Tutorial on OpenFlow

Fed4FIRE – GENI Research Experiment Summit (FGRE 2014)

Carlos Bermudo, Oscar Moya, Frederic Francois

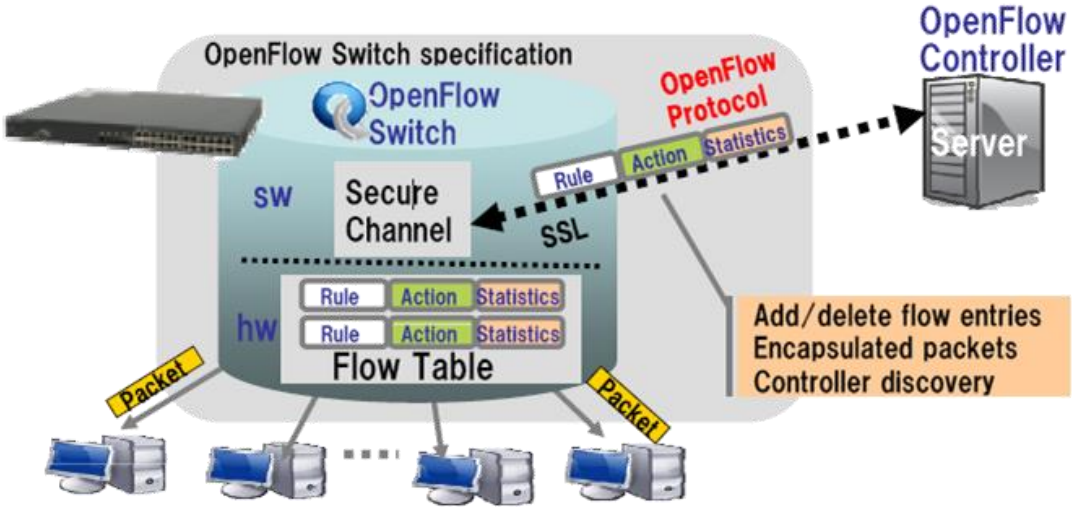
This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 318389

July 9th 2014

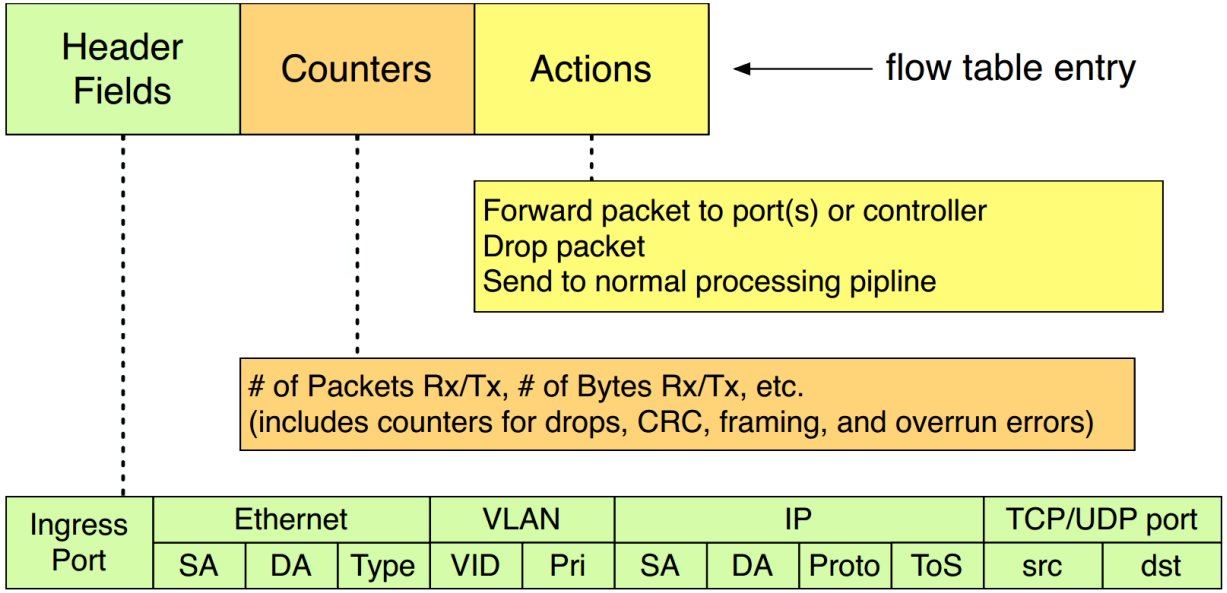
Introduction

- Refreshing OF and Optical OF concepts
- OF and OOF slicing
- OpenFlow testbeds
- Running an OF experiment
- POX tricks & tips
- OOF Demo

Definition: OpenFlow



- Characteristics:
  - Software controller
  - Secure channel with devices
  - Matching on OF packet headers



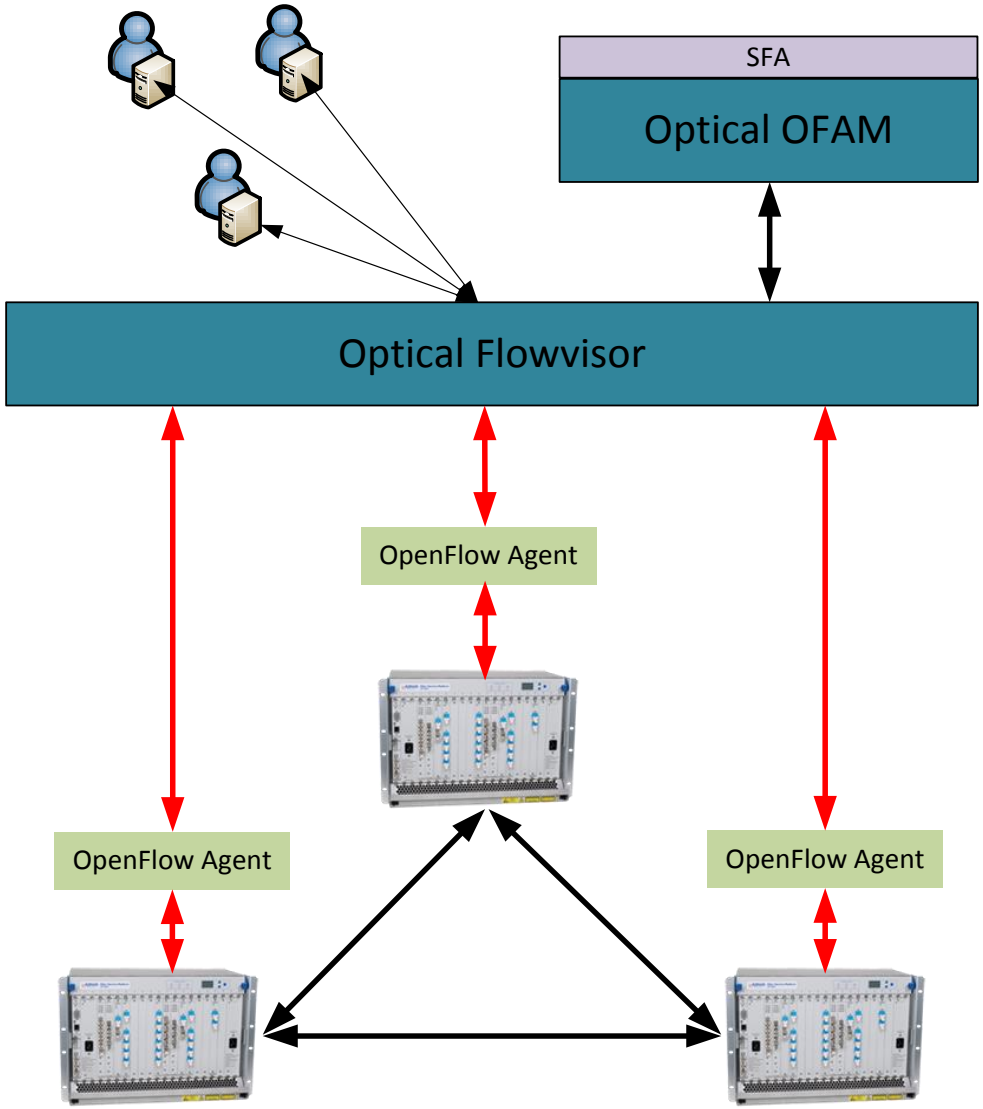
Definition: Optical OpenFlow

Objectives

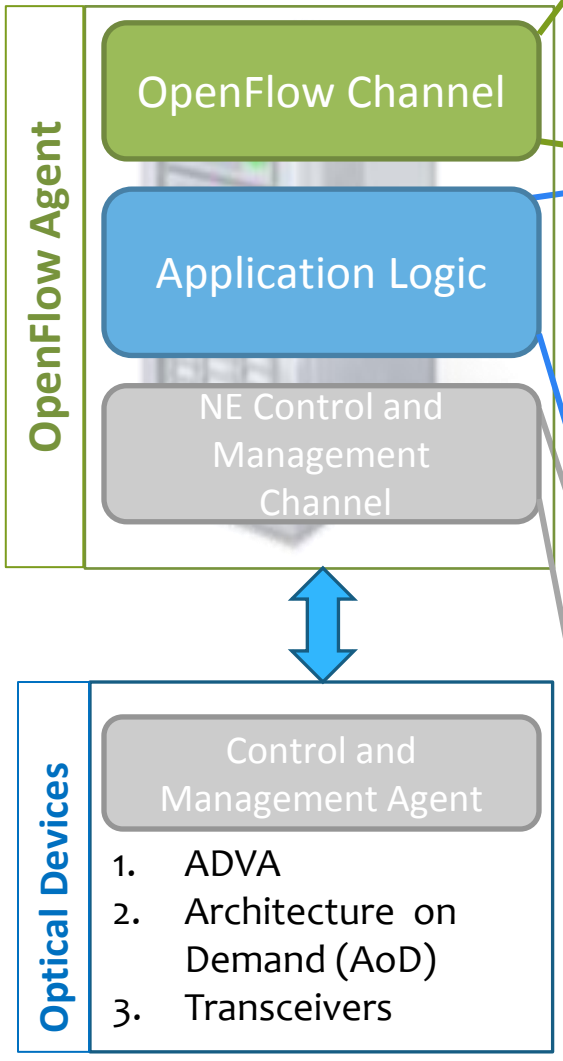
1. Enable the control of optical switches through the OpenFlow Protocol (currently using OF v1.0 with circuit extensions v0.3)

2. A single OpenFlow controller to control both optical and packet switches with the resulting benefits:
    a) abstraction of the whole network to enable simpler network management
    b) seamless coordination of packet and optical networks

Optical OpenFlow: Architecture



Optical OpenFlow: Agents

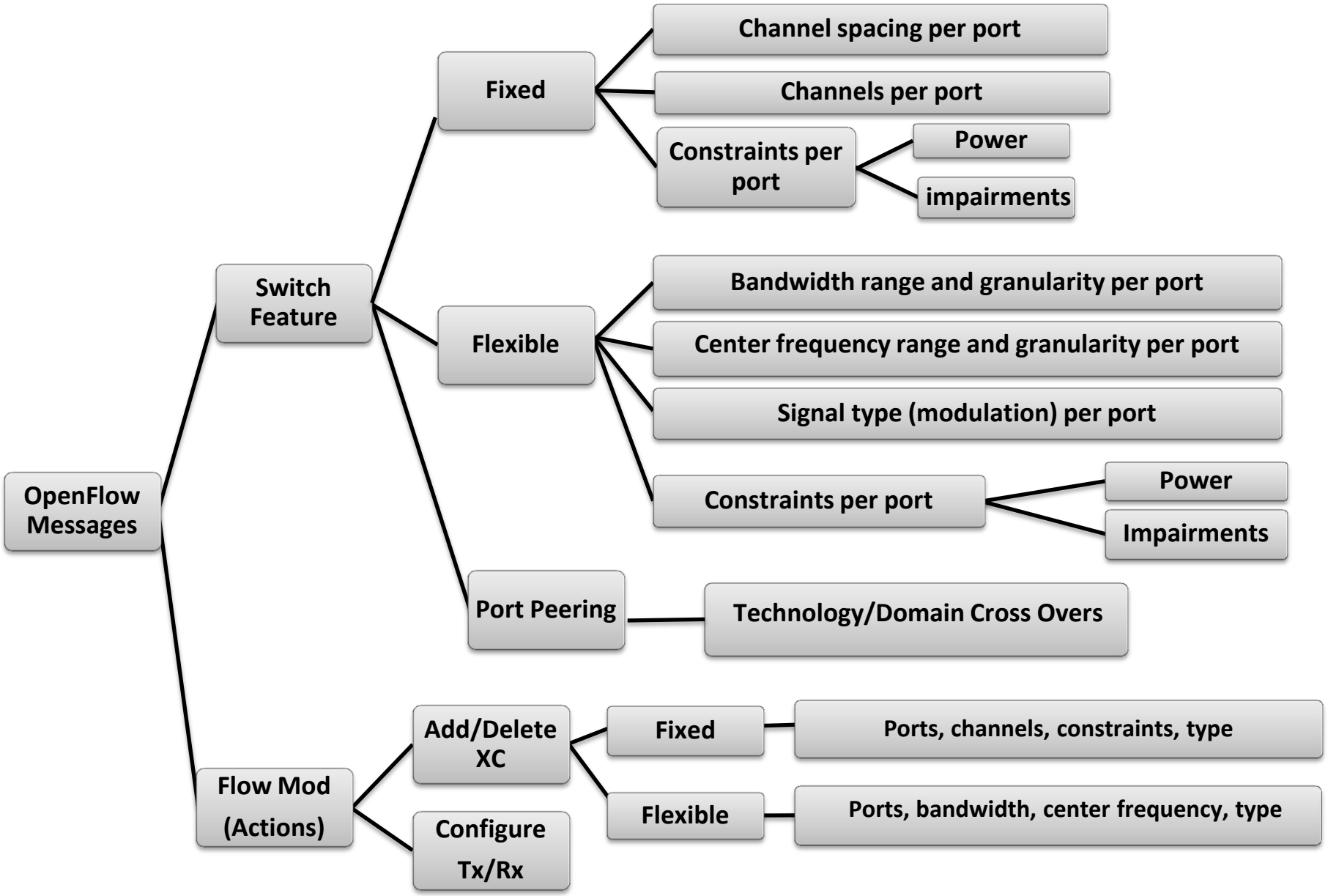


- ☑ Maintains a secure OpenFlow connection with the OpenFlow controller
- ☑ Acts as proxy between the OpenFlow controller and the Application Logic

- ☑ Implements a Resource Model to map the proprietary device feature description to the standardized OpenFlow description, i.e. map Device specific syntax to OpenFlow syntax
- ☑ Resource Model enables the retrieval of optical device description and the configuration of the optical device
- ☑ The functionality constraints of the optical device can be derived from the Resource Model description

- ☑ Network Equipment control and management channel communicates with its counterpart agent on the optical device to retrieve management information and enable configuration. The channel is specific to the type of device being controlled

Optical OpenFlow: Extensions



Software: OpenFlow controllers' overview

- Platform / Framework that allow experimenters to develop components / applications to control the behavior of their data flows

- **NOX**, **POX**
  - NOX: 1st OpenFlow controller
  - POX: high-level SDN API
  - Supported protocol(s): OpenFlow 1.0
  - Supported language(s): C++ (NOX), Python (POX)
  - *https://github.com/noxrepo/nox-classic/wiki/Developing-in-NOX*
  - *https://openflow.stanford.edu/display/ONL/POX+Wiki*

- **Ryu**
  - Open-sourced Network Operating System (NOS)
  - Supported protocol(s): OpenFlow 1.0, 1.3, 1.4
  - Supported language(s): Python
  - *http://ryu.readthedocs.org/en/latest/writing_ryu_app.html*

http://yuba.stanford.edu/~casado/of-sw.html

Software: Flow slicing

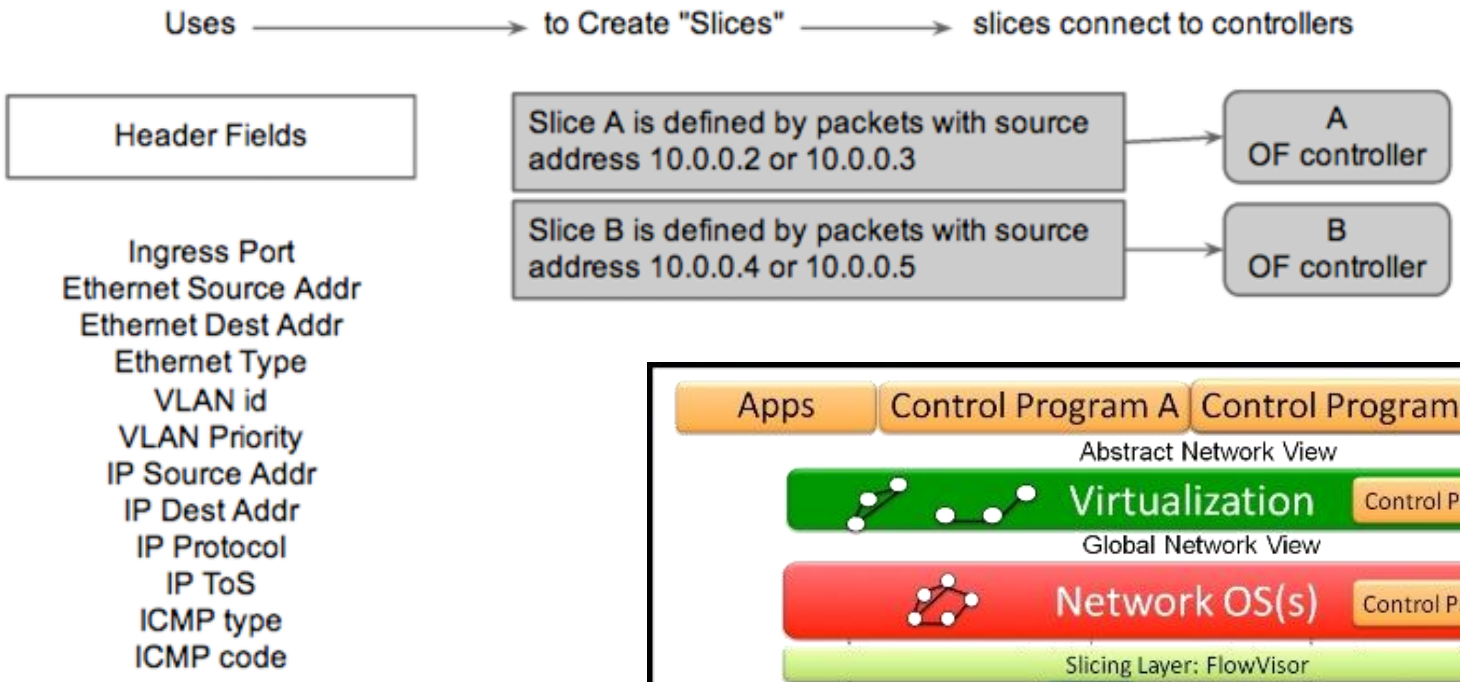
- ***The problem***
  - Each experiment generates a set of data flows
  - Multiple experiments running at the same time
  - Different data flows (experiments) must be isolated

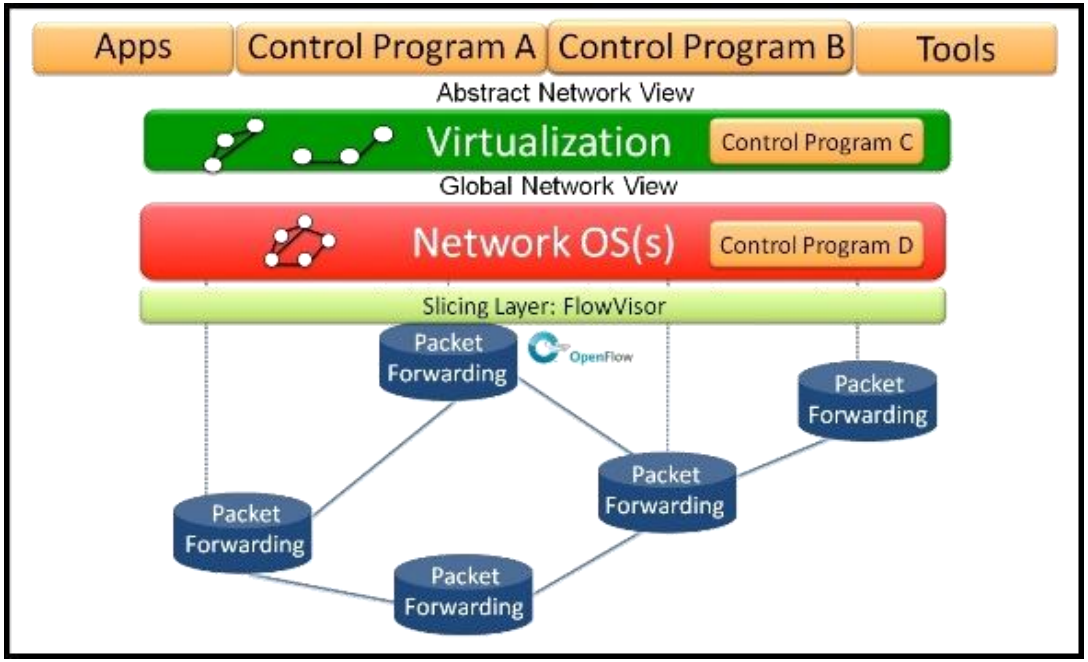
- ***FlowVisor***

  Special purpose OpenFlow controller that acts as a transparent proxy between OpenFlow switches and multiple OpenFlow (user) controllers.

Software: Flow slicing

- FlowVisor





Software: Optical Flowvisor & OFAM

Flowvisor has been extended:

1. To support new OpenFlow messages required for the optical devices.
2. To enable its Slicer component to slice according to optical ports and wavelengths.

OFAM has been extended to support the reservation of optical switches through SFA.

Advertisement RSpecs Snippet

```
<openflow:datapath
   component_id="urn:publicid:IDN+optical:openflow:ofam:univbris+datapath+00:00:00:00:0a:21:00:0a"
   component_manager_id="urn:publicid:IDN+optical:openflow:ofam:univbris+authority+cm" dpid="00:00:00:00:0a:21:00:0a"> ……
<openflow:link
component_id="urn:publicid:IDN+optical:openflow:ofam:univbris+link+00:00:00:00:0c:21:00:0a_1_00:00:00:00:0b:21:00:0a_3">
<openflow:wavelength value="193.8"/>
</openflow:link>
```

Optical keyword mean this is an optical switch

Frequency supported in link

Software: Optical Flowvisor & OFAM

A new flowspace definition must be added to the reservation RSpecs to support optical flowspaces.

Reservation RSpecs Snippet 1

```
<openflow:group name="fs1">
  <openflow:datapath
      component_id="urn:publicid:IDN+optical:openflow:ofam:univbris+datapath+00:00:00:00:0a:21:0
0:0a"
      component_manager_id="urn:publicid:IDN+optical:openflow:ofam:univbris+authority+cm”
 dpid="00:00:00:00:0a:21:00:0a">
          <openflow:port name="MOD-5-9" num="1001“/>
  </openflow:datapath>
</openflow:group>
```

New OpenFlow groups should be defined to contain **only** optical OpenFlow switches and ports. It is possible to add OpenFlow groups for packets in the same reservation RSpecs but these groups should **not** contain optical switches and ports.

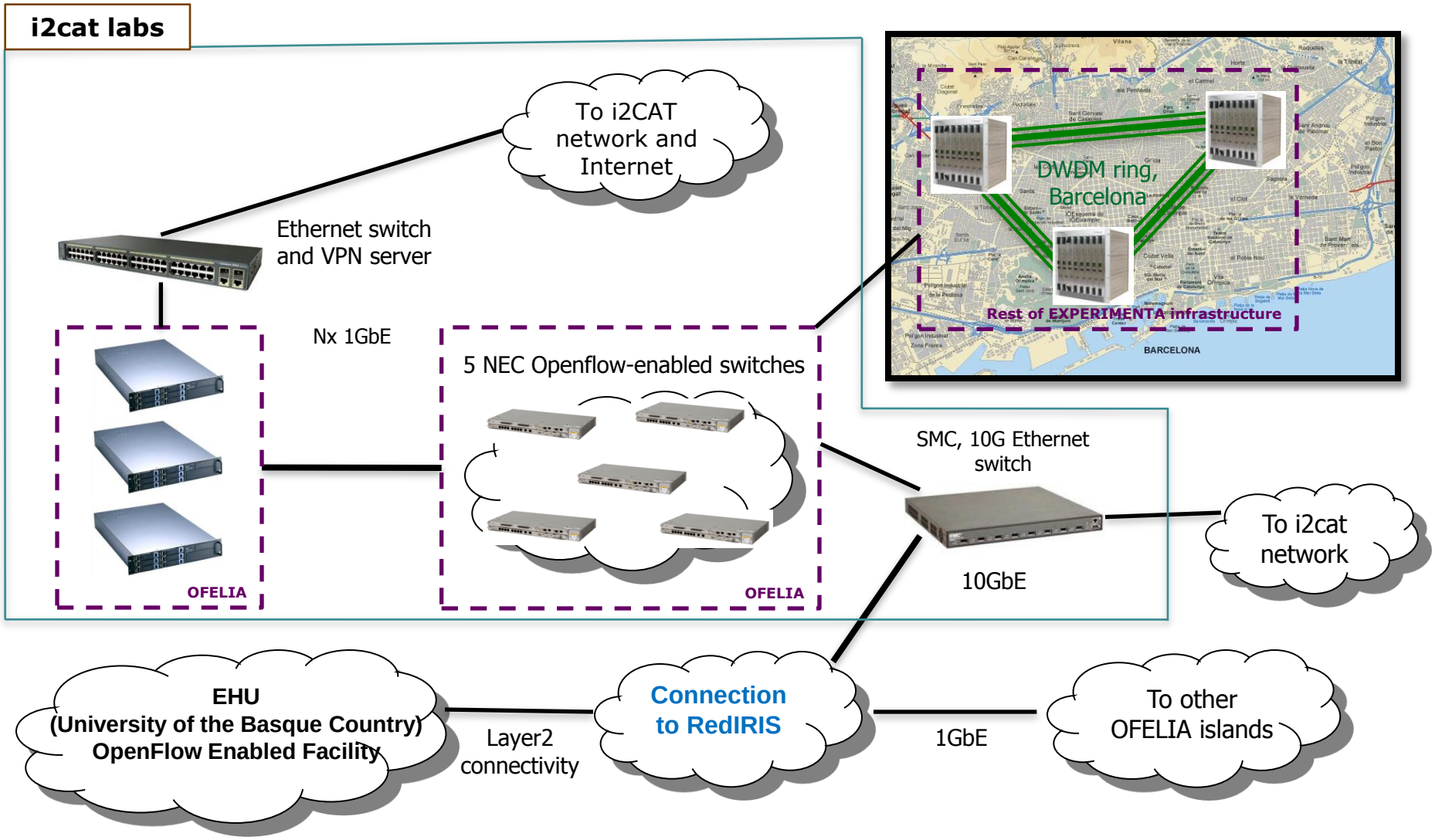
Reservation RSpecs Snippet 2

```
<openflow:match>
   <openflow:use-group name="fs1" />
      <openflow:circuit>
          <openflow:wavelength name = “193.8”/>
      </openflow:circuit>
</openflow:match>
```

OpenFlow matches can now be specified over the optical OpenFlow group that were defined above. For example, the wavelength(s) specified in snippet 2 will be reserved over the whole OpenFlow group “fs1”, i.e. in all the ports of the switches specified in snippet 1.

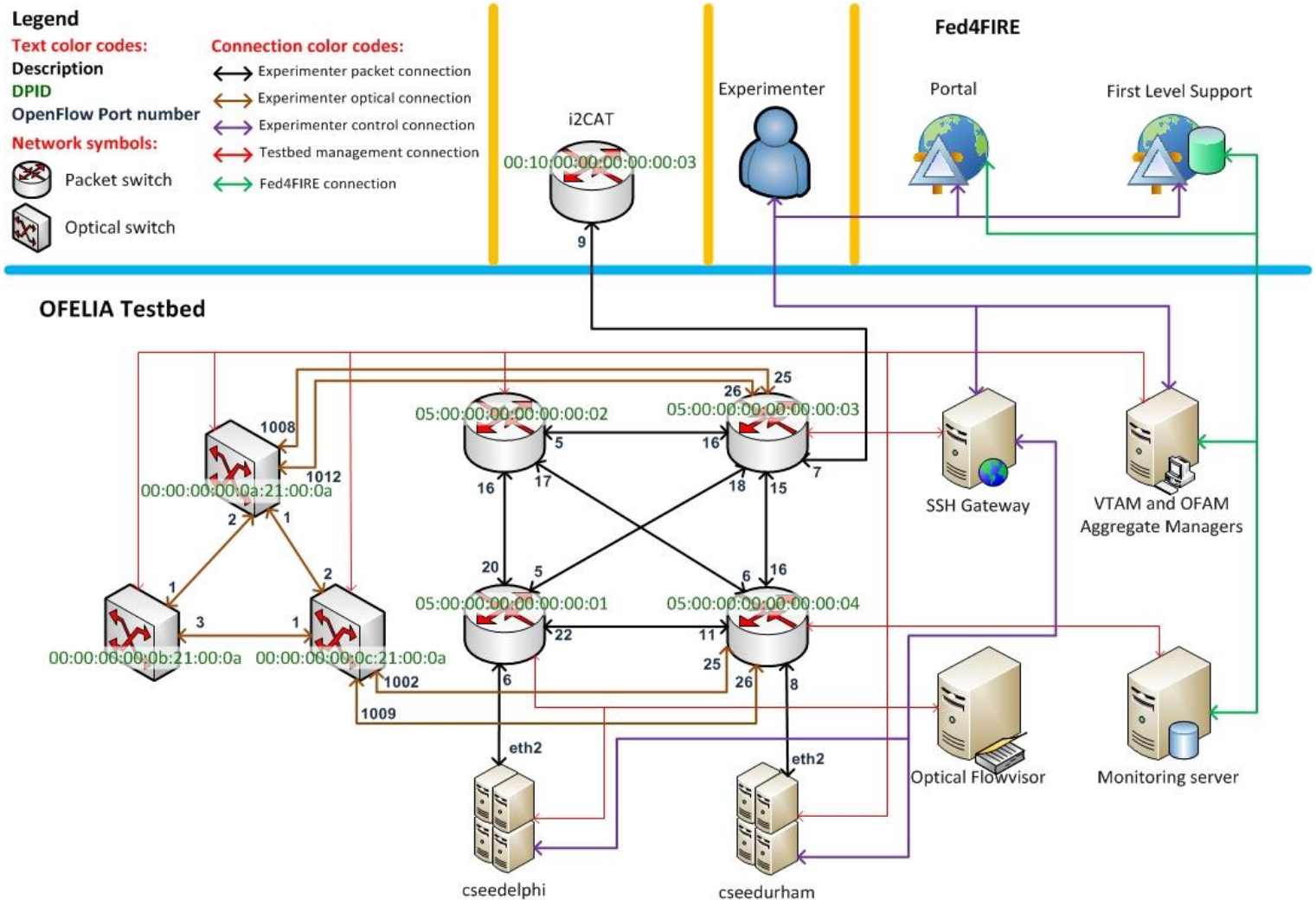
OpenFlow testbeds: i2CAT

5 x NEC IP8800/S3640-24T2XW, 24 ports (1/10GE)
3 x dedicated physical servers



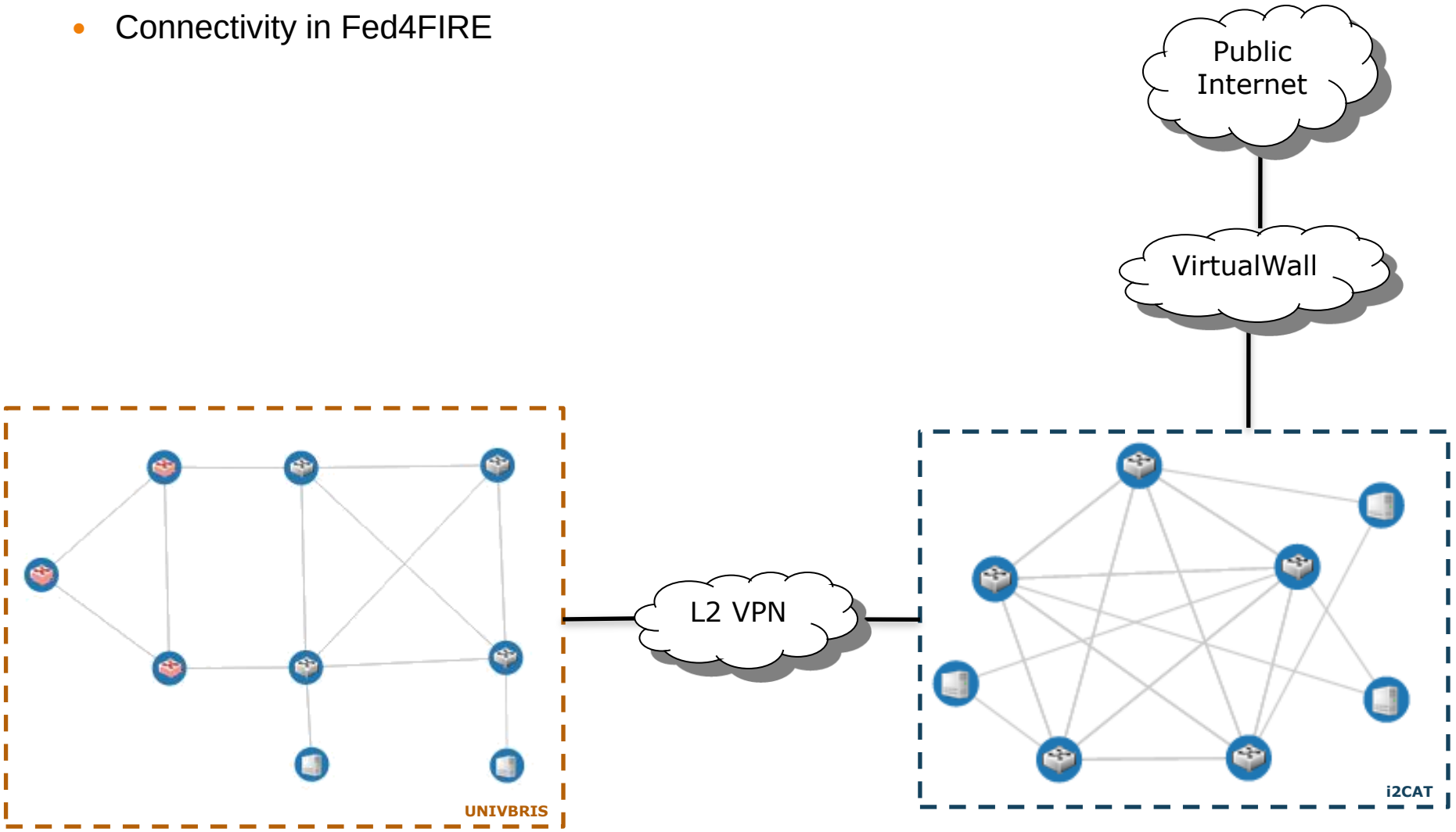
OpenFlow testbeds: UNIVBRIS

4 x Packet Switches (NEC IP8800)
2 x dedicated virtualization servers
3 x fixed WDM optical switches (ADVA FSP3000)



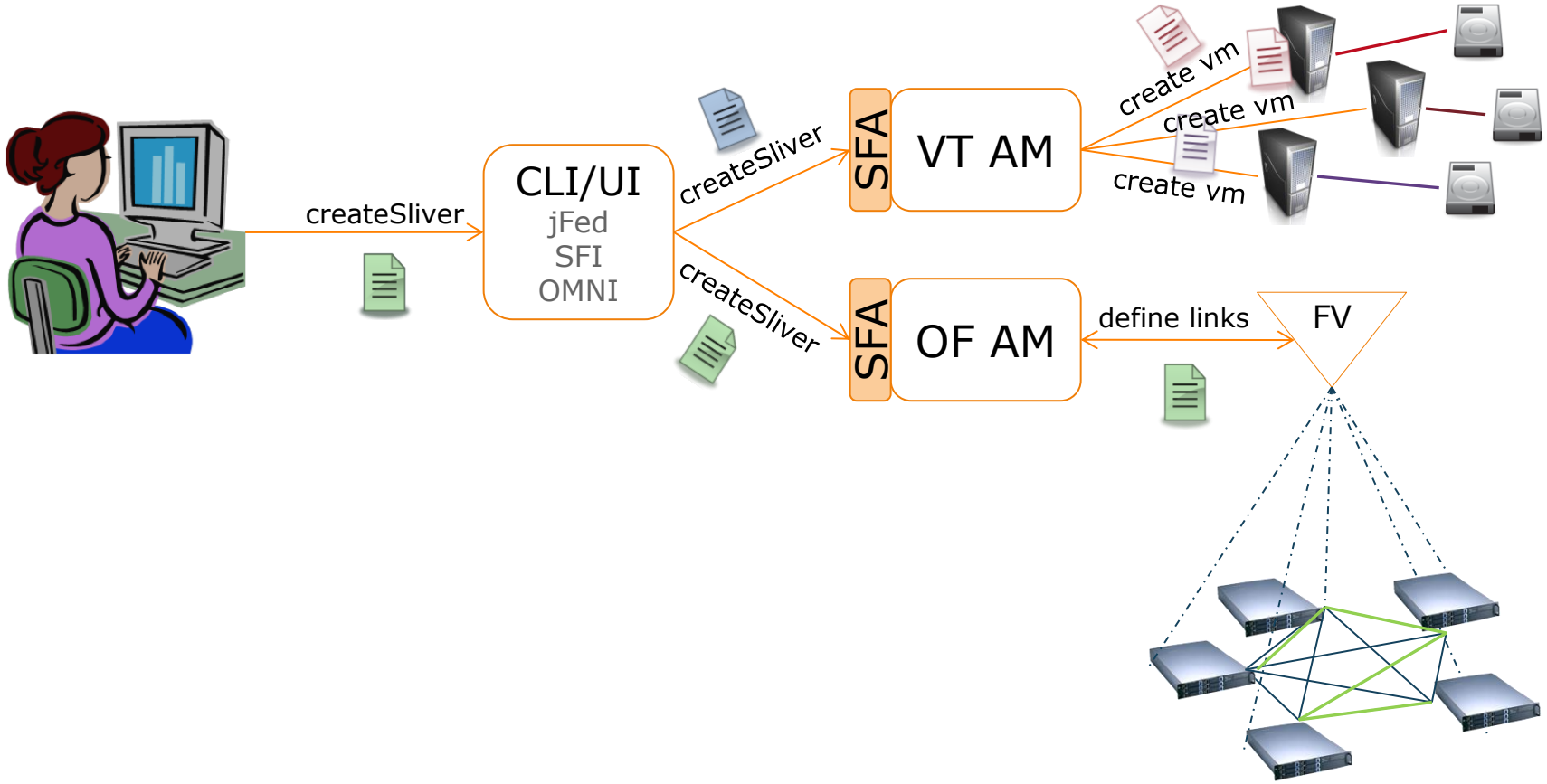
OF testbeds in federation environments

- Connectivity in Fed4FIRE



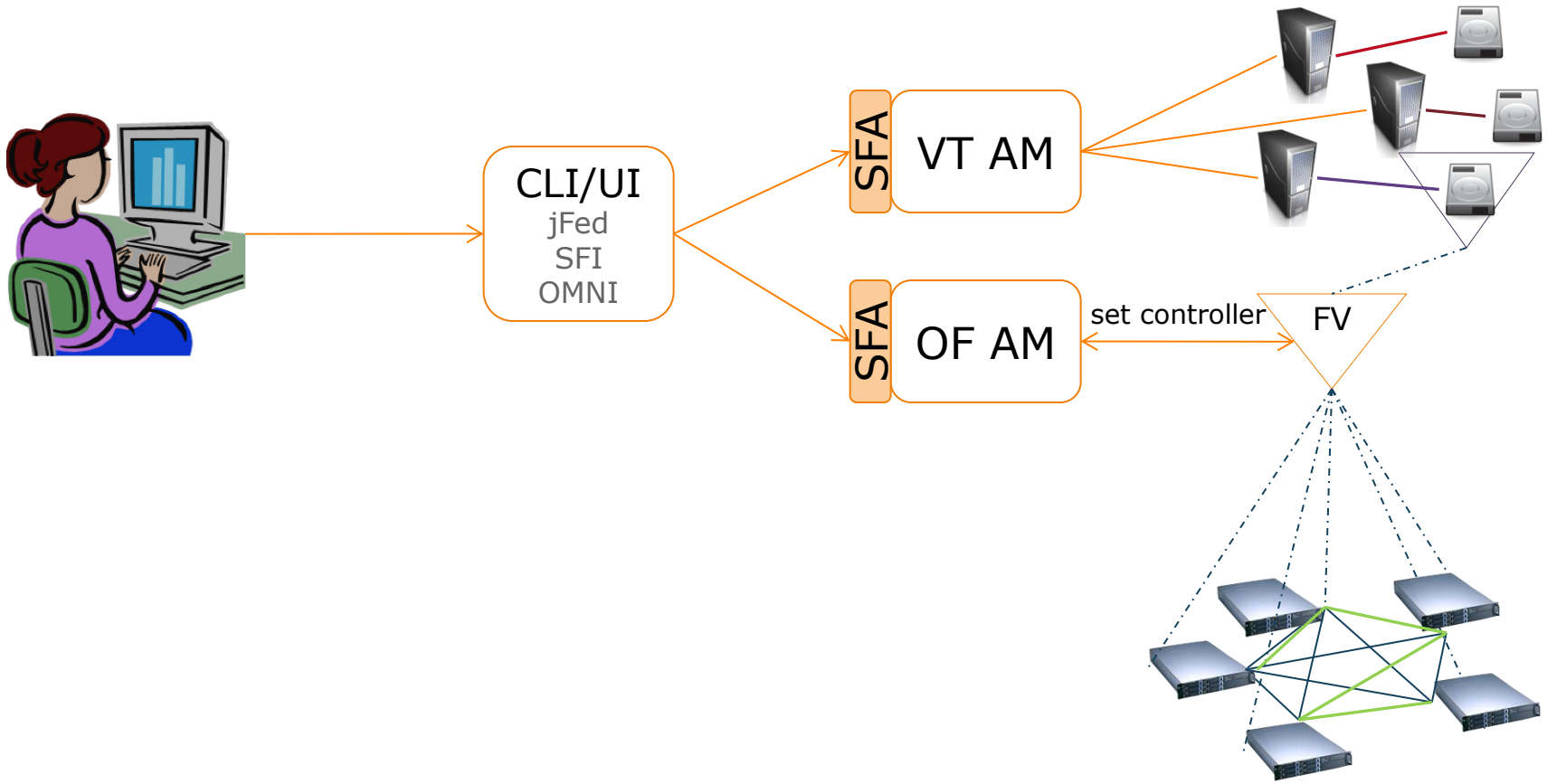
Running an experiment

1. Creating the slivers



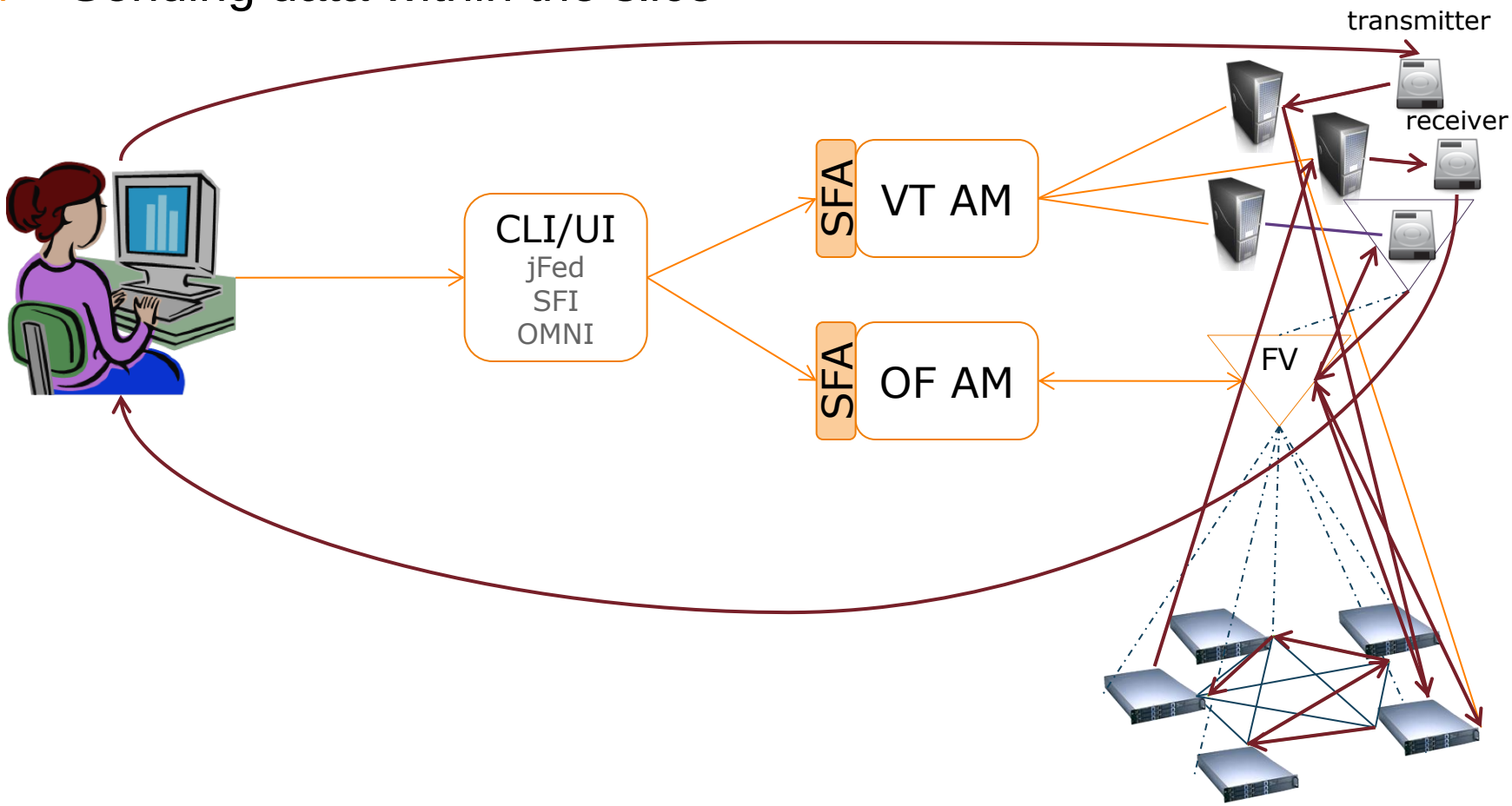
Running an experiment

1. Creating the slivers
2. Setting the controller



Running an experiment

1. Creating the slivers
2. Setting the controller
3. Sending data within the slice



Let's Create a Slice

1. Create the Vwall Node using jFED
2. Create the i2CAT VMs
3. Create the univbris VMs
4. Reserve FS at i2CAT
5. Reserve FS at Bristol

Supporting files

- Download and unzip the file at:
  - http://univbrisofeliaf4f.blogs.ilrt.org/fed4fire-geni-summer-school/

- Let's take a look at the topology ☺

Creating a Node on vwall2 using jFed

- http://jfed.iminds.be
- Quick Start
- Login
- Open
  - jFedi2CAT.rspec
- Run

Creating VM at i2CAT

- Edit file i2catVM.rspec
  - <name>tutorialVM</name> ← alphanumeric name for the virtual machine
- Execute
  - omni createslice <SLICENAME>
  - omni -a https://137.222.204.27:5001/xmlrpc/sfa/ createsliver <SLICENAME> i2catVM.rspec
- Save the IP address, it will be the controller

Creating VM at univbris

- Edit univbrisVM.rspec
  - <name>tutorialVM</name> ← alphanumeric name for the virtual machine
- Execute
  - omni -a https://137.222.204.27:8445/xmlrpc/sfa/ createsliver <SLICENAME> univbrisVM.rspec

Reserving the FS at i2cat

- Edit i2catOF.rspec
  - <openflow:controller url="tcp:10.216.12.29:6633" type="primary"/> ← replace controller IP by the previous one returned by i2CAT manifest RSpec
  - <openflow:dl_vlan value="778" /> ← vlan number
- Execute
  - omni -a https://137.222.204.27:5005/xmlrpc/sfa/ createsliver <SLICENAME> i2catOF.rspec

CAUTION: Do not use the same VLAN and modify the controller IP by one of the IPs obtained in manifest RSpecs

Reserving the FS at univbris

- Edit univbrisOF.rspec
  - <openflow:controller url="tcp:10.216.12.29:6633" type="primary"/> ← replace controller IP by the previous one returned by i2CAT manifest RSpec
  - <openflow:dl_vlan value="778" /> ← vlan number
- Execute
  - omni -a https://137.222.204.27:3626/sfa/2/ createsliver <SLICENAME> univbrisOF.rspec

CAUTION: Do not use the same VLAN and modify the controller ip by one of the IPs obtained in manifest RSpecs

Access & Configure the VMs

- Open console in vwall node
- ssh the VMs using root@vm_ip
- apt-get install vlan
- ifconfig <iface> up (for this tutorial: eth1)
- vconfig add <iface> <vlan>
- ifconfig <iface>.<vlan> 192.168.12.X/24 up
- ifconfig <iface>.<vlan> mtu 1496

Access and configure the Controller

1. access to the vwall node
2. ssh to the controller (root:openflow)
3. apt-get update
4. apt-get install git
5. git clone https://github.com/noxrepo/pox.git
6. git checkout dart
7. cd pox/
8. ./pox.py forwarding.l2_learning

Try ping between nodes

- Ping, ping, ping
  - from virtual wall to univbris VM

You may have to add manually the ARP table at the iMinds node.

arp –s srcIP srcMAC

(this is not related with the OF experiment, is related with the QinQ link)

POX overview

- To run an app:
- ./pox.py <python path to app name>
  - --port
  - --verbose
  - --loglevel

My first App

- In /pox directory, create a *.py file

- Import (at least) nox core and OF lib and some utils
  - **from** pox.core **import** core
  - **import** pox.openflow.libopenflow_01 **as** of
  - **from** pox.lib.util **import** dpid_to_str

- Set up the launch() method

```
def launch (**kwargs):
  core.registerNew(<MyApp>,kwargs)
```

- Develop your app

POX tricks – Listen events

1. Register callback functions for specific events thrown by either the OpenFlow handler module or specific modules like Topology Discovery
   - From the launch or from the init of a class, perform *core.openflow.addListenerByName("EVENTNAME", CALLBACK_FUNC, PRIORITY)* [1]

2. Register object with the OpenFlow handler module or specific modules like Topology Discovery
   - From typically the init of a class, perform *addListeners(self)*. Once this is added, the controller will look for a function with the name *_handle_EVENTNAME(self, event)*. This method is automatically registered as an event handler. [1]

POX tricks – Parse packets

- From a handled event (i. e. packet-in) POX provides ways to parse the packet and identify the headers.

```
packet = event.parsed
mac = packet.src
packet.type [ipv4, icpm, tcp, etc.]
```

POX tricks – Send messages

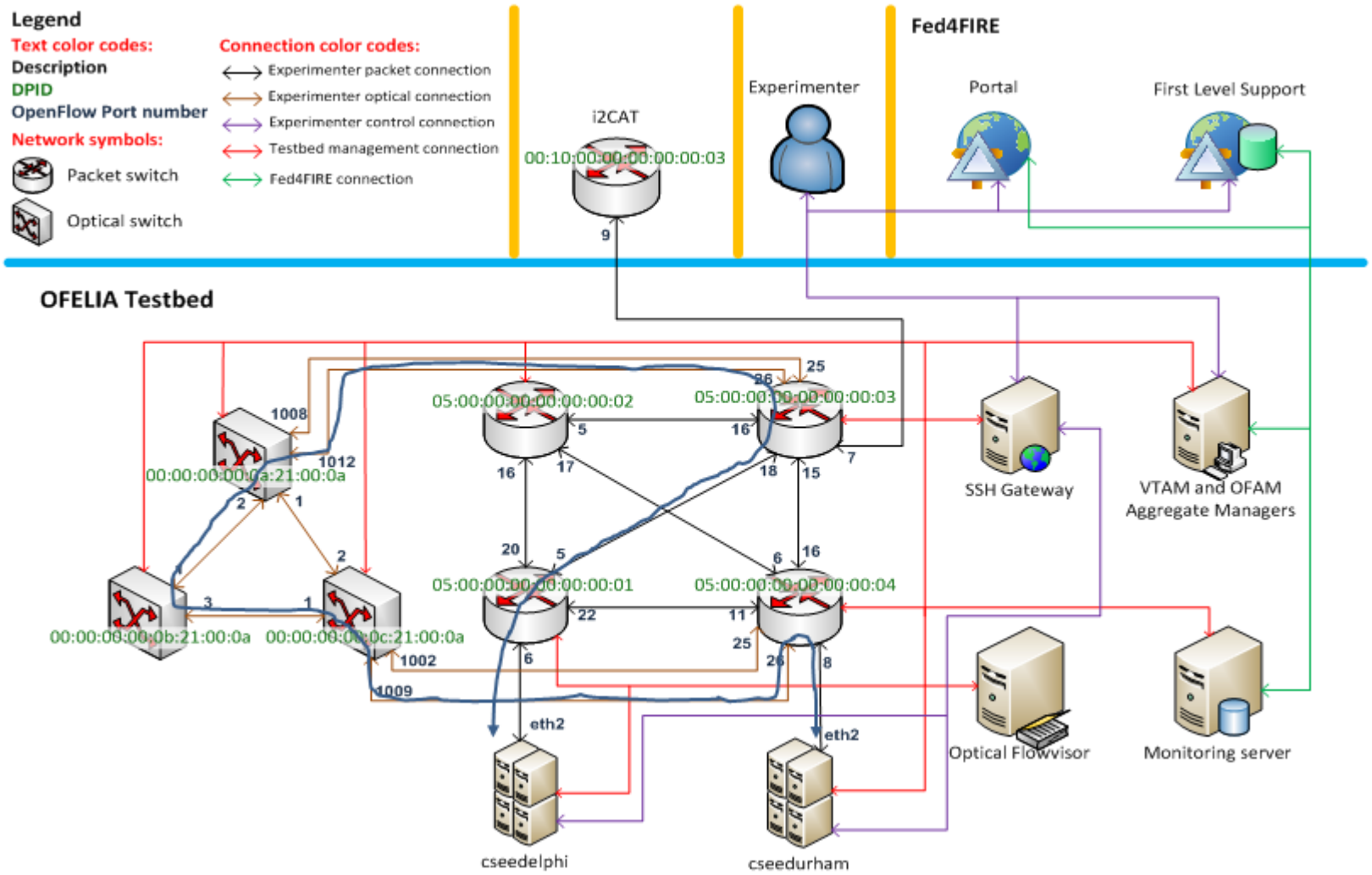
```
msg = of.ofp_packet_out()
message msg.buffer_id = event.ofp.buffer_id
msg.in_port = packet_in.in_port
msg.match = of.ofp_match.from_packet(packet)
action = of.ofp_action_output(port =
of.OFPP_FLOOD)
msg.actions.append(action)
self.connection.send(msg) [1]
```

Advices

Slide powered by master Yoda

- Read ALL samples
- Catch ALL exceptions
- Log ALL with info
- Use existing apps
- Remember that POX can load several apps at the same time
- If possible, apply TDD. Ensure that the code of your app is almost bullet-proof

Optical OpenFlow Demo Introduction



Optical Flowvisor Demo WorkFlow

1. NOX controller with optical application and normal L2 switch application
2. Reservation of a combined optical and packet flowspace
3. Perform cross-connections in the optical domain
4. Test ping and iperf through the combined optical flowspace

Documentation

http://univbrisofeliaf4f.blogs.ilrt.org/

References

- [1] http://sdnhub.org/tutorials/pox/
- https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-FAQs
- http://blog.pythonicneteng.com/2013/02/openflow-tutorial-with-pox.html

CONTROLLER

OpenFlow

SWITCH

Questions?

Acknowledgement

- This work was carried out with the support of the Fed4FIRE-project (“Federation for FIRE"), an Integrated project receiving funding from the European Union’s Seventh Framework Programme for research, technological development and demonstration under grant agreement no 318389
- It does not necessarily reflect the views of the European Commission. The European Commission is not liable for any use that may be made of the information contained herein.

Backup Slides

2. Software: Known issues on controllers

- Entities
  - FlowVisor (island controller)
  - Ryu, NOX, POX, FloodLight (user controller
- Known issues
  - FlowVisor
    - LLDP discovery sends malformed packets (junk data instead of real TLVs)
  - Ryu
    - If exceptions are not properly handled all apps will break
  - POX
    - High resource consumption
  - FloodLight
    - LLDP spam sent from the GUI makes FlowVisor to lose connection with the switches.

3. Hardware

- NEC IP8800/S3640-24T2XW
  - Supports OpenFlow 1.0
  - 24 ports (1/10GE)
  - Max. switching capacity: 88 Gbps
  - Max. packet processing performance: 65.5 Mpackets/s
  - Ports:
    - 16 x 1Gbit 1000BASE-T