

This presentation slide is available at  
<http://trema.github.com/trema/>  
or google “Trema”

# Trema updates and introduction to GENI researchers

HIDEyuki Shimonishi  
Trema development team  
Nov. 4, 2011

# NEC's OpenFlow activities related to GENI

- “ProgrammableFlow”
  - Production quality OpenFlow switches and controller for data center networks
- “Trema”
  - OpenFlow controller platform for research and academia
  - Free software (GPLv2)
  - Originally developed by NEC research lab.



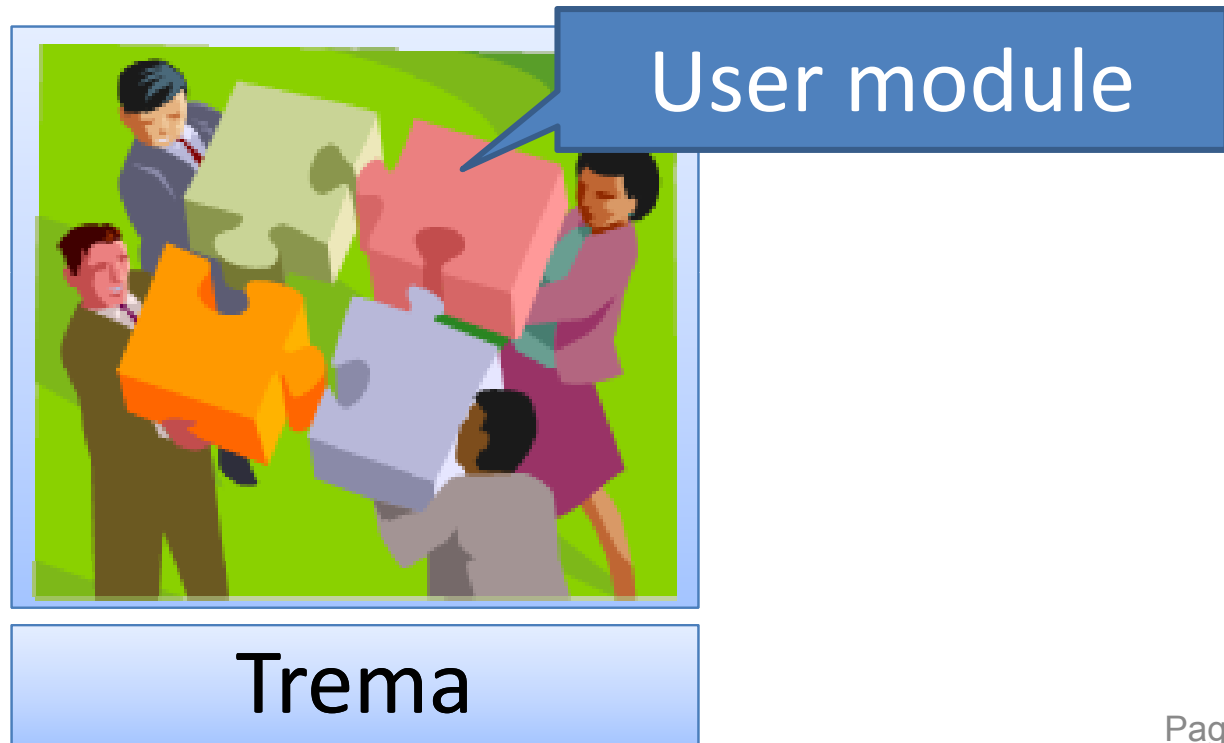
# Goal of Trema project

- Provide fairly good quality OpenFlow controller platform to researchers and developers
  - Continuous development, maintenance, bug-fixes and user support from the project team (researchers and professional programmers at NEC and other companies)
- Help GENI experimenters for their OpenFlow related activities :-)



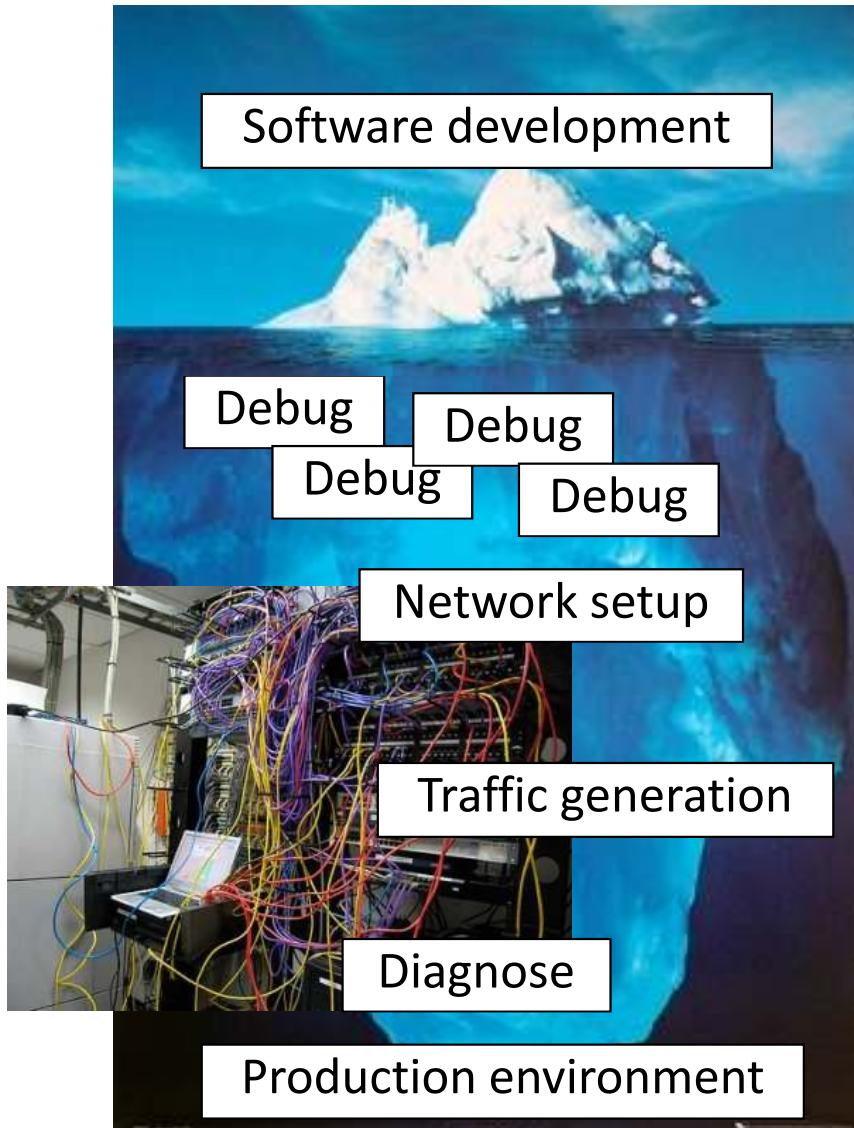
# IMPORTANT goal of Trema project

- Researchers develop their own controllers on top of Trema and contribute to the community
  - Recycling controller modules accelerates our research activities
  - Independent repository @ <https://github.com/trema/apps>



# Motivation

## OpenFlow iceberg



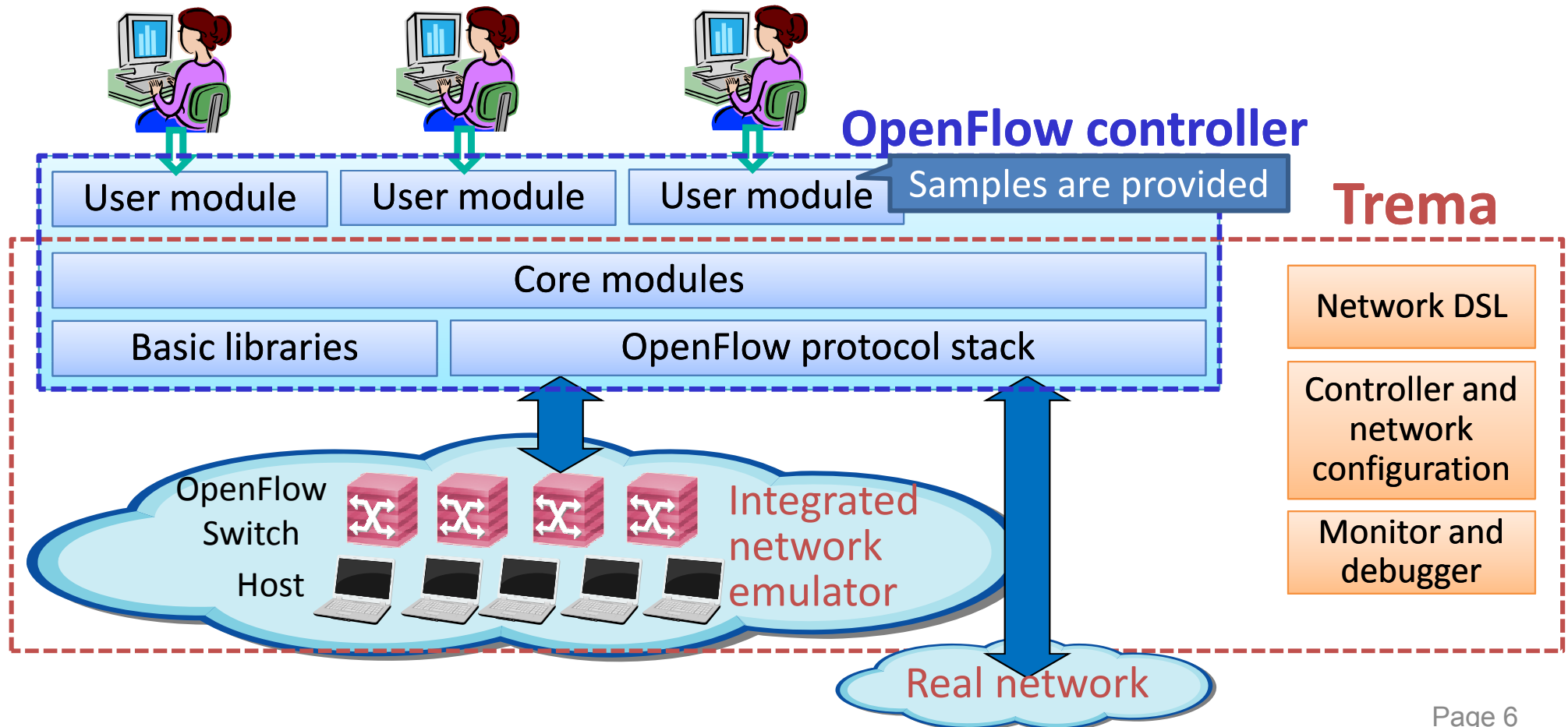
### Scope of Tremas

Trema is an OpenFlow platform for entire development process (like Ruby on Rails)

- ***Shorter development cycle***
- ***Reduce labor cost***
- ***More and more research outputs :-)***

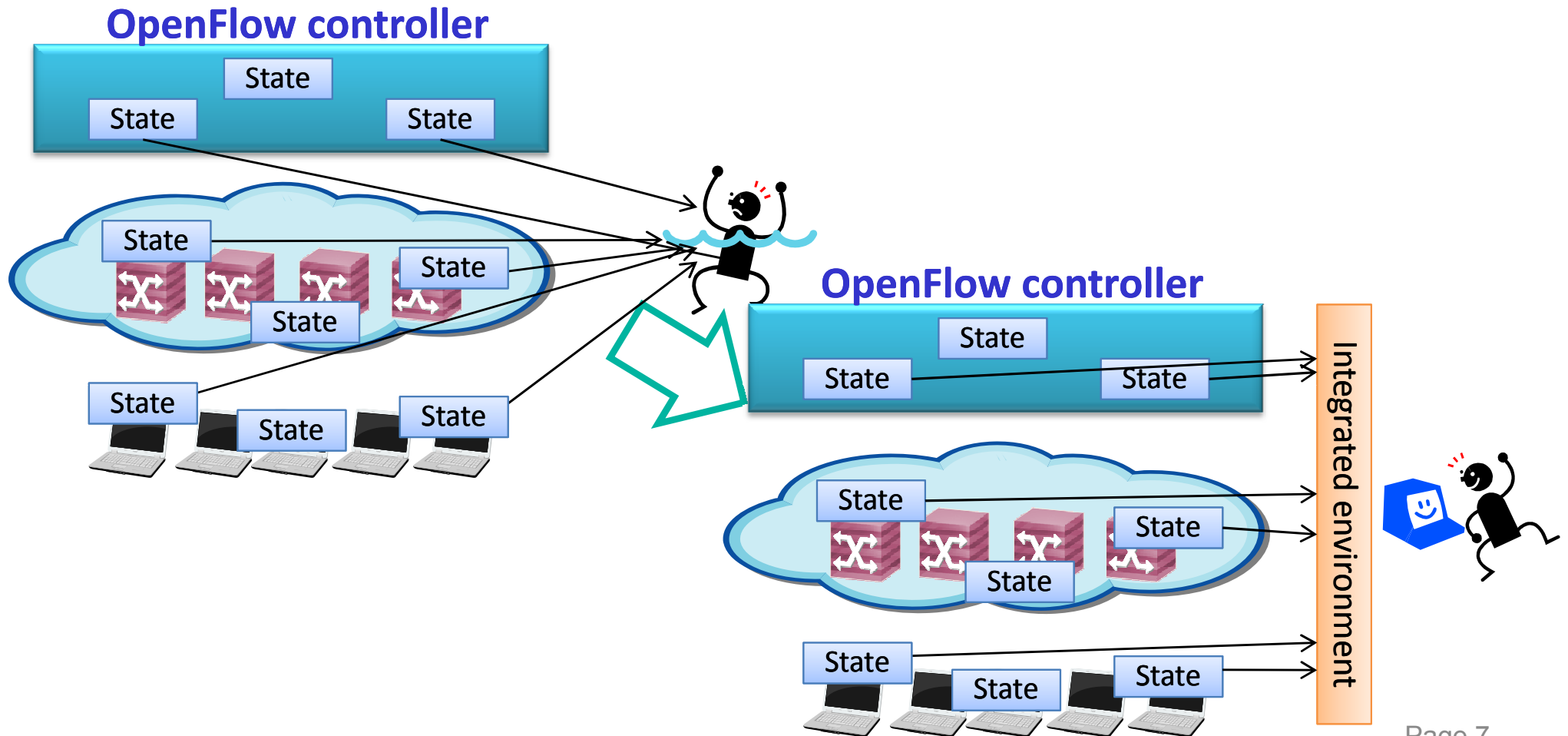
# Scope of Trema

- OpenFlow controller = Trema + user modules
- Trema = OpenFlow controller platform + integrated testing and debugging environment



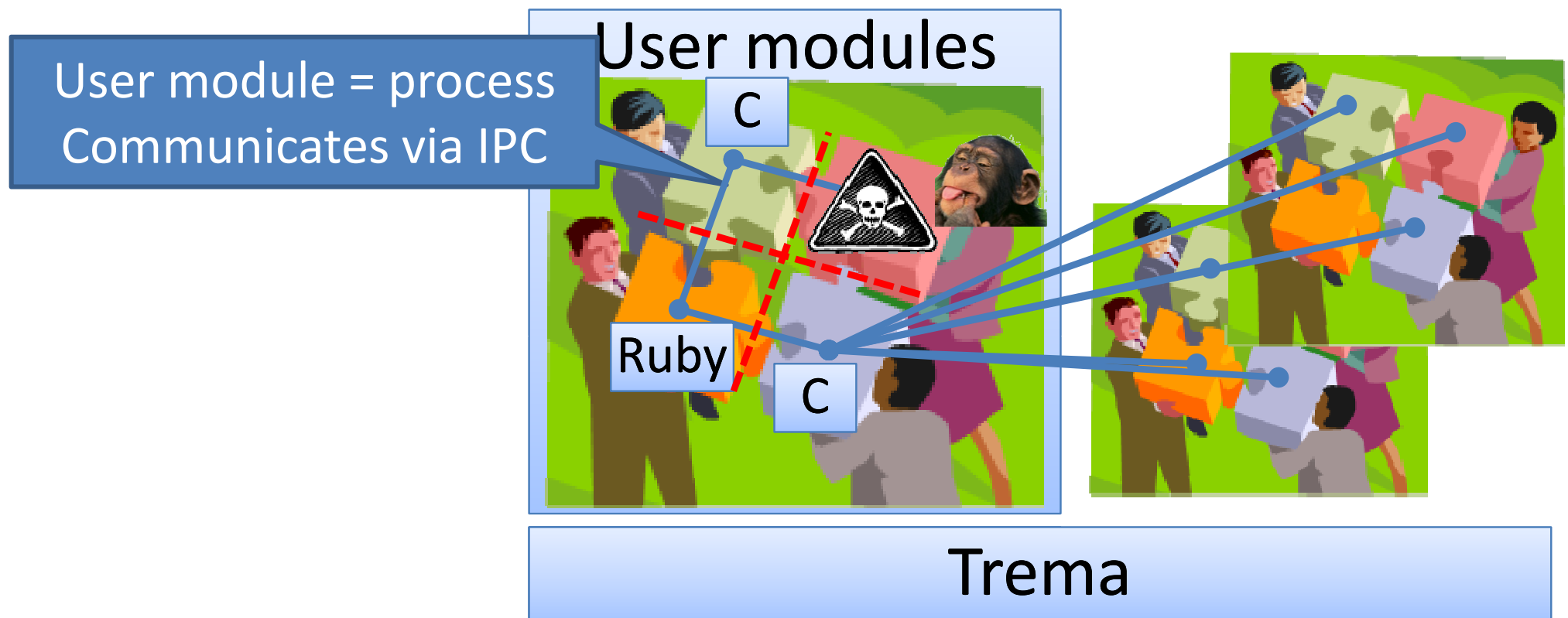
# Integrated testing and debugging environment

- Network programming is essentially distributed programming
- Trema provides a system support to manage, monitor, and diagnosis entire system



# Multi-process modular architecture

- Extensibility and stability
- User modules written in C or Ruby
- Can be extended to distributed controller





# How to get started

1. Trema tutorial @<http://trema.github.com/trema/>
  1. Download and build – quite easy
  2. Try examples – quite easy

# How to start your own project

Backup

1. Create your controller
  1. Develop your controller from scratch or modifying existing 3rd party user modules
  2. Combine 3rd party user modules with your controller
2. Create your network environment
  1. Configure emulated network
  2. And/or connect real OpenFlow switches
3. Test your system
  1. Operate OpenFlow controller and emulated network with Trema
4. Deploy to GENI infrastructure and enjoy !

Trema network DSL helps all these steps

# How to learn to program

- Learn from tutorials
  - <http://trema.github.com/trema>
  - <https://github.com/trema/trema/wiki>
- Learn from examples
  - <https://github.com/trema/trema/tree/develop/src/examples>
  - Copy-and-paste the code for your own modules
- Learn from actual use cases (Trema Apps)
  - <https://github.com/trema/apps>
  - Many projects use and modify “Routing Switch”

# Meta

- Web: <http://trema.github.com/trema/>
- ML: [trema-dev@googlegroups.com](mailto:trema-dev@googlegroups.com)
- Repository
  - Trema : <https://github.com/trema/trema>
  - User modules (Trema Apps): <https://github.com/trema/apps>
- Tutorial
  - <http://trema.github.com/trema>
  - <https://github.com/trema/trema/wiki>
- API document
  - Ruby: <http://rubydoc.info/github/trema/trema/master/frames>

Recent activities

# C and Ruby support completed

- Complete set of APIs available for both C and Ruby programmers
  - OpenFlow 1.0.0, packet parser, logging, timer, messenger, etc.
  - Different modules can use different languages
    - Ruby's productivity and C's performance
  - C and Ruby examples
    - <https://github.com/trema/trema/tree/develop/src/examples>
  - Ruby API documents
    - <http://rubydoc.info/github/trema/trema/master/frames>

# Ruby support – an example

- Writing repeater-hub emulation code with Ruby

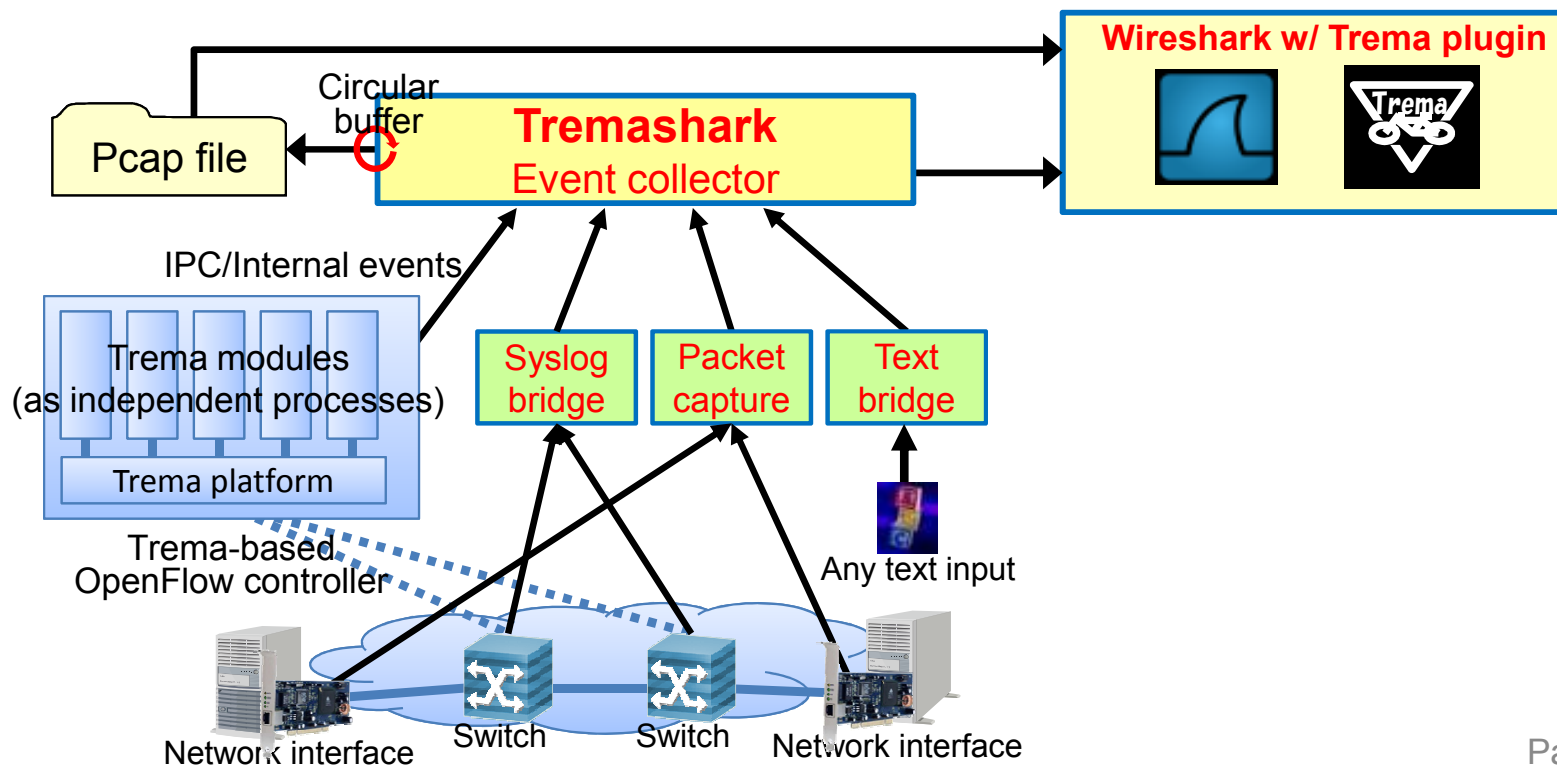
```
class RepeaterHub < Trema::Controller  ←—— Create new controller class
  def packet_in datapath_id, message  ←—— Define a handler for packet_in event
    send_flow_mod_add(
      datapath_id,
      :match => Match.from( message ),
      :actions => Trema::ActionOutput.new( OFPP_FLOOD )
    )
    send_packet_out(
      datapath_id,
      :buffer_id => message.buffer_id,
      :actions => Trema::ActionOutput.new( OFPP_FLOOD ),
      :data => message.buffered? ? nil : message.data
    )
  end
end
```

Send flow\_mod

Send packet\_out

# Tremashark

- Collects/monitors various events in a single verifiable point
  - IPC/Internal events in Trema-based OpenFlow controller, log outputs, packet exchanges, etc
- Stores events in a circular buffer to monitor running system
- Powerful event filtering function leveraging Wireshark and its extensibility





# Tremashark – screen shot

The screenshot displays the Tremashark application window. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Tools, and Help. Below the menu is a toolbar with various icons. The main window is divided into two panes. The top pane shows a list of events with columns: No., Time, Source, Destination, Protocol, and Info. The bottom pane shows a detailed view of the selected event (Frame 84).

No.	Time	Source	Destination	Protocol	Info
110	12:23:23.036781	192.168.0.2	192.168.0.1	TREMA+OFF+UDP	switch.abc > learning_switch (Sent) => Packet In (AMI) (BufID=275) (82B)
111	12:23:23.037153	192.168.0.2	192.168.0.1	TREMA+OFF+UDP	learning_switch (Received) => Packet In (AMI) (BufID=275) (82B) => Source
112	12:23:23.037608	learning_switch	switch.abc	TREMA+OFF	learning_switch > switch.abc (Sent) => Packet Out (CSMI) (BufID=275) (24B)
113	12:23:23.037758		switch.abc	TREMA+OFF	switch.abc (Received) => Packet Out (CSMI) (BufID=275) (24B)
114	12:23:23.037843	127.0.0.1	127.0.0.1	TREMA+OFF	lo (Packet Capture) => Packet Out (CSMI) (BufID=275) (24B)
115	12:23:23.037905	127.0.0.1	127.0.0.1	TREMA+TCP	lo (Packet Capture) => 49623 > 6633 [ACK] Seq=2171394473 Ack=2161533291
116	12:23:27.552783	192.168.0.2	192.168.0.1	TREMA+OFF+UDP	lo (Packet Capture) => Packet In (AMI) (BufID=276) (82B) => Source port:
117	12:23:27.553115	192.168.0.2	192.168.0.1	TREMA+OFF+UDP	switch.abc > learning_switch (Sent) => Packet In (AMI) (BufID=276) (82B)
118	12:23:27.553155	192.168.0.2	192.168.0.1	TREMA+OFF+UDP	learning_switch (Received) => Packet In (AMI) (BufID=276) (82B) => Source
119	12:23:27.553274	learning_switch	switch.abc	TREMA+OFF	learning_switch > switch.abc (Sent) => Packet Out (CSMI) (BufID=276) (24B)
120	12:23:27.553338		switch.abc	TREMA+OFF	switch.abc (Received) => Packet Out (CSMI) (BufID=276) (24B)
121	12:23:27.553418	127.0.0.1	127.0.0.1	TREMA+OFF	lo (Packet Capture) => Packet Out (CSMI) (BufID=276) (24B)
122	12:23:27.553457	127.0.0.1	127.0.0.1	TREMA+TCP	lo (Packet Capture) => 49623 > 6633 [ACK] Seq=2171394555 Ack=2161533315
123	12:23:36.122365		syslog_relay	TREMA+Syslog	(Syslog) => USER.NOTICE: Sep 15 12:23:36 axtest logger: Hello Tremashark.

The bottom pane shows the details of Frame 84: 138 bytes on wire (1104 bits), 138 bytes captured (1104 bits) on DLT: 147. The event is a Tremashark Unified Event Dumper. The dump header shows: Type: Packet Capture (10), Event occurred at: Sep 15, 2011 12:22:41.731927000 JST, Application name length: 3, Service name length: 15, Data length: 102, Application name: lo, Service name: packet\_capture. The PCAP dump header shows: Datalink: Ethernet (10Mb) (11), Interface: lo. The PCAP packet header shows: Captured at: Sep 15, 2011 12:22:41.731927000 JST, Capture length: 66, Frame length: 66. The packet details show: Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00), Internet Protocol, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1), Transmission Control Protocol, Src Port: 49623 (49623), Dst Port: 6633 (6633), Seq: 2171393823, Ack: 2161533123, Len: 0.

- IPC events
- Packet capture
- IPC events
- Packet capture
- Syslog message

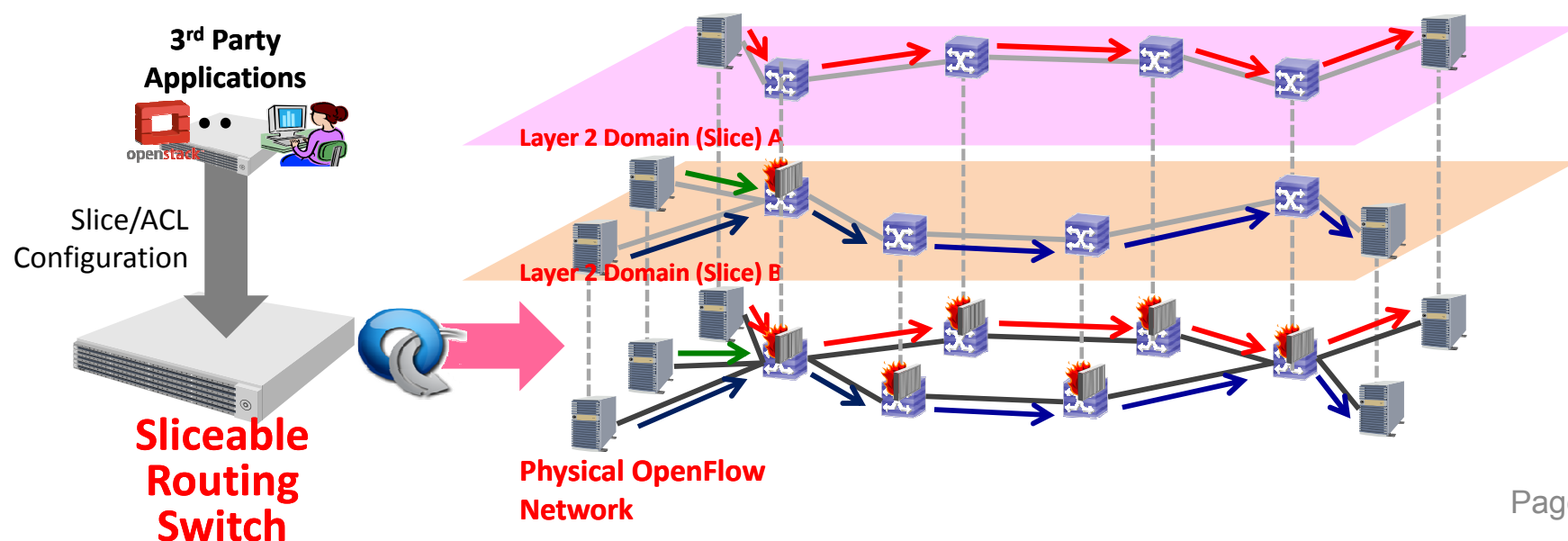
*NEWS!*

*New Trema-based controller  
“Sliceable Routing Switch”  
is available now @Trema Apps*

*[https://github.com/trema/apps/tree/master/sliceable\\_routing\\_switch](https://github.com/trema/apps/tree/master/sliceable_routing_switch)*

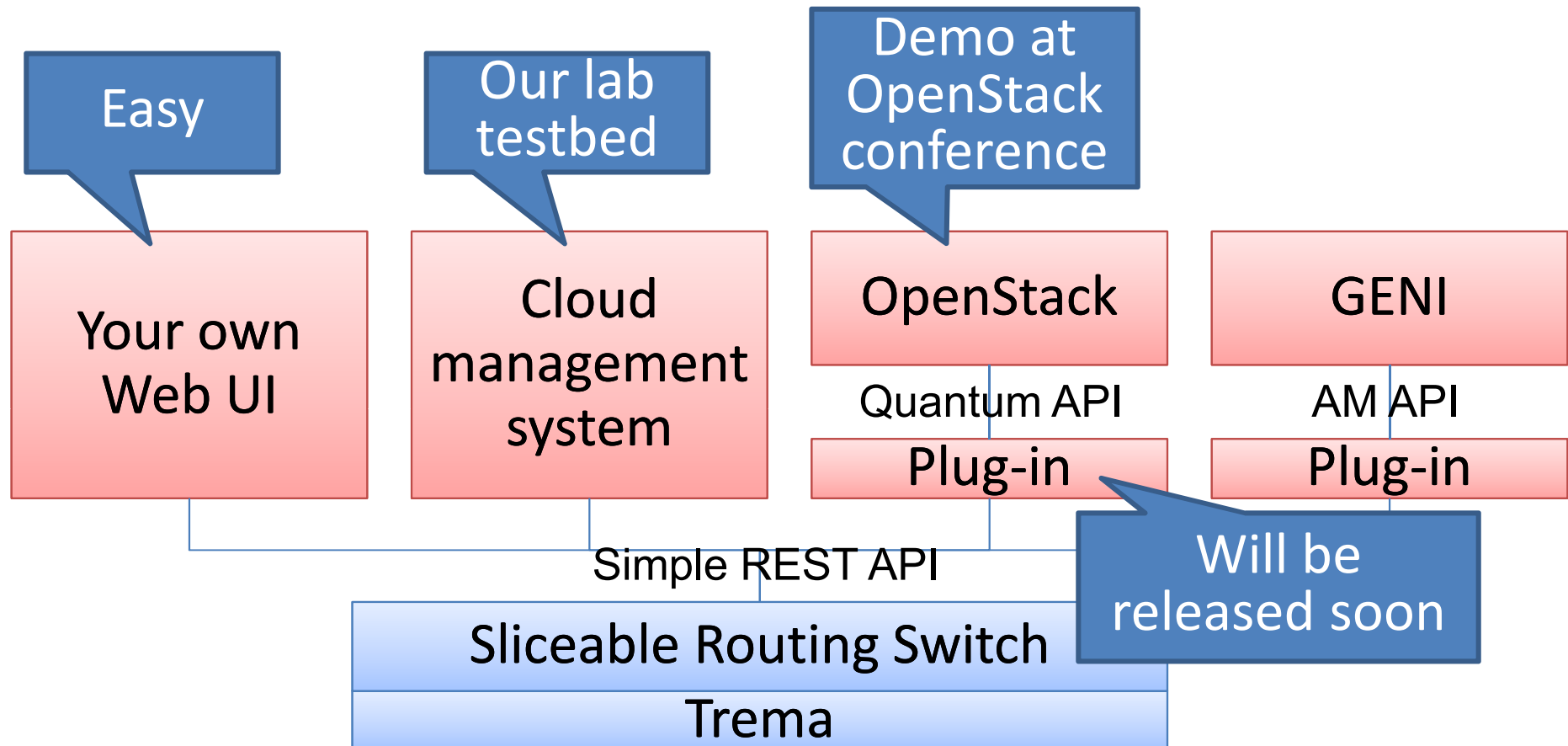
# Sliceable Routing Switch

- Focused on most commonly used slicing (in cloud computing)
  - Virtual flat L2 network domains + L1-4 access control list
  - No VLAN ID dependency - no 4K limitation
- Very simple REST-API to manage the slices
  - Create slice with just only slice name
  - Attach host by port or MAC address
- Modify for your own projects
  - We use this for our own production network, but production quality/performance is NOT guaranteed



# Sliceable Routing Switch – use case

- Virtual L2 network service on your local network
- Shared L2 network pool for backbone network
- Private cloud system (with OpenStack or whatever)



- Creates flat layer 2 domains with multiple OpenFlow switches
  - Topology discovery using LLDP
  - Shortest path setup for L4 flows (work with loop topologies)
- Slicing function for creating multiple layer 2 domains
  - Port and MAC-based host-to-slice bindings
  - Several operation modes for broadcast handling
- Global access control list (ACL) function for filtering traffic within a slice
  - Layer 1 – 4 filters for each incoming L4 flows

# Sliceable Routing Switch – REST APIs

Backup

Date: 2011/10/3, Revision: 0.05

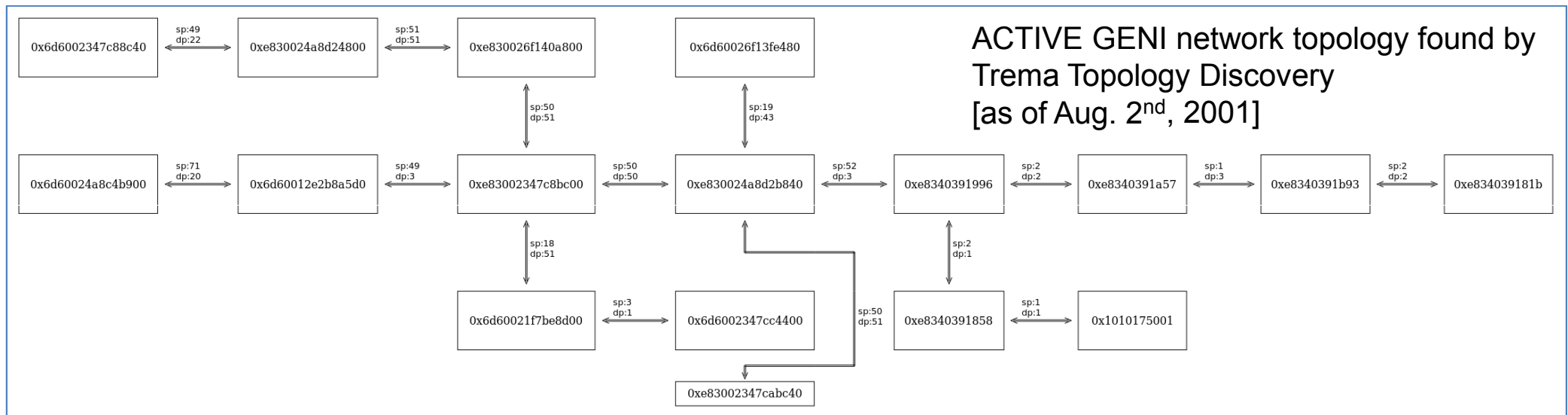
Path	Method	Behavior
/tenants/<tenant_id>/networks	GET	List summary of networks associated with a tenant.
	POST	Create a new network associated with a tenant.
/tenants/<tenant_id>/networks/<net_id>	GET	Show details of network identified by <i>net_id</i> .
	PUT	Update details of network identified by <i>net_id</i> .
	DELETE	Remove the network identified by <i>net_id</i> .
/tenants/<tenant_id>/networks/<net_id>/ports	GET	List all the ports currently defined.
	POST	Create a logical port on the network specified in the request URI.
/tenants/<tenant_id>/networks/<net_id>/ports/<port_id>	GET	Retrieve detail of the port <i>port_id</i> configured for the network <i>net_id</i> .
	DELETE	Remove a port from the network.
/tenants/<tenant_id>/networks/<net_id>/ports/<port_id>/attachments	GET	List complete information about the attachment currently plugged into the specified port.
	POST	Plug resources into a logical port on a virtual network.
/tenants/<tenant_id>/networks/<net_id>/ports/<port_id>/attachments/<attachment_id>	GET	Show information about the attachment specified by the request URI.
	DELETE	Remove the attachment specified by the request URI.
/tenants/<tenant_id>/networks/<net_id>/attachments	GET	Retrieve all the resources currently attached to a network.
	POST	Plug resources into a virtual network.
/tenants/<tenant_id>/networks/<net_id>/attachments/<attachment_id>	GET	Show information about the attachment specified by the request URI.
	DELETE	Remove the attachment specified by the request URI.

## Other Trema use cases

- Wakame-vdc (cloud management software)
  - <http://wakame.jp/>
- Radware demonstration at Open Networking Summit
  - Anomaly traffic detection
- 10 controllers on Trema Apps
  - <https://github.com/trema/apps>
- 10+ experimental controllers in NEC
  - Operating at our production network more than 6 month
- NICT and several Japanese universities
- Other projects may be happening (if so, please let us know)

# Other Trema use cases

- Tests at GENI
  - @ GPO lab OpenFlow testbed (TangoGENI)
  - Several Trema-based controllers tested
    - Learning Switch, Routing Switch, Topology Discovery and Topology Viewer, Flow Statistics Monitor





# WANTED

- User module developers and experimenters
  - Experiments at GENI infrastructure
  - Share your modules with others
- Questions and comments to Trema mailing list
  - [trema-dev@googlegroups.com](mailto:trema-dev@googlegroups.com)
- Contributors to Trema (core) development

This presentation slide is available at  
<http://trema.github.com/trema/>