OpenFlow Framework Trema Hands-on Tutorial

Hideyuki Shimonishi
@hide_shimonishi
Yasuhito Takamiya @yasuhito
Kazushi Sugyo
NEC

CHANGE & OFELIA Summer School
November 9, 2011

Agenda

- Introduction and design condept
- Hands-on coding
- Summary

We are...

Trema team from NEC laboratories

- Hideyuki (Networking Ninja)
- Yasuhito (HPC and Middleware)
- Kazushi (KAME: IPv6 stack for *BSD)

Q: Why Framework?

A: High productivity (e.g., Rails)

- "common operations" can be written in short
- Full-stack: develop with your laptop

→ Tight loops of "coding, testing, and debugging"

...And there were no such programming framework in OpenFlow yet.

Features of Trema Framework

Most "modern" framework for OpenFlow

- Ruby or C (Ruby's productivity + C's speed)
- Full-stack: develop with your laptop
- Open development process on github (GPL2)

Voices from Experts

*"I poked through Trema recently. It looks like a *great* project. Very clean." (openflow-discuss list)*

Martin Casado (Nicira CTO)

Trema Design Concept

"Write it short, run it quickly"

Hideyuki Shimonishi, Yasuhito Takamiya, Kazushi Sugyo
CHANGE & OFELIA Summer School 2011

Trema on github
Twitter @trema_news

Trema Design Concept

" Write it short, run it quickly"

Hideyuki Shimonishi, Yasuhito Takamiya, Kazushi Sugyo
CHANGE & OFELIA Summer School 2011

Trema on github
Twitter @trema_news

Why short code?

If you can keep your code short,

- more quickly you can finish your product,
- less incidence of software bugs,
- and easier to maintainance your code.

Example: Trema repeater-hub

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

Very concise, and there are only a few essential pieces (flow_mod + FLOOD) in this code.

Let's compare Trema with

- NOX (Python binding)
- Beacon (Java)

NOX (Python binding)

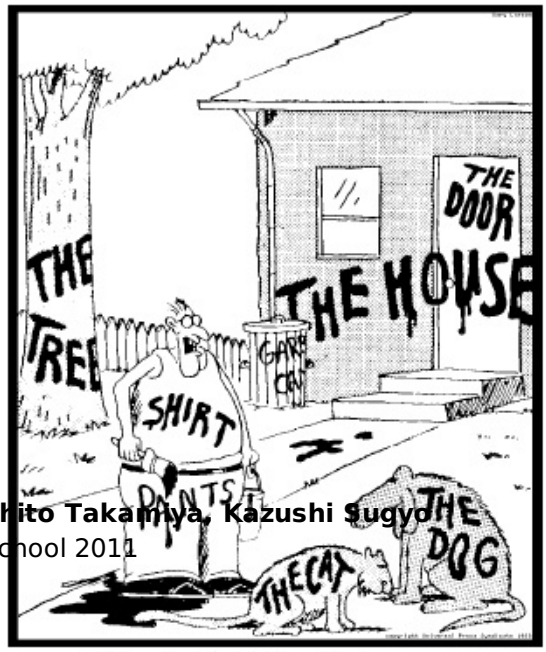
Redundant code!

```
class pyswitch(Component):
    def __init__(self, ctxt):
        global inst
        Component.__init__(self, ctxt)
        self.st = {}
        inst = self

    def install(self):
        inst.register_for_packet_in(packet_in_callback)
        inst.register_for_datapath_leave(datapath_leave_callback)
        inst.register_for_datapath_join(datapath_join_callback)
        inst.post_callback(1, timer_callback)

    def getInterface(self):
        return str(pyswitch)

    def getFactory():
        class Factory:
            def instance(self, ctxt):
                return pyswitch(ctxt)
        return Factory()
```

Beacon (Java)

We don't use Java :-) (static typing system WITHOUT good type inference mechanism makes your code looong!)

Hideyuki Shimonishi, Yasuhito Takamiya, Kazushi Sugyo
CHANGE & OFELIA Summer School 2011



"Now! *That* should clear up
a few things around here!"

Trema on github
Twitter @trema_news

Trema Design Concept

"Write it short, run it quickly"

Hideyuki Shimonishi, Yasuhito Takamiya, Kazushi Sugyo
CHANGE & OFELIA Summer School 2011

Trema on github
Twitter @trema_news

Pains of OpenFlow development

"I don't have a OpenFlow switch."

"It is complicated to setup OpenFlow development environment (network configurations and VM installations etc.)"

"There are no network environments here usable for OpenFlow experiments."

Anonymous programmer

Trema Works with No Pain

You can build a virtual network on your laptop and test your own controllers on it.

- Run your controller without modifications
- Send and receive test packets between virtual hosts
- Dump packets and flow-tables etc.

See how it works.

Hands-on Coding Section

Walk-Through of Development with Trema

Hideyuki Shimonishi, Yasuhito Takamiya, Kazushi Sugyo
CHANGE & OFELIA Summer School 2011

Trema on github
Twitter @trema_news

Let's Setup Trema!

```
$ git clone git://github.com/trema/trema.git
$ ./trema/build.rb
```

And there are no step three!

Coding #1: Your first controller

Define a controller as a class and make it derived from Controller class

```
# simple.rb
class SimpleController < Controller
end
```

Run it with trema run command:

```
% ./trema run ./simple.rb
```

Coding #2: Hello Trema!

Add a start method to the controller class.

```
# hello.rb
class HelloController < Controller
  def start
    puts "Hello, Trema!"
  end
end
```

It is a hook method invoked at startup (initializer).

You can run it with trema run

```
% ./trema run ./hello.rb
Hello, Trema!
```

Class in Ruby

```
class ClassName < SuperClassName
  def method_name arg1, arg2
    # method body
  end
end
```

Hideyuki Shimonishi, Yasuhito Takamiya, Kazushi Sugyo
CHANGE & OFELIA Summer School 2011

Trema on github
Twitter @trema_news

Coding #3: Switch Monitor

Switch monitor tracks all switches in a network

- Updates the list of switches in real-time
- Notifies when a new switch is UP
- Notifies when a switch becomes DOWN

Coding #3: Switch Monitor

```
# src/examples/switch_monitor/switch-monitor.rb
class SwitchMonitor < Controller
  periodic_timer_event :show_switches, 10

  def start
    @switches = []
  end

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

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

  private

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

Event Handlers

```
# src/examples/switch_monitor/switch-monitor.rb
class SwitchMonitor < Controller
  periodic_timer_event :show_switches, 10

  def start
    @switches = []
  end

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

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

  private

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

switch_ready handler

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

  def start
    @switches = []
  end

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

- When a switch is connected to the controller, add (<<) its dpid to the switch-list instance variable (@switches)
- Then output like "Switch 0xabc is UP"

List in Ruby

```
% irb
> zoo = []
=> []

> zoo << "bat"
=> ["bat"]

> zoo << "gnu"
=> ["bat", "gnu"]

> zoo << "dog"
=> ["bat", "gnu", "dog"]

> zoo -= ["bat"]
=> ["gnu", "dog"]

> zoo.sort
=> ["dog", "gnu"]
```

irb = interactive ruby (REPL)

switch_disconnected handler

- When a switch becomes DOWN, remove (-=) its dpid from @switches list
- Then output like "Switch 0xabc is DOWN"

```
# src/examples/switch_monitor/switch-monitor.rb
class SwitchMonitor < Controller

    ...

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

    ...
```

show_switches timer handler

- Output the switch list like "All switches = 0xabc, 0xdef, ..." in every 10 seconds
- periodic_timer_handler registers show_switches method as a timer handler

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

    ...

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

Run Switch Monitor

Write a virtual network configuration (DSL) that adds three switches to the network.

```
# src/examples/switch_monitor/switch-monitor.conf
vswitch { datapath_id 0x1 }
vswitch { datapath_id 0x2 }
vswitch { datapath_id 0x3 }
```

Then pass it to trema run with -c option

```
% ./trema run ./src/examples/switch_monitor/switch-monitor.rb -c ./src/examples/switch_monitor/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
  ...
```

Virtual Network DSL

- You can build arbitrally network topology on your laptop and run your controlller on it.
- Debug your controller by sending and receiving test packets between virtual hosts.

```
switch_monitor/switch-monitor.conf
```

- You can use virtual switch, host, and link to construct the virtual network.

Shutdown a Switch

Let's test switch_disconnected handler by causing a fault in the virtual network

Open another terminal and run trema kill

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

Then you can see "Switch 0x??? is DOWN"

```
% ./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
```

Summary: Switch Monitor

- Implement the logic of a controller by adding event handlers (e.g., switch_ready and switch_disconnected)
- Network DSL makes it possible to test your own controller even if you don't have OpenFlow switches.

switch_monitor/switch_monitor.conf

Coding #4: packet-in handling

Add the following method to handle packet-in

```
class SimpleController < Controller
  def packet_in dpid, message
    puts "New packet_in message!"
  end
    ...
```

- dpid: the datapath ID of the switch that received a packet-in
- message: packet-in message object

Let's Send Packets!

Firstly, you must add virtual hosts to send/receive packets and connect them to a virtual switch:

```
vswitch { dpid 0xabc }

# Add two virtual hosts
vhost("host1") {
  mac "00:00:00:00:00:01"
}
vhost("host2") {
  mac "00:00:00:00:00:02"
}

# connect hosts to a switch
link "0xabc", "host1"
link "0xabc", "host2"
```

Let's Send Packets!

- Send test packets with send_packets
- The following will send ten packets from host1 to host2 in a second

```
% ./trema send_packets --source host1 --dest host2 --n_pkts 10 --pps 10
New packet_in message!
New packet_in message!
New packet_in message!
New packet_in message!
New packet_in message!
  ...
```

Large and complex topology?

Ten switches full-mesh topology configurations in flat (about 80 LOC)

```
vswitch { dpid "0x1" }
vswitch { dpid "0x2" }
vswitch { dpid "0x3" }
vswitch { dpid "0x4" }
  ...

link "0x1", "0x2"
link "0x1", "0x3"
link "0x1", "0x4"
link "0x1", "0x5"
link "0x1", "0x6"
link "0x1", "0x7"
link "0x1", "0x8"
link "0x1", "0x9"
link "0x1", "0x10"
link "0x2", "0x3"
link "0x2", "0x4"
  ...
```

Advanced DSL Usage

- You can use full Ruby syntax in the DSL
- the full-mesh topology can be rewritten briefly with nested loops:

```
$nswitch = 10

1.upto( $nswitch ).each do | sw1 |
  vswitch { dpid sw1.to_hex }

  1.upto( $nswitch ).each do | sw2 |
    link sw1.to_hex, sw2.to_hex if sw1 < sw2
  end
end
```

Very concise!

Coding #5: Modifying flow-table

Write a flow-entry that forwards incoming packets to the next switch port

```
class SimpleController < Controller
  def packet_in dpid, message
    send_flow_mod_add(
      dpid,
      :match => ExactMatch.from( message ),
      :buffer_id => message.buffer_id,
      :actions => ActionOutput.new( message.in_port + 1 )
    )
  end
end
```

Very concise!

Comparison of flow_mod API

Trema

```
send_flow_mod_add(
  dpid,
  :match => ExactMatch.from( message ),
  :buffer_id => message.buffer_id,
  :actions => ActionOutput.new( message.in_port + 1 )
)
```

NOX

```
inst.install_datapath_flow(
  dpid,
  extract_flow(packet),
  CACHE_TIMEOUT,
  openflow.OFP_FLOW_PERMANENT,
  [[openflow.OFPAT_OUTPUT, [0, prt[0]]]],
  bufid,
  openflow.OFP_DEFAULT_PRIORITY,
  inport,
  buf
)
```

Creating a Match Structure

You can easily create an exact match from incoming packet_in message:

```
send_flow_mod_add(
  dpid,
  :match => ExactMatch.from( message ),
  :buffer_id => message.buffer_id,
  :actions => ActionOutput.new( message.in_port + 1 )
)
```

This is equivalent with:

```
:match => 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,
    ...
)
```

Traffic Monitor

- L2 switch
- Counts the number of packets sent by each user
- Each user is identified by its MAC address

Traffic Monitor classes

- FDB class learns {MAC, switch-port} pair
- Counter class keeps number of packets sent by each user
- TrafficMonitor class acts as a Controller

Let's implement them one by one

Coding #6: FDB Class

```
# src/examples/traffic_monitor/fdb.rb
class FDB
  def initialize
    @db = {}
  end

  def lookup mac
    @db[ mac ]
  end

  def learn mac, port_number
    @db[ mac ] = port_number
  end
end
```

FDB Class

```
% irb
> require "fdb"
> fdb = FDB.new

> fdb.lookup "00:00:00:00:00:01"
=> nil

> fdb.learn "00:00:00:00:00:01", 1
> fdb.lookup "00:00:00:00:00:01"
=> 1

> fdb.learn "00:00:00:00:00:02", 2
> fdb.lookup "00:00:00:00:00:02"
=> 2
```

Hash in Ruby

```
% irb
> price = { "Mac" => 1000, "iPhone" => 500, "iPad" => 600 }
=> { "Mac" => 1000, "iPhone" => 500, "iPad" => 600 }

> price[ "iPhone" ]
=> 500

> price[ "iPad" ]
=> 600

> price[ "iPod Touch" ]
=> nil

> price[ "iPod Touch" ] = 300
=> 300

> price[ "iPod Touch" ]
=> 300
```

Coding #7: Counter Class

```
# src/examples/traffic_monitor/counter.rb
class Counter
  def initialize
    @db = {}
  end

  def add mac, packet_count, byte_count
    @db[ mac ] ||= { :packet_count => 0, :byte_count => 0 }
    @db[ mac ][ :packet_count ] += packet_count
    @db[ mac ][ :byte_count ] += byte_count
  end

  def each_pair &block
    @db.each_pair &block
  end
end
```

Counter Class

```
% irb
> require "counter"
> counter = Counter.new

> counter.add "00:00:00:00:00:01", 1, 100
> counter.add "00:00:00:00:00:02", 1, 100
> counter.add "00:00:00:00:00:02", 1, 100

> counter.each_pair do | mac, counter |
>   p mac, counter
> end
"00:00:00:00:00:01"
{:packet_count=>1, :byte_count=>100}
"00:00:00:00:00:02"
{:packet_count=>2, :byte_count=>200}
```

TrafficMonitor Class Overview

TrafficMonitor has the following two handlers

- packet_in: L2 switching. When a packet_in comes, learn MAC <=> switch-port pair and send flow_mod to a switch
- flow_removed: counts the number of packets and bytes from a user. This information is piggy-backed with flow_removed messages

Coding #8: TrafficMonitor#packet_in

```
class TrafficMonitor < Controller
  periodic_timer_event :show_counter, 10

  def start
    @counter = Counter.new
    @fdb = FDB.new
  end

  def packet_in datapath_id, message
    macsa = message.macsa
    macda = message.macda

    @fdb.learn macsa, message.in_port
    out_port = @fdb.lookup( macda )
    if out_port
      flow_mod datapath_id, macsa, macda, out_port
      packet_out datapath_id, message, out_port
    else
      flood datapath_id, message
    end
  end
    ...
```

Coding #9: TrafficMonitor#flow_mod

```
private
  ...
def flow_mod datapath_id, macsa, macda, out_port
  send_flow_mod_add(
    datapath_id,
    :hard_timeout => 10,
    :match => Match.new(
      :dl_src => macsa.to_s,
      :dl_dst => macda.to_s
    ),
    :actions => Trema::ActionOutput.new( out_port )
  )
end
```

:hard_timeout forces flow-entries to be removed in ten seconds and therefore causes flow_removed

Coding #10: TrafficMonitor#packet_out and flood

```
class TrafficMonitor < Controller
  ...
  private
  ...
  def packet_out datapath_id, message, out_port
    send_packet_out(
      datapath_id,
      :packet_in => message,
      :actions => Trema::ActionOutput.new( out_port )
    )
  end

  def flood datapath_id, message
    packet_out datapath_id, message, OFPP_FLOOD
  end
end
```

Coding #11: TrafficMonitor#flow_removed

```
class TrafficMonitor < Controller
  ...

def flow_removed datapath_id, message
  @counter.add(
    message.match.dl_src,
    message.packet_count,
    message.byte_count
  )
end
```

Run Traffic Monitor

DSL

```
vswitch { dpid 0xabc }

# Add two virtual hosts
vhost("host1") {
  mac "00:00:00:00:00:01"
}
vhost("host2") {
  mac "00:00:00:00:00:02"
}

# connect hosts to a switch
link "0xabc", "host1"
link "0xabc", "host2"
```

Run

```
% ./trema run ./traffic-monitor.rb -c ./traffic-monitor.conf
```

Let's Send Packets!

Open a new terminal and send packets betweeen host1 and host2:

```
% ./trema send_packets --source host1 --dest host2 --n_pkts 10 --pps 10
% ./trema send_packets --source host2 --dest host1 --n_pkts 10 --pps 10
```

Then the trema run terminal will look like:

```
...
00:00:00:00:00:01 10 packets (640 bytes)
00:00:00:00:00:02 10 packets (640 bytes)
...
```

Wrap-up

Hideyuki Shimonishi, Yasuhito Takamiya, Kazushi Sugyo
CHANGE & OFELIA Summer School 2011

Trema on github
Twitter @trema_news

Trema Sample Apps

- Trema distribution includes various samples (in Ruby and C)
- We recommend to run them and read its source code

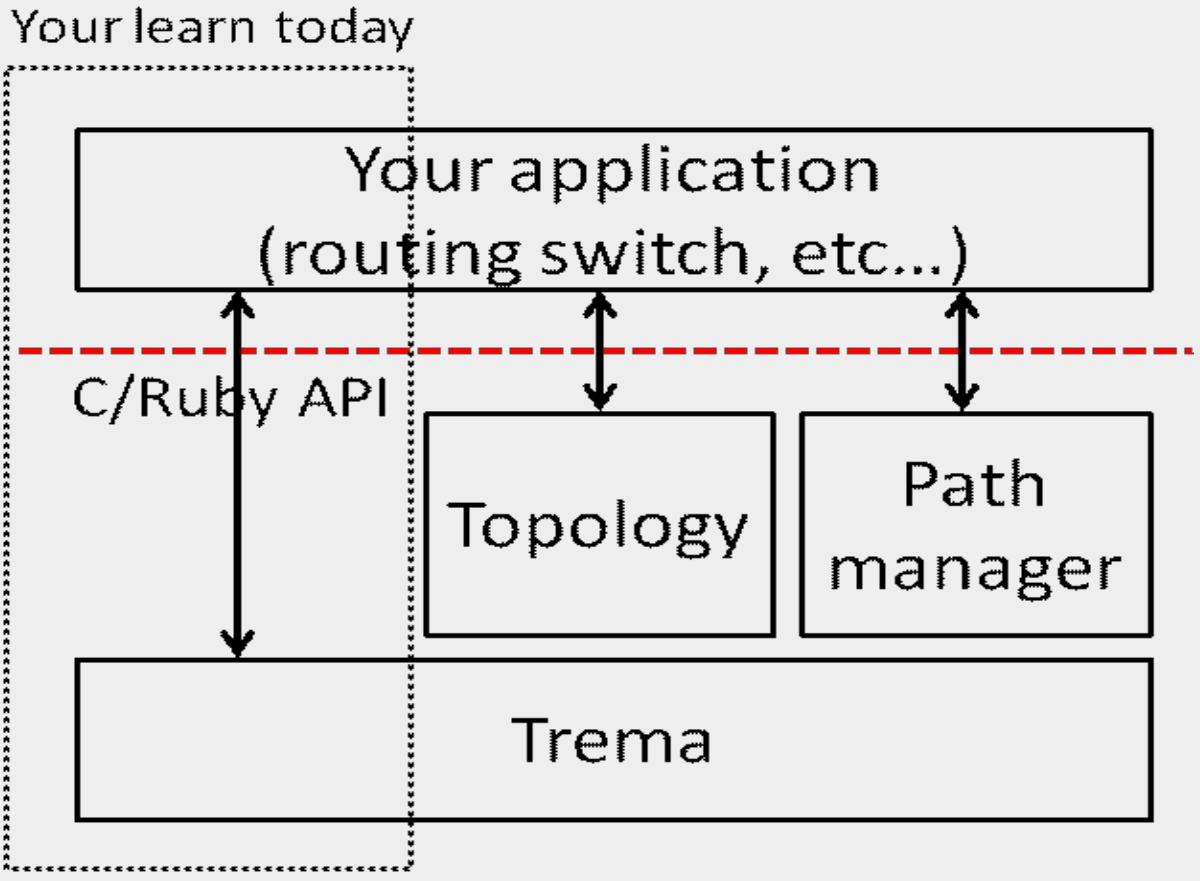
```
% ls src/examples
cbench_switch          dumper             hash_bench
hello_trema            learning_switch    list_switches
multi_learning_switch  openflow_message   packet_in
repeater_hub           switch_info        switch_monitor
traffic_monitor
```

trema/apps

https://github.com/trema/apps

"Nearly production-level" apps written in Trema

- routing switch
- load balancer
- topology manager etc. etc...


Your learn today
Your application
(routing switch, etc...)
C/Ruby API
Topology
Path
manager
Trema

Performance: cbench benchmarks

cbench = A controller benchmark that measures flow-setup/sec

```
1   switches: fmods/sec:  10627   total = 10.626989 per ms
1   switches: fmods/sec:  8184   total = 8.183992 per ms
1   switches: fmods/sec:  7542   total = 7.541992 per ms
1   switches: fmods/sec:  7852   total = 7.851992 per ms
1   switches: fmods/sec:  8243   total = 8.242992 per ms
1   switches: fmods/sec:  7807   total = 7.806977 per ms
1   switches: fmods/sec:  8484   total = 8.483992 per ms
1   switches: fmods/sec:  8401   total = 8.400992 per ms
1   switches: fmods/sec:  8129   total = 8.128992 per ms
1   switches: fmods/sec:  7853   total = 7.852788 per ms
RESULT: 1 switches 9 tests min/max/avg/stdev = 7541.99/8483.99/8054.97/292.12 responses/s
```

About 8,000 flow-setup/sec

cbench with C version controller

About 14,000 flow-setup/sec

```
1   switches: fmods/sec:  8891   total = 8.890991 per ms
1   switches: fmods/sec:  14864   total = 14.863985 per ms
1   switches: fmods/sec:  14316   total = 14.315986 per ms
1   switches: fmods/sec:  14548   total = 14.547985 per ms
1   switches: fmods/sec:  14648   total = 14.647956 per ms
1   switches: fmods/sec:  13318   total = 13.317987 per ms
1   switches: fmods/sec:  14376   total = 14.375986 per ms
1   switches: fmods/sec:  13143   total = 13.142987 per ms
1   switches: fmods/sec:  14275   total = 14.274986 per ms
1   switches: fmods/sec:  14280   total = 14.279986 per ms
RESULT: 1 switches 9 tests min/max/avg/stdev = 13142.99/14863.99/14196.43/549.16 responses/s
```

sponses/s

We recommend to write a controller in Ruby at first, then rewrite them with C if need be

Hideyuki Shimonishi, Yasuhito Takamiya, Kazushi Sugyo
CHANGE & OFELIA Summer School 2011

Trema on github
Twitter @trema_news

Unit-Test

- You can write unit-tests of send/receive packets between hosts and flow-table entries of switches etc. with RSpec
- The details can be found in spec/ directory

responses/s

sponses/s

Unit-Test Example

```
describe RepeaterHub do
  around do | example |
    network {
      vswitch("switch") { datapath_id "0xabc" }
      vhost("host1") { promisc "on" }
      vhost("host2") { promisc "on" }
      link "switch", "host1"
      link "switch", "host2"
    }.run( RepeaterHub ) {
      example.run
    }
  end

  context "when host1 sends one packet to host2" do
    describe "switch" do
      before { send_packets "host1", "host2" }
      subject { switch( "switch" ) }
      it { should have( 1 ).flows }
      its( "flows.first.actions" ) { should == "FLOOD" }
    end

      ...
```

responses/s

sponses/s

Summary

OpenFlow frmework Trema

- You can write controllers short and run it quickly
- Many examples in src/examples and trema/apps repository
- Pull-requests and bug-reports are welcome!

responses/s

sponses/s

Meta

- Trema HomePage
  http://trema.github.com/trema/
- Ruby API Document:
  http://rubydoc.info/github/trema/trema/
- Twitter Account: @trema_news

responses/s

sponses/s