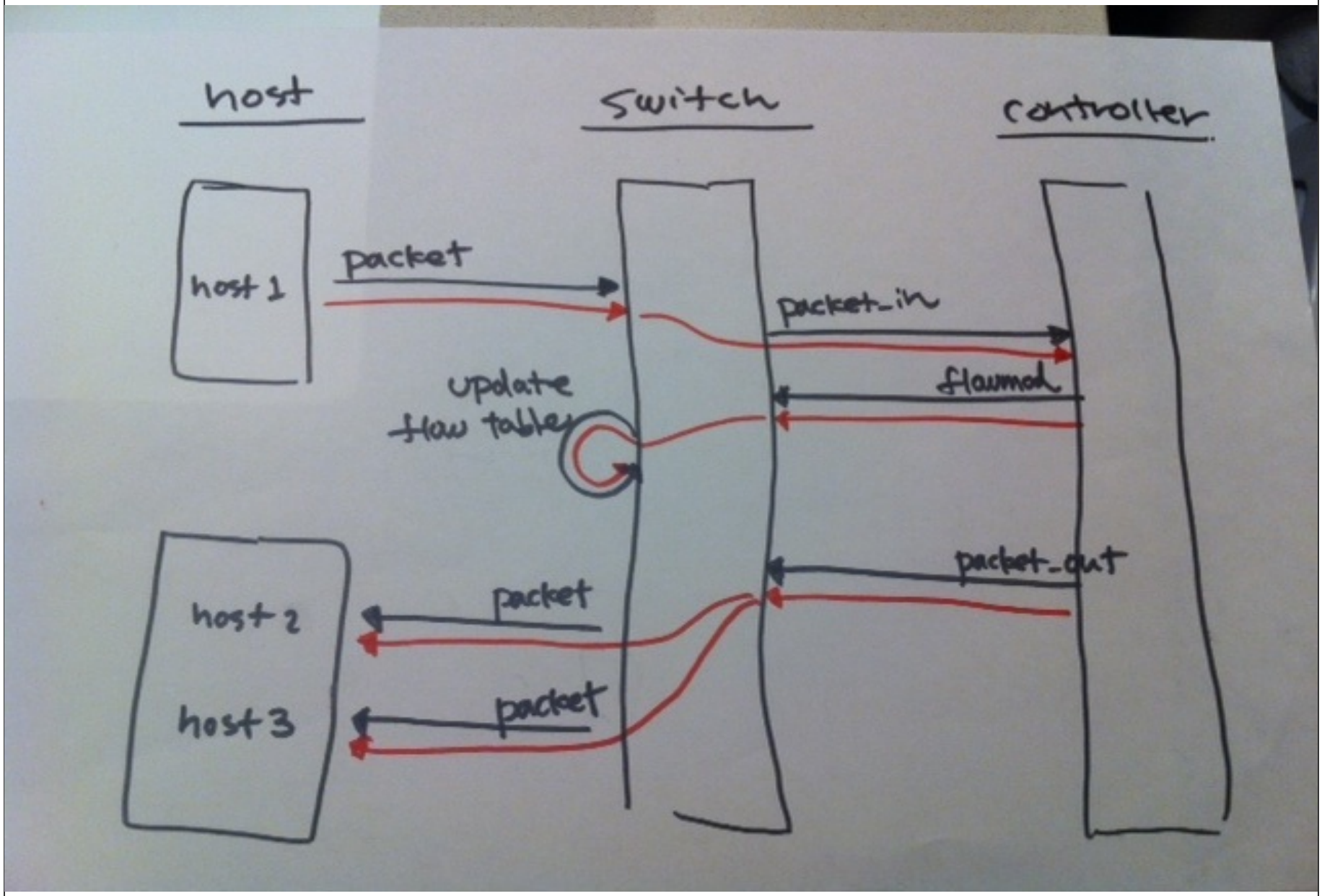
```
class RepeaterHub < Trema::Controller
  def packet_in datapath_id, message
    send_flow_mod_add(
      datapath_id,
      :match => ExactMatch.from(message),
      :actions => Trema::ActionOutput.new(OFPP_FLOOD)
    )
    send_packet_out( # Add
      datapath_id,
      :packet_in => message,
      :actions => Trema::ActionOutput.new(OFPP_FLOOD)
    )
  end
end

# => SUCCESS
```

Packet-out API

```
send_packet_out(
  datapath_id,
  :packet_in => message,
  :actions => Trema::ActionOutput.new(OFPP_FLOOD)
)

# vs.

send_packet_out(
  datapath_id,
  :data => message.buffered? ? nil : message.data,
  :in_port => message.in_port,
  :buffer_id => message.buffer_id,
  :actions => Trema::ActionOutput.new(OFPP_FLOOD)
)
```

TODO

- ~~packet_in (switch → controller)~~
- ~~flow_mod (controller → switch)~~
- ~~Update flow table. (switch)~~
- ~~packet_out (controller → switch)~~


host
switch
controller
host 1
packet
packet_in
flowmod
update
flow table
packet_out
host 2
packet
host 3
packet

That's all?

Few more tests left to do

Send packets from host2 or host3

Send many packets and receive them all (stress test)

Subsequent packets with the same flow does not cause another packet_in

Full version:

src/examples/repeater_hub/repeater-hub_spec.rb

Summary

Test framework integrated with virtual network environment

Incremental development with test-first

Clean and concise coding using Ruby

More samples: src/examples/**/*.rb

Questions?