Trema Basic Operations

Yasunobu Chiba

Oct. 5, 2011

Objectives and agenda

- Objectives
  - Learn how to build and run Trema and its sample applications
  - Learn how to configure the integrated network emulator and applications
  - Learn how to use ‘trema’ command which allows you to interact with Trema, its application, and network emulator

- Agenda
  - How to obtain and build Trema
  - How to run Trema applications
  - Configuring network emulator and applications
  - ‘trema’ command overview

- There are 4 lessons (demos) – Please join the lessons if you have your own Linux environment

How to obtain and build Trema: Before you begin…

- Of course we need a build and run environment !
  - Ubuntu 11.04 (i386/amd64)
  - Ubuntu 10.10 (i386/amd64)
  - Ubuntu 10.04 – LTS (i386/amd64)
  - Debian GNU/Linux 6.0 (i386/amd64)
- May also work on other modern/latest Linux operating systems with Linux 2.6.26+, GCC 4.3+, and Ruby 1.8.7
- CentOS 5.x is definitely NOT supported due to very old Linux kernel and GCC
- Please make sure you are connected to the Internet !

How to obtain and build Trema: It's time to get and build

- Make sure that all necessary software packages are

```
$ sudo apt-get install git gcc make ruby ruby-dev rubygems
libpcap-dev libsqlite3-dev
```

- Get Trema from github

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

- Checkout the latest release and build

```
$ cd trema
$ git tag -l
0.1.0
X.Y.Z
$ git checkout X.Y.Z
$ ./build.rb
```

Find the latest release from the list

That's It!

Lesson 1 : Let's build !

```
$ sudo apt-get install git gcc make ruby ruby-dev rubygems
libpcap-dev libsqlite3-dev
$ git clone git://github.com/trema/trema.git
$ cd trema
$ git tag -l
0.1.0
X.Y.Z
$ git checkout X.Y.Z
$ ./build.rb
```

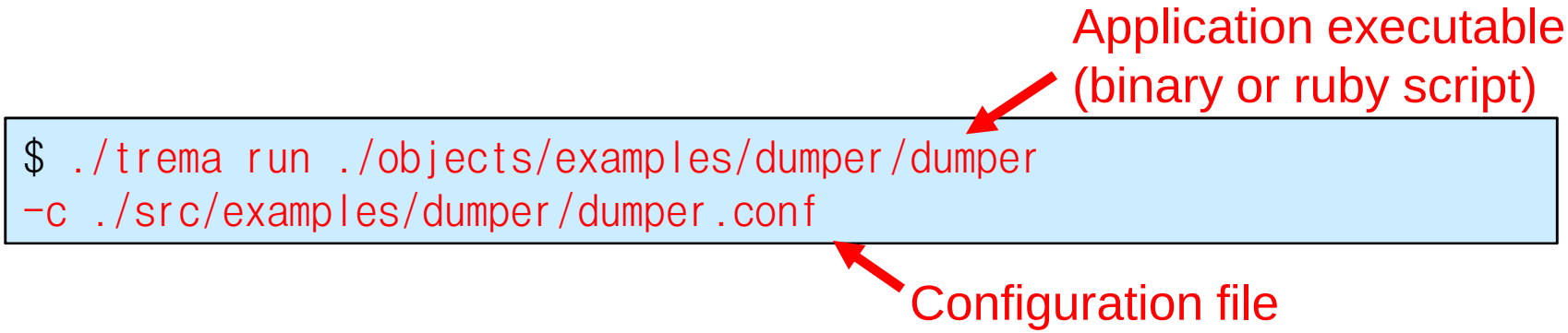
How to run Trema applications: Sample applications

- Sample applications found in trema/src/examples:
  - Dumper : OpenFlow event dumper
  - Repeater Hub : Repeater Hub emulation
  - Learning Switch : Learning switch emulation with a single OpenFlow Switch
  - Multi Learning Switch: Learning switches emulation with multiple OpenFlow switches support
  - etc.

- Most of samples have both C and Ruby version code with the same functionality

How to run Trema applications: It's time to run !

- Run Dumper with a configuration file




```
[switch_ready]
datapath_id: 0xabc
[features_reply]
datapath_id: 0xabc
transaction_id: 0x660c0001
n_buffers: 256
n_tables: 2
capabilities: 0x87
actions: 0x7ff
port_no: 65534
  hw_addr: da:ad:cd:76:d8:3f
  name: vsw_0xabc
...
```

How to run Trema applications: It's time to terminate !

- Just type Ctrl-C to terminate running application

```
$ ./trema run ./objects/examples/dumper/dumper
-c ./src/examples/dumper/dumper.conf
^C
terminated
```

How to run Trema applications: Run in background?

- Just add '-d' option

```
$ ./trema run ./objects/examples/dumper/dumper
-c ./src/examples/dumper/dumper.conf -d
```

- 'trema killall' to terminate daemonized application

```
$ ./trema killall
```

How to run Trema applications: Other sample apps?

- Repeater Hub

```
$ ./trema run ./objects/examples/repeater_hub/repeater_hub
-c ./src/examples/repeater_hub/repeater_hub.conf
```

- Learning Switch

```
$ ./trema run ./objects/examples/learning_switch/learning_switch
-c ./src/examples/learning_switch/learning_switch.conf
```

- Multi Learning Switch

```
$ ./trema
run ./objects/examples/multi_learning_switch/multi_learning_switch
-c ./src/examples/multi_learning_switch/multi_learning_switch.conf
```

How to run Trema applications: Run w/o network emu?

- Don’t worry. Just run without configuration file !

```
$ ./trema run ./objects/examples/dumper/dumper
```

- Now Trema accepts secure channel connections from any OpenFlow switches without network emulator

It is not possible to allow connections only from authorized switches at this moment. But still application can ignore any events from unauthorized (unknown) switches.

Even if you have an emulator configuration in configuration file, secure channel connections are accepted.

Detailed configuration file structure is explained later.

How to run Trema applications: Need to debug?

- Set LOGGING_LEVEL environment variable to change log level

```
$ LOGGING_LEVEL=debug ./trema run ./objects/examples/dumper/dumper
```

critical, **error**, **warn**, **notice**, **info**, or **debug** is allowed to set. **info** is default.

- Log files found in trema/tmp/log
- Sending SIGUSR1 to applications output statistics collected by Trema to stdout or log files (if daemonized)

```
$ kill -USR1 `cat tmp/dumper.pid`
```

Lesson 2 : Let’s run dumper sample

```
$ ./trema run ./objects/examples/dumper/dumper
-c ./src/examples/dumper/dumper.conf
^C
$ ./trema run ./objects/examples/dumper/dumper
-c ./src/examples/dumper/dumper.conf -d
$ cat tmp/log/dumper.log
$ ./trema killall
```

Configuring network emulator and applications

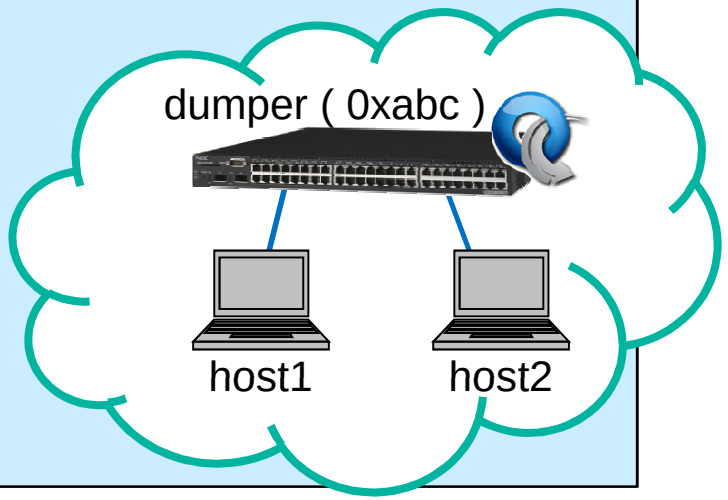
- In configuration file, you can configure
  - Integrated OpenFlow network emulator
  - Applications on top of Trema

- No configuration cause default behaviors
  - No network emulator configuration – no network emulator
  - No applications configuration – an application must be specified as an argument of ‘trema’ command

Configuring network emulator and applications

- dumper.conf defines emulated network environment

```
vswitch("dumper") { datapath_id "0xabc" }

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

link "dumper", "host1"
link "dumper", "host2"
```




You can assign any human-readable name to switch / host.


MAC and IP addresses are automatically assigned if you omit them.


How to interact with emulated network is introduced later.

Configuring network emulator and applications

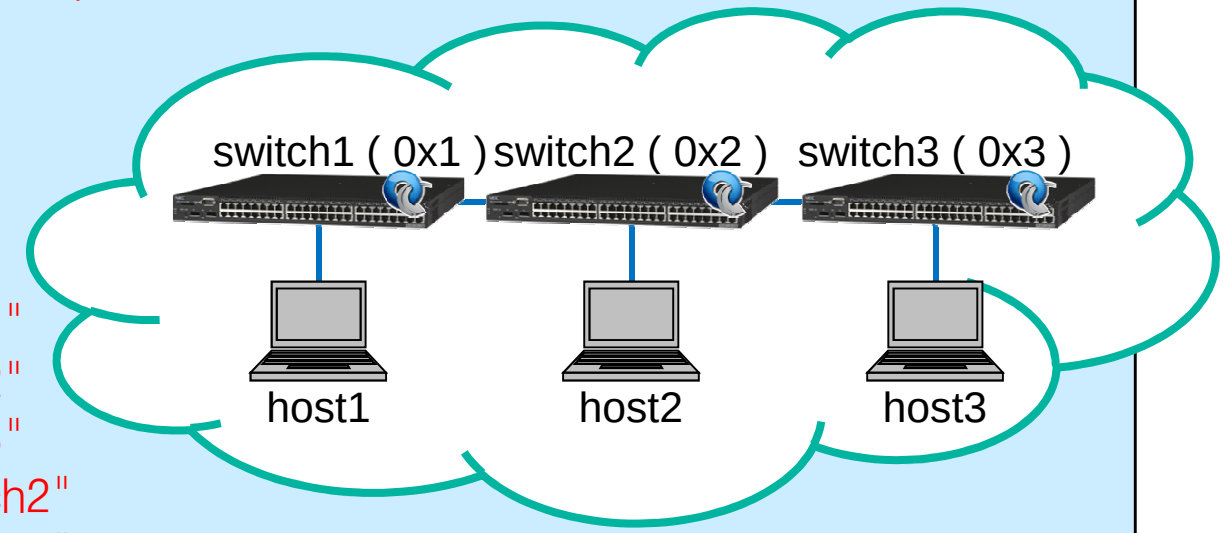
- Want to specify addresses?

```
vswitch("dumper") { datapath_id "0xabc" }

vhost("host1") {
  ip "172.16.0.1"
  netmask "255.255.255.0"
  mac "00:00:00:01:00:01"
}

vhost("host2") {
  ip "172.16.0.2"
  netmask "255.255.255.0"
  mac "00:00:00:01:00:02"
}

link "dumper", "host1"
link "dumper", "host2"
```

Configuring network emulator and applications

- Want to have more switches and hosts?

```
vswitch("switch1") { datapath_id "0x1" }
vswitch("switch2") { datapath_id "0x2" }
vswitch("switch3") { datapath_id "0x3" }

vhost("host1")
vhost("host2")
vhost("host3")

link "switch1", "host1"
link "switch2", "host2"
link "switch3", "host3"
link "switch1", "switch2"
link "switch2", "switch3"
```



Configuring network emulator and applications

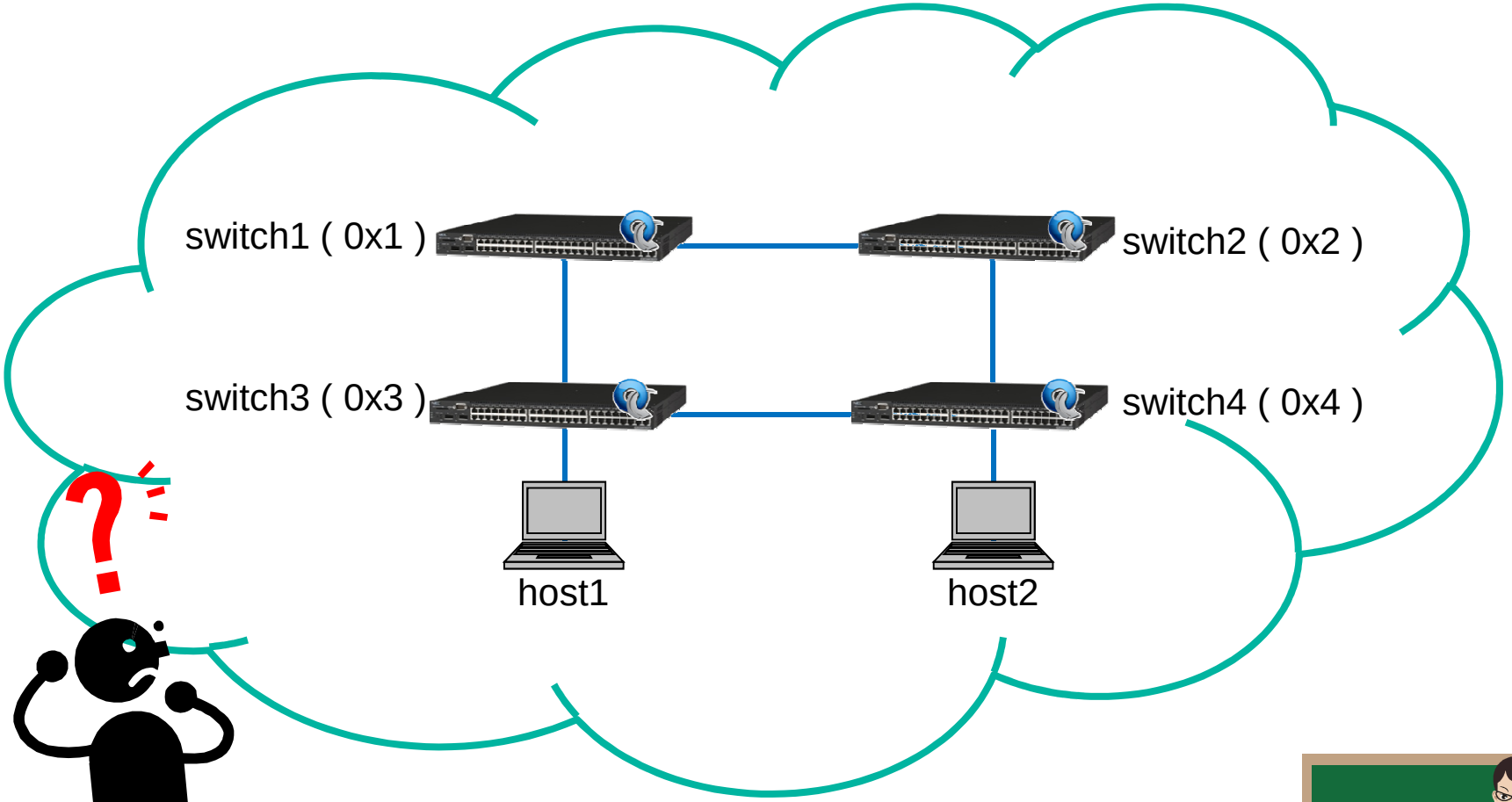
- Defining an application in configuration file?

```
vswitch("dumper") { datapath_id "0xabc" }

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

link "dumper", "host1"
link "dumper", "host2"

app {
  path "./objects/examples/dumper/dumper"
}
```

- Now you can simply run an application with a configuration file:

```
$ ./trema run -c ./src/examples/dumper/dumper.conf
```

Lesson 3 : Let’s create your emulated network



Lesson 3 : Let's create an emulated network

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

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

link "switch1", "switch2"
link "switch1", "switch3"
link "switch2", "switch4"
link "switch3", "switch4"
link "switch3", "host1"
link "switch4", "host2"
```

‘trema’ command overview

- ‘trema’ command allows you to run and terminate application developed on top of Trema (as you saw)
- It also allows you to interact with emulated network (i.e. switches and hosts)

```
$ ./trema help
usage: ./trema <COMMAND> [OPTIONS ...]

Trema command-line tool
Type './trema help <COMMAND>' for help on a specific command.

Available commands:
  run             - runs a trema application.
  kill            - terminates a trema process.
  killall         - terminates all trema processes.
  send_packets    - sends UDP packets to destination host.
  show_stats      - shows stats of packets.
  reset_stats     - resets stats of packets.
  dump_flows      - print all flow entries.
```

‘trema’ command overview: send packets

- Send UDP packets from a host to another

```
$ ./trema send_packets --source host1 --dest host2
```


Only one UDP packet is sent out by default.

- Send 1000 packets with specified port number for 10 seconds

```
$ ./trema send_packets --source host1 --dest host2 --pps 1000
--duration 10 --tp_src 1234 --tp_dst 5678
```

- Send packets with incrementing source port number from 0

```
$ ./trema send_packets --source host1 --dest host2 --duration 10
--tp_src 0 --inc_tp_src
```


Port number is wraparound if it overflows.

‘trema’ command overview: send packets – more options

- Full options description

```
$ ./trema help send_packets
Usage: ./trema send_packets [OPTIONS ...]
    -s, --source HOSTNAME
        --inc_ip_src [NUMBER]
    -d, --dest HOSTNAME
        --inc_ip_dst [NUMBER]
        --tp_src NUMBER
        --inc_tp_src [NUMBER]
        --tp_dst NUMBER
        --inc_tp_dst [NUMBER]
        --pps NUMBER
        --n_pkts NUMBER
        --duration NUMBER
        --length NUMBER
        --inc_payload [NUMBER]

    -h, --help
    -v, --verbose
```

‘trema’ command overview: show stats and reset stats

- Show TX/RX packet counters on a host

```
$ ./trema show_stats host1 --tx
$ ./trema show_stats host1 --rx
```

- Full options description

```
$ ./trema help show_stats
Usage: ./trema show_stats [OPTIONS ...]
    -t, --tx
    -r, --rx

    -h, --help
    -v, --verbose
```

- Reset (clear) packet counters on a host

```
$ ./trema reset_stats host1
```

‘trema’ command overview: dump flows and kill

- Show flow entries registered on a switch

```
$ ./trema dump_flows dumper
```

- Kill a process managed by Trema

```
$ ./trema kill dumper
```

Only virtual switches can be terminated at this moment.

Lesson 4 : Run learning switch and confirm its behavior

1. Run learning switch sample application with two hosts (host1 and host2) and a single switch (lsw)
2. Send a packet from a host (host1) to another (host2)
3. Confirm that the packet is received on the destination host (host2)
4. Confirm that no flow entry is created
5. Send a packet from a host (host2) to another (host1)
6. Confirm that the packet is received on the destination host (host1) and a flow entries is registered on the switch

Lesson 4 : Run learning switch and confirm its behavior

```
$ ./trema run ./objects/examples/learning_switch/learning_switch -c
./src/examples/learning_switch/learning_switch.conf -d
$ ./trema send_packets --source host1 --dest host2
$ ./trema show_stats host2 --rx
ip_dst,tp_dst,ip_src,tp_src,n_pkts,n_octets
192.168.0.2,1,192.168.0.1,1,1,50
$ ./trema dump_flows lsw
Jul 24 17:51:01|00001|ofctl|INFO|connecting to unix:/home/y-chiba/trema/tmp/vsw_0xabc.mgmt
stats_reply (xid=0xc3d5ecea): flags=none type=1(flow)
$ ./trema send_packets --source host2 --dest host1
$ ./trema show_stats host1 --rx
ip_dst,tp_dst,ip_src,tp_src,n_pkts,n_octets
192.168.0.1,1,192.168.0.2,1,1,50
$ ./trema dump_flows lsw
Jul 24 17:51:13|00001|ofctl|INFO|connecting to unix:/home/y-chiba/trema/tmp/vsw_0xabc.mgmt
stats_reply (xid=0xffa1cd04): flags=none type=1(flow)
  cookie=0x1, duration_sec=7s, duration_nsec=339000000ns, table_id=1, priority=65535,
n_packets=1, n_bytes=64,
idle_timeout=60,udp,in_port=2,dl_vlan=65535,dl_vlan_pcp=0,dl_src=00:00:00:01:
00:00:00:01:00:01,nw_src=192.168.0.2,nw_dst=192.168.0.1,tp_src=1,tp_dst=1,act
```

More Advanced Topics

Not explained in this tutorial though…

Configuring multiple applications?

- You have multiple applications that handles different asynchronous events?

```
app {
  path "./somewhere/app1"
}

app {
  path "./somewhere/app2"
}

event :port_status => "app1", :packet_in => "app2", :state_notify => "app1"
```

Trema specific events
(switch connected/disconnected)

Trema-provided event distribution might be still not flexible enough. Please let us know if you have any specific requirements on event distribution.

Reply messages and Flow Removed messages are routed based on transaction ids or flow cookie values.

Interact with virtual switches directly?

- Trema uses unmodified Open vSwitch as a virtual switch for the integrated network emulator
- Thus you can interact with the switches with command line tools (i.e. ovs-ofctl) provided by Open vSwitch
- ‘trema’ creates ovs-openflowd instances with vsw_*datapath_id_in_hex* (e.g. vsw_0xabc)
  - ovs-ofctl show vsw_0xabc

Look into internals?

- Tremashark allows to visualize various events including IPC events when you need to inspect events among/on Trema core modules and applications

- How to use it?
  - Build and install wireshark plugin (packet-trema.so)
  - Run trema application with ‘use_tremashark’ option (in configuration file)
  - Send SIGUSR2 to processes to be monitored
    - $ kill –USR2 `cat tmp/dumper.pid`

- See trema/src/tremashark/README for details

Want to run with network namespace or mininet?

- Trema's network emulator creates virtual Ethernet devices (veth) which connect between switches and hosts. The devices are named as trema-*X*-*Y*
- You can attach trema-*X*-*Y.* to the namespaces with "ip link set trema-*X*-*Y* netns *PID*" as usual

Want to more interactively play with the Trema world?

- Just run ‘trema’ command without any arguments ! Then you can enter to the Trema Shell that is based on irb (Interactive Ruby)